2.1 Aria Class Hierarchy ............................................ 27

3 Aria Compound Index ........................................... 31
3.1 Aria Compound List ............................................. 31

4 Aria Class Documentation ......................................... 37
4.1 ArAction Class Reference ....................................... 37
4.2 ArActionAvoidFront Class Reference ......................... 41
4.3 ArActionAvoidSide Class Reference ......................... 43
4.4 ArActionBumpers Class Reference ............................. 45
4.5 ArActionConstantVelocity Class Reference .................. 47
4.6 ArActionDesired Class Reference .............................. 49
4.7 ArActionDesiredChannel Class Reference ................... 55
4.8 ArActionGoto Class Reference .................................. 56
4.9 ArActionGroup Class Reference ............................... 58
4.10 ArActionGroupInput Class Reference ....................... 61
4.11 ArActionGroupStop Class Reference ......................... 62
4.12 ArActionGroupTeleop Class Reference ...................... 63
4.13 ArActionGroupWander Class Reference ...................... 64
4.14 ArActionInput Class Reference ................................ 65
4.15 ArActionJoydrive Class Reference ......................... 67
4.16 ArActionKeydrive Class Reference ......................... 70
4.17 ArActionLimiterBackwards Class Reference .................. 73
4.18 ArActionLimiterForwards Class Reference ................... 75
4.19 ArActionLimiterTableSensor Class Reference ............... 77
4.20 ArActionStallRecover Class Reference ...................... 79
4.21 ArActionStop Class Reference ............................... 81
4.22 ArActionTurn Class Reference ................................ 83
4.23 ArACTS_1_2 Class Reference .................................. 85
4.24 ArACTSBlob Class Reference .................................. 90
4.25 ArAMPTU Class Reference ..................................... 92
4.26 ArAMPTUCommands Class Reference ......................... 95
| 4.27 | ArAMPTUPacket Class Reference | 97 |
| 4.28 | ArArg Class Reference | 99 |
| 4.29 | ArArgumentBuilder Class Reference | 102 |
| 4.30 | ArArgumentParser Class Reference | 103 |
| 4.31 | ArASyncTask Class Reference | 105 |
| 4.32 | ArBasePacket Class Reference | 107 |
| 4.33 | ArCommands Class Reference | 113 |
| 4.34 | ArCondition Class Reference | 116 |
| 4.35 | ArDeviceConnection Class Reference | 118 |
| 4.36 | ArDPPTU Class Reference | 124 |
| 4.37 | ArDPPTUCommands Class Reference | 130 |
| 4.38 | ArDPPTUPacket Class Reference | 132 |
| 4.39 | ArFunctor Class Reference | 133 |
| 4.40 | ArFunctor1 Class Template Reference | 135 |
| 4.41 | ArFunctor1C Class Template Reference | 137 |
| 4.42 | ArFunctor2 Class Template Reference | 141 |
| 4.43 | ArFunctor2C Class Template Reference | 144 |
| 4.44 | ArFunctorC Class Template Reference | 150 |
| 4.45 | ArGlobalFunctor Class Reference | 153 |
| 4.46 | ArGlobalFunctor1 Class Template Reference | 155 |
| 4.47 | ArGlobalFunctor2 Class Template Reference | 158 |
| 4.48 | ArGlobalRetFunctor Class Template Reference | 163 |
| 4.49 | ArGlobalRetFunctor1 Class Template Reference | 165 |
| 4.50 | ArGlobalRetFunctor2 Class Template Reference | 168 |
| 4.51 | ArGripper Class Reference | 172 |
| 4.52 | ArGripperCommands Class Reference | 179 |
| 4.53 | Aria Class Reference | 181 |
| 4.54 | ArInterpolation Class Reference | 186 |
| 4.55 | ArIrrfDevice Class Reference | 188 |
| 4.56 | ArJoyHandler Class Reference | 190 |
| 4.57 | ArKeyHandler Class Reference | 195 |
| 4.89  | ArRobotPacketReceiver Class Reference | 342  |
| 4.90  | ArRobotPacketSender Class Reference  | 345  |
| 4.91  | ArRobotParams Class Reference       | 349  |
| 4.92  | ArSectors Class Reference           | 352  |
| 4.93  | ArSensorReading Class Reference     | 353  |
| 4.94  | ArSerialConnection Class Reference  | 358  |
| 4.95  | ArSick Class Reference              | 364  |
| 4.96  | ArSickLogger Class Reference        | 376  |
| 4.97  | ArSickPacket Class Reference        | 378  |
| 4.98  | ArSickPacketReceiver Class Reference| 381  |
| 4.99  | ArSignalHandler Class Reference     | 384  |
| 4.100 | ArSocket Class Reference            | 390  |
| 4.101 | ArSonarDevice Class Reference       | 396  |
| 4.102 | ArSonyPacket Class Reference        | 398  |
| 4.103 | ArSonyPTZ Class Reference           | 400  |
| 4.104 | ArSyncTask Class Reference          | 403  |
| 4.105 | ArTaskPool Class Reference          | 408  |
| 4.106 | ArTaskState Class Reference         | 410  |
| 4.107 | ArTcpConnection Class Reference     | 411  |
| 4.108 | ArThread Class Reference            | 417  |
| 4.109 | ArTime Class Reference              | 421  |
| 4.110 | ArTransform Class Reference         | 423  |
| 4.111 | ArTypes Class Reference             | 426  |
| 4.112 | ArUtil Class Reference              | 427  |
| 4.113 | ArVCC4 Class Reference              | 436  |
| 4.114 | ArVCC4Commands Class Reference      | 440  |
| 4.115 | ArVCC4Packet Class Reference        | 442  |
Chapter 1

ARIA overview

1.1 Introduction

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.
1.2 ARIA, 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.

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 WorldLink, 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 newslst 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/

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

1.4.2 Other ARIA Files of Note

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARIA.dsp</td>
<td>MSVC++ project file for building the ARIA libraries and examples</td>
</tr>
<tr>
<td>ARIA.dsw</td>
<td>Associated MSVC++ workspace for building ARIA and examples</td>
</tr>
<tr>
<td>Makefile</td>
<td>Linux makefile for building ARIA and examples</td>
</tr>
<tr>
<td>Makefile.dep</td>
<td>Linux dependency</td>
</tr>
<tr>
<td>run</td>
<td>Linux-only; builds and executes your ARIA application</td>
</tr>
<tr>
<td>tests/</td>
<td>Test files, somewhat esoteric but useful during ARIA development</td>
</tr>
<tr>
<td>utils/</td>
<td>Utility commands, not generally needed</td>
</tr>
<tr>
<td>advanced/</td>
<td>Advanced demos, not for the faint of heart (or ARIA novice)</td>
</tr>
</tbody>
</table>

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. 118) class, at the behest of your application code, establishes and maintains a communication channel with the robot server, packaging commands to (ArRobotPacketSender (p. 345)) and decoding responses (ArRobotPacketReceiver (p. 342)) from the robot in safe and reliable packet formats (ArRobotPacket (p. 340)) prescribed by the client-server protocols.

At its heart, ARIA’s ArRobot (p. 290) 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. 290)’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. 290)’s synchronous tasks (ArSyncTask (p. 403)) 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. 290)’s convenient sensor interp (ArRobot::addSensorInterpTask (p. 309)) and user task (ArRobot::addUserTask (p. 309)) 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 onboard servers and devices.

1.7.1 Connecting with a Robot or the Simulator

ArDeviceConnection (p. 118) is ARIA’s communications object; ArSerialConnection (p. 358) and ArTcpConnection (p. 411) 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. 358), for instance, to also configure a serial port and establish a connection with a robot accessory, such as with the SICK laser range finder.

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. 118) 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. 332) 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. 411) con;
ArRobot (p. 290) 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:
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. 120) 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. 312) or `ArRobot::asyncConnect` (p. 310), for example, to establish the client-server connection between ARIA `ArRobot` (p. 290) and the ActivMedia robot microcontroller or SRIsim simulated server. The blockingConnect method doesn't return from the call until a connection succeeds or fails:

```c
robot.setDeviceConnection(&con);
if (!robot.blockingConnect())
{
    printf("Could not connect to robot... Exiting.");
    Arria::shutdown();
    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. 411) con; with `ArSerialConnection` (p. 358) 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 `ArDeviceConnection::read` (p. 122) and `ArDeviceConnection::write` (p. 122). Simple enough. `ArDeviceConnection::close` (p. 120) also is inherited and important. You probably won’t use direct read or write to the robot device, although you could. Rather, `ArRobot` (p. 290) 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. 118) subclasses have support for timestamping (`ArDeviceConnection::getTimeRead` (p. 121)). 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. 290) 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. `ArRobot` (p. 290) 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. 290) 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. 290)’s State Reflection (see State Reflection (p. 9) 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. 290)’s standard SIP handler automatically runs as an ArRobot (p. 290) synchronized task. Other SIP handlers are built in, but your client must 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. 308). ArListPos (p. 199) keeps track of the order by which ArRobot (p. 290) calls each handler. When run, your packet handler must test the SIP type (ArRobotPacket::getID (p. 340)) 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. 11) 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. 11) 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. 290).
1.8.4 Robot-ARIA Synchronization

**ArRobot** (p. 290) 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. 290) 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. 319) and **ArRobot::setCycleTime** (p. 331), respectively. **ArRobot** (p. 290) waits up to twice that cycle time for a standard SIP before cycling automatically.

**ArRobot** (p. 290)’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 **ArTaskState::State** (p. 410) 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. 16)) and then add your sensor interpreter (**ArRobot::addSensorInterpTask** (p. 309)) or user task (**ArRobot::addUserTask** (p. 309)) to the list of syncTasks. These tasks can be removed; use **ArRobot::remSensorInterpTask** (p. 329) or **ArRobot::remUserTask** (p. 329) 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. 290) task list, as well as leaves on the branches. Do these things by getting the root of the tree with **ArRobot::getSyncTaskRoot** (p. 322), and then using the **ArSyncTask** (p. 403) class to do the desired manipulation.

You may disassociate **ArRobot** (p. 290)’s syncTask from firing when the standard SIP is received, through **ArRobot::setCycleChained** (p. 302). But in doing so, you may degrade robot performance, as the robot’s cycle will simply be run once every **ArRobot::getCycleTime** (p. 319) milliseconds.

1.8.5 State Reflection

State reflection in the **ArRobot** (p. 290) 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. 290)
methods for examining these values include \texttt{ArRobot::getPose} (p. 293), \texttt{ArRobot::getX} (p. 293), \texttt{ArRobot::getY} (p. 293), \texttt{ArRobot::getTh} (p. 293), \texttt{ArRobot::getVel} (p. 294), \texttt{ArRobot::getRotVel} (p. 294), \texttt{ArRobot::getBatteryVoltage} (p. 294), \texttt{ArRobot::isLeftMotorStalled} (p. 294), \texttt{ArRobot::isRightMotorStalled} (p. 294), \texttt{ArRobot::getCompass} (p. 295), \texttt{ArRobot::getAnalogPortSelected} (p. 295), \texttt{ArRobot::getAnalog} (p. 295), \texttt{ArRobot::getDigIn} (p. 295), \texttt{ArRobot::getDigOut} (p. 295).

The standard SIP also contains low-level sonar readings, which are reflected in \texttt{ArRobot} (p. 290) and examined with the methods: \texttt{ArRobot::getNumSonar} (p. 296), \texttt{ArRobot::getSonarRange} (p. 321), \texttt{ArRobot::isSonarNew} (p. 324), \texttt{ArRobot::getSonarReading} (p. 321), \texttt{ArRobot::getClosestSonarRange} (p. 296), \texttt{ArRobot::getClosestSonarNumber} (p. 297). This information is more useful when applied to a range device; see \textbf{Range Devices} (p. 10) for details. And read the link pages for \texttt{ArRobot} (p. 290) state reflection method details.

ARIA’s \texttt{ArRobot} (p. 290) also, by default, reflects in the State Reflection \textbf{Robot-ARIA Synchronization} (p. 9) syncTask the latest client Motion Command to the robot server at a rate set by \texttt{ArRobot::setStateReflectionRefreshTime} (p. 334). If no command is in effect, the \texttt{ArCommands::PULSE} (p. 113) 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 \texttt{ArRobot::ArRobot} (p. 305) 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 \texttt{ArRobot::getStateReflectionRefreshTime} (p. 322) milliseconds to prevent the watchdog from timing out.

### 1.9 Range Devices

Range devices (\texttt{ArRangeDevice} (p. 256)) are abstractions of sensors for which there are histories of relevant readings. Currently, there are two ARIA RangeDevices: sonar (\texttt{ArSonarDevice} (p. 396)) and the SICK laser (\texttt{ArSick} (p. 364)). 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 \texttt{ArRobot::addRangeDevice} (p. 297) and remove it with \texttt{ArRobot::remRangeDevice} (p. 328). Query for RangeDevices with \texttt{ArRobot::findRangeDevice} (p. 317). \texttt{ArRobot::hasRangeDevice} (p. 323) 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. 320).

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. 290) 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. 250)): current and cumulative, and each support two different reading formats: box and polar (ArRangeDevice::currentReadingPolar (p. 261), ArRangeDevice::currentReadingBox (p. 260), ArRangeDevice::cumulativeReadingPolar (p. 259), ArRangeDevice::cumulativeReadingBox (p. 259)). The current buffer contains the most recent reading; the cumulative buffer contains several readings over time, limited by `ArRangeBuffer::setSize` (p. 255).

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. 314), `ArRobot::checkRangeDevicesCurrentBox` (p. 313), `ArRobot::checkRangeDevicesCumulativePolar`, `ArRobot::checkRangeDevicesCumulativeBox` (p. 312).

Note that each range device also has a threading mutex (ArRangeDevice::lockDevice (p. 261) and ArRangeDevice::unlockDevice (p. 262)) 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 checkRangeDevice 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. 18).

### 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 \texttt{ArRobot} (p. 290). 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 \texttt{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 \texttt{ArRobot::com} (p. 315) for commands that have no argument, such as \texttt{PULSE}; \texttt{ArRobot::comInt} (p. 315) for a 2-byte integer argument, signed or unsigned, such as the motors \texttt{ENABLE} command; \texttt{ArRobot::com2Bytes} (p. 315) for when you want to define each of the two bytes in the argument, such as the \texttt{VEL2} command; and \texttt{ArRobot::comStr} (p. 315) or \texttt{ArRobot::comStrN} (p. 316) for a null-terminated or defined-length (N extra argument) string argument, respectively, such as the sonar \texttt{POLLING} sequencing command.

The \texttt{ArCommands} (p. 113) class contains an enum with all the direct commands; \texttt{ArCommands::ENABLE} (p. 113), 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 \texttt{ArRobot} (p. 290)’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 (\texttt{ArRobot::setVel2} (p. 335), \texttt{ArRobot::setVel} (p. 335), \texttt{ArRobot::setRotVel} (p. 334), respectively); change the robot’s absolute or relative heading (\texttt{ArRobot::setHeading} (p. 333) or \texttt{ArRobot::setDeltaHeading} (p. 331), respectively); move a prescribed distance (\texttt{ArRobot::move} (p. 325)); or just stop (\texttt{ArRobot::stop} (p. 336)).

Examine the directMotionDemo.cpp example file 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 \texttt{ArRobot::clearDirectMotion} (p. 314) to cancel the overriding effect of a Motion Command so that your Action is able to regain control the robot.
1.10 Commands and Actions

Or limit the time a Motion Command prevents other motion actions with \texttt{ArRobot::setDirectMotionPrecedenceTime} (p. 332). Otherwise, the Motion Command will prevent actions forever. Use \texttt{ArRobot::getDirectMotionPrecedenceTime} (p. 319) 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. \texttt{ArAction} (p. 37) is the base class; \texttt{ArAction::fire} (p. 40) 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 \texttt{ArAction} (p. 37) 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 \texttt{ArRobot::addAction} (p. 306), including a priority which determines its position in the action list. \texttt{ArAction::setRobot} (p. 39) 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 resolver in descending order of priority (lowest priority goes last) in each \texttt{ArRobot} (p. 290) syncTask cycle just prior to State Reflection. The resolver goes through the actions to find a single end actionDesired (\texttt{ArActionDesired} (p. 49)). 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 \texttt{ArActionDesired.reset()} to the actions if it did not want the actions to know about its state.

1.10.4 Action Desired

\texttt{ArActionDesired} (p. 49) is the meat of actions. Desired actions should be reset (\texttt{ArActionDesired::reset} (p. 49)) before they are used or reused.
There are six desired action channels: velocity (`ArActionDesired::setVel` (p. 54)), relative heading (`ArActionDesired::setDeltaHeading` (p. 53)), absolute heading (`ArActionDesired::setHeading` (p. 53)), maximum forward translational velocity (`ArActionDesired::setMaxVel` (p. 54)), maximum reverse translational velocity (`ArActionDesired::setMaxNegVel` (p. 53)), and maximum rotational velocity (`ArActionDesired::setMaxRotVel` (p. 53)).

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 actionDesired 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 (`ArActionDesired::merge` (p. 52)) and averaged (`ArActionDesired::startAverage` (p. 54), `ArActionDesired::addAverage` (p. 52), `ArActionDesired::endAverage` (p. 52)).

### 1.10.5 Resolvers

`ArResolver` (p. 269) is the base action-resolver class. `ArPriorityResolver` (p. 243) is the default resolver. `ArResolver::resolve` (p. 269) is the function that `ArRobot` (p. 290) calls with the action list (actually `ArResolver::ActionMap` (p. 269)) 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::setResolver` (p. 301). 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 adaptative 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:
1.10 Commands and Actions

<table>
<thead>
<tr>
<th>step</th>
<th>action</th>
<th>priority</th>
<th>Desired::setVel</th>
<th>strength</th>
<th>currentDesired</th>
<th>strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>-400</td>
<td>0.25</td>
<td>-400</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>-100</td>
<td>1.0</td>
<td>(combine to 2&amp;3)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>200</td>
<td>0.50</td>
<td>(combine to 2&amp;3)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2&amp;3</td>
<td>3</td>
<td>0</td>
<td>0.75</td>
<td>-100</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>0.50</td>
<td>-100</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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. 37) 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 need 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. 79)) 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. 41) and ArActionConstantVelocity (p. 47) interact.

1.11 Robot Callbacks

There are a number of useful callbacks in the ARIA system, including ArRobot::addConnectCB (p. 306), ArRobot::remConnectCB (p. 327), ArRobot::addFailedConnectCB (p. 308), ArRobot::remFailedConnectCB (p. 328), ArRobot::addDisconnectNormallyCB (p. 307), ArRobot::remDisconnectNormallyCB (p. 327), ArRobot::addDisconnectOnErrorCB (p. 307), ArRobot::remDisconnectOnErrorCB (p. 327), ArRobot::addRunExitCB (p. 308), ArRobot::remRunExitCB (p. 329). Read their individual documentation pages for details.

Examples of callbacks are in the directMotionDemo and in joydriveThreaded. Also, ArGripper (p. 172) 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. 133). This is the base class for all the different functors. Its for a function that has no parameters and no return value.
1.13 User Input

But you cannot create an *ArFunctor* (p. 133), because it is an abstract base class. Rather, you need to instantiate one of these classes:

*ArFunctorC* (p. 150), *ArFunctor1C* (p. 137), *ArFunctor2C* (p. 144), *ArRetFunctorC* (p. 287), *ArRetFunctor1C* (p. 275), *ArRetFunctor2C* (p. 281)

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 <ExampleClass> functor(obj, &ExampleClass::aFunction);
```

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. 133), *ArFunctor1* (p. 135), *ArFunctor2* (p. 141), *ArRetFunctor* (p. 271)<ReturnType>, *ArRetFunctor1* (p. 273)<ReturnType>, and *ArRetFunctor2* (p. 279)<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. 181), 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 *ArJoyHandler* (p. 190), the important functions are *ArJoyHandler::getButtons, ArJoyHandler::getAdjusted* (p. 192), *ArJoyHandler::setSpeeds* (p. 190), and *ArJoyHandler::getDoubles* (p. 193). With a keyboard is most useful for setting and changing modes, and exiting the program, but it can also be used to drive the robot well (with the arrow keys and the space bar typically). *ArKeyHandler* (p. 195) 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. 195) is *ArKeyHandler::addKeyHandler* (p. 197), which binds a specific key to a given functor, also look at the enum *ArKeyHandler::KEY* (p. 196) for values to pass in for special keys. You
also need to attach a key handler to some robot with ArRobot::attachKey-Handler (p. 311). ArActionJoydrive (p. 67) will use the joystick to drive the robot around, while ArActionKeydrive (p. 70) 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. 195) to a robot, escape will exit the program, however you can change 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. 105), ArCondition (p. 116), ArMutex (p. 227), ArPeriodicTask (p. 229), ArSemaphore, ArTaskPool (p. 408), and ArThread (p. 417).

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. 227), ArSemaphore, and ArCondition (p. 116). ArMutex (p. 227) 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 may become deadlocked and hang. See the mutex example in the ARIA distribution for more details.

ArCondition (p. 116) and ArSemaphore are useful for delaying the execution of a thread. A thread suspends execution while waiting on an ArSemaphore or ArCondition (p. 116) until another thread wakes it up. For instance, use ArSemaphore or ArCondition (p. 116) while waiting for a mutex to become
1.14 ARIA Threading

free. Performance is better, too. ArSemaphore and \textbf{ArCondition} (p. 116) put
the thread to sleep. The processing expensive alternative is to have the thread
continuously check for a change in condition.

ArSemaphores are more thread-safe than ArConditions in that a semaphore
keeps track of the number of threads that use it. \textbf{ArCondition} (p. 116) notifies
only those threads that are currently waiting on it at the time the condition
changes. A semaphore counts how many times its been woken up and keeps
track of the fact that it needs to wake up that many threads. So, when a thread
goes to wait on the semaphore, the call to wait() will not block because the
condition already has been met.

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 \textbf{ArASyncTask} (p. 105) needs to have a thread
under its control for the full lifetime of the program.

To create an asynchronous task, derive a class from \textbf{ArASyncTask} (p. 105)
and override the \textbf{ArASyncTask::runThread()} (p. 106) function. (The func-
tion automatically is called within the new thread, when the \textbf{ArASync-
Task} (p. 105) gets created.) To create and start the thread, call \textbf{ArASync-
Task::create()} (p. 105). When the \textbf{ArASyncTask::runThread()} (p. 106)
function exits, the thread will exit and be destroyed.

This class is mainly a convenience wrapper around \textbf{ArThread} (p. 417) so that
you can easily create your own object that encapsulates the concept of a thread.

1.14.3 Task Pool

The \textbf{ArTaskPool} (p. 408) class is a structure that contains a pool of threads
which can be allocated to run ArPeriodicTasks. This is useful for tasks that
need to be run in their own thread, but you don’t want to have the overhead of
creating and destroying many threads. The \textbf{ArTaskPool} (p. 408) class manages
the threads for you.

\textbf{ArTaskPool} (p. 408) uses the singleton design pattern; there is one and only
one instance of the \textbf{ArTaskPool} (p. 408). That instance can be reached through
the ArTaskPool::getPool() function.

To use \textbf{ArPeriodicTask} (p. 229), create a class which derives from \textbf{Ar-
PeriodicTask} (p. 229) and override the \textbf{ArPeriodicTask::runTask()} (p. 229).
The user defined runTask() function simply needs to do the task and exit when
its done. This function will be called within its own thread. The conventions on
how to handle data access between multiple threads should be observed within
the runTask() code and any functions that it calls.

Once the user-defined class is created, instantiate an object and pass it to the
ArTaskPool::runTask() (p. 408) function. This will cause the task to be run
immediately in its own thread.

1.15 ARIA Global Data

ARIA contains a list of all the ArRobot (p. 290) 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. 181) or
/usr/local/Aria (p. 181), 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. 181) 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. 107) and ArRobotPacket (p. 340)–the basic
algorithms for constructing command packets and decoding server information
packets.

Above the packet layer is the packet handler classes, ArRobotPacketReceiver
(p. 342) and ArRobotPacketSender (p. 345), when send and receive packets
to and from the robot. Finally, on top of all these lowest layers is ArRobot
(p. 290), 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. 290) has builtin
tasks, actions, state reflection and so forth, all of which can be disabled from
the constructor (ArRobot::ArRobot (p. 305)) and ignored or reimplemented.

Also note that if all you do is turn off state reflection, which only affects sending
ArRobot (p. 290)-mediated motion commands to the robot, not receiving SIPs
from the robot, none of the other activities which ArRobot (p. 290) 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. 325)
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. 303), so that the loop
counter will be updated. You could also just call ArRobot::packetHandler
1.17 Robot Parameter Files

Found in the Aria (p. 181)/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:

```
@end
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. 202), **ArUtil** (p. 427), **ArTime** (p. 421), **ArPose** (p. 231), and **ArSectors** (p. 352).

### 1.19 Sockets

The **ArSocket** (p. 390) class is a wrapper around the socket network communication layer of your operating system. ARIA mostly uses **ArSocket** (p. 390)
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:

```cpp
ArSocket (p. 390) sock("host.name.com", 4040, ArSocket::TCP);
```

Or call the `ArSocket::connect()` (p. 390) function, such as:

```cpp
ArSocket (p. 390) sock;
sock.connect("host.name.com", 4040, ArSocket::TCP);
```

To open a server port, simple construct a socket:

```cpp
ArSocket (p. 390) sock(4040, true, ArSocket::TCP);
```

Or call:

```cpp
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.

```lisp
(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 declaration 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 different manners:

```cpp
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 constructor can give arguments to the constructors of the member variables it contains and to the constructors which it inherits. For example if you have:

```cpp
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 initialize 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. 16) 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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArAction</td>
<td>37</td>
</tr>
<tr>
<td>ArActionAvoidFront</td>
<td>41</td>
</tr>
<tr>
<td>ArActionAvoidSide</td>
<td>43</td>
</tr>
<tr>
<td>ArActionBumpers</td>
<td>45</td>
</tr>
<tr>
<td>ArActionConstantVelocity</td>
<td>47</td>
</tr>
<tr>
<td>ArActionGoto</td>
<td>56</td>
</tr>
<tr>
<td>ArActionInput</td>
<td>65</td>
</tr>
<tr>
<td>ArActionJoydrive</td>
<td>67</td>
</tr>
<tr>
<td>ArActionKeydrive</td>
<td>70</td>
</tr>
<tr>
<td>ArActionLimiterBackwards</td>
<td>73</td>
</tr>
<tr>
<td>ArActionLimiterForwards</td>
<td>75</td>
</tr>
<tr>
<td>ArActionLimiterTableSensor</td>
<td>77</td>
</tr>
<tr>
<td>ArActionStallRecover</td>
<td>79</td>
</tr>
<tr>
<td>ArActionStop</td>
<td>81</td>
</tr>
<tr>
<td>ArActionTurn</td>
<td>83</td>
</tr>
<tr>
<td>ArActionDesired</td>
<td>49</td>
</tr>
<tr>
<td>ArActionDesiredChannel</td>
<td>55</td>
</tr>
<tr>
<td>ArActionGroup</td>
<td>58</td>
</tr>
<tr>
<td>ArActionGroupInput</td>
<td>61</td>
</tr>
<tr>
<td>ArActionGroupStop</td>
<td>62</td>
</tr>
<tr>
<td>ArActionGroupTeleop</td>
<td>63</td>
</tr>
<tr>
<td>ArActionGroupWander</td>
<td>64</td>
</tr>
<tr>
<td>ArACTS_1_2</td>
<td>85</td>
</tr>
<tr>
<td>ArACTSBlob</td>
<td>90</td>
</tr>
<tr>
<td>ArAMPTUCommands</td>
<td>95</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
</tr>
<tr>
<td>ArArg</td>
<td>99</td>
</tr>
<tr>
<td>ArArgumentBuilder</td>
<td>102</td>
</tr>
<tr>
<td>ArArgumentParser</td>
<td>103</td>
</tr>
<tr>
<td>ArBasePacket</td>
<td>107</td>
</tr>
<tr>
<td>ArAMPTUPacket</td>
<td>97</td>
</tr>
<tr>
<td>ArDPPTUPacket</td>
<td>132</td>
</tr>
<tr>
<td>ArRobotPacket</td>
<td>340</td>
</tr>
<tr>
<td>ArSickPacket</td>
<td>378</td>
</tr>
<tr>
<td>ArSonyPacket</td>
<td>398</td>
</tr>
<tr>
<td>ArVCC4Packet</td>
<td>442</td>
</tr>
<tr>
<td>ArCommands</td>
<td>113</td>
</tr>
<tr>
<td>ArCondition</td>
<td>116</td>
</tr>
<tr>
<td>ArDeviceConnection</td>
<td>118</td>
</tr>
<tr>
<td>ArSerialConnection</td>
<td>358</td>
</tr>
<tr>
<td>ArTcpConnection</td>
<td>411</td>
</tr>
<tr>
<td>ArDPPTUCommands</td>
<td>130</td>
</tr>
<tr>
<td>ArFunctor</td>
<td>133</td>
</tr>
<tr>
<td>ArFunctor1&lt; P1 &gt;</td>
<td>135</td>
</tr>
<tr>
<td>ArFunctor1C&lt; T, P1 &gt;</td>
<td>137</td>
</tr>
<tr>
<td>ArGlobalFunctor1&lt; P1 &gt;</td>
<td>155</td>
</tr>
<tr>
<td>ArFunctor2&lt; P1, P2 &gt;</td>
<td>141</td>
</tr>
<tr>
<td>ArFunctor2C&lt; T, P1, P2 &gt;</td>
<td>144</td>
</tr>
<tr>
<td>ArGlobalFunctor2&lt; P1, P2 &gt;</td>
<td>158</td>
</tr>
<tr>
<td>ArFunctorC&lt; T &gt;</td>
<td>150</td>
</tr>
<tr>
<td>ArGlobalFunctor</td>
<td>153</td>
</tr>
<tr>
<td>ArRetFunctor&lt; Ret &gt;</td>
<td>271</td>
</tr>
<tr>
<td>ArGlobalRetFunctor&lt; Ret &gt;</td>
<td>163</td>
</tr>
<tr>
<td>ArRetFunctor1&lt; Ret, P1 &gt;</td>
<td>273</td>
</tr>
<tr>
<td>ArGlobalRetFunctor1&lt; Ret, P1 &gt;</td>
<td>165</td>
</tr>
<tr>
<td>ArRetFunctor1C&lt; Ret, T, P1 &gt;</td>
<td>275</td>
</tr>
<tr>
<td>ArRetFunctor2&lt; Ret, P1, P2 &gt;</td>
<td>279</td>
</tr>
<tr>
<td>ArGlobalRetFunctor2&lt; Ret, P1, P2 &gt;</td>
<td>168</td>
</tr>
<tr>
<td>ArRetFunctor2C&lt; Ret, T, P1, P2 &gt;</td>
<td>281</td>
</tr>
<tr>
<td>ArRetFunctorC&lt; Ret, T &gt;</td>
<td>287</td>
</tr>
<tr>
<td>ArGripper</td>
<td>172</td>
</tr>
<tr>
<td>ArGripperCommands</td>
<td>179</td>
</tr>
<tr>
<td>Aria</td>
<td>181</td>
</tr>
<tr>
<td>ArInterpolation</td>
<td>186</td>
</tr>
<tr>
<td>ArJoyHandler</td>
<td>190</td>
</tr>
<tr>
<td>ArKeyHandler</td>
<td>195</td>
</tr>
<tr>
<td>ArListPos</td>
<td>199</td>
</tr>
<tr>
<td>ArLog</td>
<td>200</td>
</tr>
<tr>
<td>ArMath</td>
<td>202</td>
</tr>
<tr>
<td>ArMode</td>
<td>207</td>
</tr>
</tbody>
</table>
### 2.1 Aria Class Hierarchy

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArModeBumps</td>
<td>211</td>
</tr>
<tr>
<td>ArModeCamera</td>
<td>213</td>
</tr>
<tr>
<td>ArModeGripper</td>
<td></td>
</tr>
<tr>
<td>ArModeIO</td>
<td></td>
</tr>
<tr>
<td>ArModeLaser</td>
<td></td>
</tr>
<tr>
<td>ArModePosition</td>
<td></td>
</tr>
<tr>
<td>ArModeSonar</td>
<td>215</td>
</tr>
<tr>
<td>ArModeTeleop</td>
<td>217</td>
</tr>
<tr>
<td>ArModeWander</td>
<td>219</td>
</tr>
<tr>
<td>ArModule</td>
<td>221</td>
</tr>
<tr>
<td>ArModuleLoader</td>
<td>224</td>
</tr>
<tr>
<td>ArMutex</td>
<td>227</td>
</tr>
<tr>
<td>ArPeriodicTask</td>
<td>229</td>
</tr>
<tr>
<td>ArPose</td>
<td>231</td>
</tr>
<tr>
<td>ArPoseWithTime</td>
<td>235</td>
</tr>
<tr>
<td>ArPref</td>
<td>236</td>
</tr>
<tr>
<td>ArPreferences</td>
<td></td>
</tr>
<tr>
<td>ArRobotParamFile</td>
<td></td>
</tr>
<tr>
<td>ArRobotAmigo</td>
<td></td>
</tr>
<tr>
<td>ArRobotGeneric</td>
<td></td>
</tr>
<tr>
<td>ArRobotMapper</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2AT</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2CE</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2D8</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2DF</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2DX</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2DXe</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2IT</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2PB</td>
<td></td>
</tr>
<tr>
<td>ArRobotP2PP</td>
<td></td>
</tr>
<tr>
<td>ArRobotPerfPB</td>
<td></td>
</tr>
<tr>
<td>ArRobotPion1M</td>
<td></td>
</tr>
<tr>
<td>ArRobotPion1X</td>
<td></td>
</tr>
<tr>
<td>ArRobotPionAT</td>
<td></td>
</tr>
<tr>
<td>ArRobotPowerBot</td>
<td></td>
</tr>
<tr>
<td>ArRobotPsos1M</td>
<td></td>
</tr>
<tr>
<td>ArRobotPsos1X</td>
<td></td>
</tr>
<tr>
<td>ArRobotPsos43M</td>
<td></td>
</tr>
<tr>
<td>ArPrefSection</td>
<td></td>
</tr>
<tr>
<td>ArPTZ</td>
<td>244</td>
</tr>
<tr>
<td>ArAMPTU</td>
<td>92</td>
</tr>
<tr>
<td>ArDPPTU</td>
<td>124</td>
</tr>
<tr>
<td>ArSonyPTZ</td>
<td>400</td>
</tr>
<tr>
<td>ArVCC4</td>
<td>436</td>
</tr>
<tr>
<td>ArRangeBuffer</td>
<td>250</td>
</tr>
</tbody>
</table>
ArRangeDevice ........................................ 256
ArIrrfDevice ........................................ 188
ArRangeDeviceThreaded .............................. 264
  ArSick ................................................ 364
ArSonarDevice ...................................... 396
ArResolver ............................................ 269
  ArPriorityResolver ................................. 243
ArRobot ................................................ 290
ArRobotPacketReceiver .............................. 342
ArRobotPacketSender ................................. 345
ArRobotParams ....................................... 349
ArSectors ............................................. 352
ArSemaphore .......................................... 376
ArSensorReading ..................................... 353
ArSickLogger ......................................... 376
ArSickPacketReceiver ............................... 381
ArSocket .............................................. 390
ArSyncTask ............................................ 403
ArTaskPool .......................................... 408
ArTaskState .......................................... 410
ArThread .............................................. 417
  ArAsyncTask ......................................... 105
    ArRangeDeviceThreaded ......................... 264
  ArRecurrentTask .................................. 267
  ArSignalHandler .................................. 384
  ArSyncLoop .........................................
  ArTimer ............................................
ArTaskPoolTask
ArTime .................................................. 421
ArTransform ......................................... 423
ArTypes .............................................. 426
ArUtil .................................................. 427
ArVCC4Commands ..................................... 440
Chapter 3

Aria Compound Index

3.1 Aria Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

- \texttt{ArAction} (Action class, what typically makes the robot move) \ldots 37
- \texttt{ArActionAvoidFront} (This action does obstacle avoidance, controlling both trans and rot) \ldots \ldots \ldots \ldots \ldots \ldots 41
- \texttt{ArActionAvoidSide} (Action to avoid impacts by firing into walls at a shallow angle) \ldots \ldots \ldots \ldots \ldots 43
- \texttt{ArActionBumpers} (Action to deal with if the bumpers trigger) \ldots 45
- \texttt{ArActionConstantVelocity} (Action for going straight at a constant velocity) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 47
- \texttt{ArActionDesired} (Class used to say what movement is desired) \ldots 49
- \texttt{ArActionDesiredChannel} (Class used by \texttt{ArActionDesired} (p. 49) for each channel, internal) \ldots \ldots \ldots \ldots \ldots 55
- \texttt{ArActionGoto} (This action goes to a given \texttt{ArPose} (p. 231) very naively) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 56
- \texttt{ArActionGroup} (Class for groups of actions to accomplish one thing) 58
- \texttt{ArActionGroupInput} (Input to drive the robot) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 61
- \texttt{ArActionGroupStop} (Stop the robot) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 62
- \texttt{ArActionGroupTeleop} (Teleop the robot) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots 63
- \texttt{ArActionGroupWander} (Has the robot wander) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \}
ArActionLimiterForwards (Action to limit the forwards motion of the robot) ........................................ 75
ArActionLimiterTableSensor (Action to limit speed based on whether there the table-sensors see anything) .... 77
ArActionStallRecover (Action to recover from a stall) ...... 79
ArActionStop (Action for stopping the robot) ................. 81
ArActionTurn (Action to turn when the behaviors with more priority have limited the speed) ...................... 83
ArACTS_1_2 (Driver for ACTS) ................................. 85
ArACTSBlob (A class for the acts blob) .......................... 90
ArAMPTU (Driver for the AMPUT) .............................. 92
ArAMPTUCommands (A class with the commands for the AMPTU) 95
ArAMPTUPacket (A class for for making commands to send to the AMPTU) ..................................... 97
ArArg (Argument class, mostly for actions, could be used for other things) ...................................... 99
ArArgumentBuilder (This class is to build arguments for things that require argc and argv) ...................... 102
ArArgumentParser (Class for parsing arguments) ............ 103
ArAsyncTask (Asynchronous task (runs in its own thread)) .. 105
ArBasePacket (Base packet class) ............................... 107
ArCommands (A class with an enum of the commands that can be sent to the robot) .................................. 113
ArCondition (Threading condition wrapper class) ............ 116
ArDeviceConnection (Base class for device connections) ..... 118
ArDPPTU (Driver for the DPPTU) ............................... 124
ArDPPTUCommands (A class with the commands for the DPPTU) 130
ArDPPTUPacket (A class for for making commands to send to the DPPTU) ......................................... 132
ArFunctor (Base class for functors) .............................. 133
ArFunctor1< P1 > (Base class for functors with 1 parameter) ... 135
ArFunctor1C< T, P1 > (Functor for a member function with 1 parameter) .............................................. 137
ArFunctor2< P1, P2 > (Base class for functors with 2 parameters) .................................................. 141
ArFunctor2C< T, P1, P2 > (Functor for a member function with 2 parameters) ....................................... 144
ArFunctorC< T > (Functor for a member function) ............ 150
ArGlobalFunctor (Functor for a global function with no parameters) 153
ArGlobalFunctor1< P1 > (Functor for a global function with 1 parameter) ............................................ 155
ArGlobalFunctor2< P1, P2 > (Functor for a global function with 2 parameters) ...................................... 158
ArGlobalRetFunctor< Ret > (Functor for a global function with return value) ....................................... 163
ArGlobalRetFunctor1< Ret, P1 > (Functor for a global function with 1 parameter and return value) ............ 165
ArGlobalRetFunctor2< Ret, P1, P2 > (Functor for a global function with 2 parameters and return value) .......................... 168
ArGripper (A class of convenience functions for using the gripper) .......................................................... 172
ArGripperCommands (A class with an enum of the commands for the gripper) .................................................. 179
Aria (This class performs global initialization and deinitialization) .......................................................... 181
ArInterpolation .......................................................... 186
ArIrffDevice (A class for connecting to a PB-9 and managing the resulting data) .................................................. 188
ArJoyHandler (Interfaces to a joystick) .......................................................... 190
ArKeyHandler (This class will read input from the keyboard) .......................................................... 195
ArListPos (Has enum for position in list) .......................................................... 199
ArLog (Logging utility class) .......................................................... 200
ArMath (This class has static members to do common math operations) .......................................................... 202
ArMode (A class for different modes, mostly as related to keyboard input) .................................................. 207
ArModeCamera (Mode for controlling the gripper) .......................................................... 211
ArModeGripper (Mode for controlling the gripper) .......................................................... 213
ArModeSonar (Mode for displaying the sonar) .......................................................... 215
ArModeTeleop (Mode for teleoping the robot with joystick + keyboard) .................................................. 217
ArModeWander (Mode for wandering around) .......................................................... 219
ArModule (Dynamically loaded module base class, read warning in more) .................................................. 221
ArModuleLoader (Dynamic ArModule (p. 221) loader) .......................................................... 224
ArMutex (Mutex wrapper class) .......................................................... 227
ArPeriodicTask (A periodic task base class) .......................................................... 229
ArPose (The class which represents a position) .......................................................... 231
ArPoseWithTime (A subclass of pose that also has the time the pose was taken) .................................................. 235
ArPref (Preference instance. Used by ArPreferences) .......................................................... 236
ArPriorityResolver (Default resolver), takes the action list and uses the priority to resolve) .................................................. 243
ArPTZ (Base class which handles the PTZ cameras) .......................................................... 244
ArRangeBuffer (This class is a buffer that holds ranging information) .................................................. 250
ArRangeDevice (The class for all devices which return range info (laser, sonar)) .................................................. 256
ArRangeDeviceThreaded (A range device which can run in its own thread) .................................................. 264
ArRecurentTask (Recurent task (runs in its own thread)) .......................................................... 267
ArResolver (Resolves a list of actions and returns what to do) .......................................................... 269
ArRetFunctor< Ret > (Base class for functors with a return value) .................................................. 271
ArRetFunctor1< Ret, P1 > (Base class for functors with a return value with 1 parameter) .................................................. 273
ArRetFunctor1C< Ret, T, P1 > (Functor for a member function with return value and 1 parameter) .................................................. 275
Aria Compound Index

ArRetFunctor2< Ret, P1, P2 > (Base class for functors with a return value with 2 parameters) ............... 279
ArRetFunctor2C< Ret, T, P1, P2 > (Functor for a member function with return value and 2 parameters) ....... 281
ArRetFunctorC< Ret, T > (Functor for a member function with return value) ........................................ 287
ArRobot (THE important class) ........................................ 290
ArRobotPacket (Represents the packets sent to the robot as well as those received from it) ..................... 340
ArRobotPacketReceiver (Given a device connection it receives packets from the robot through it) ............ 342
ArRobotPacketSender (Given a device connection this sends commands through it to the robot) ............... 345
ArRobotParams (Contains the robot parameters, according to the parameter file) .............................. 349
ArSectors (A class for keeping track of if a complete revolution has been attained) ............................ 352
ArSensorReading (A class to hold a sensor reading, should be one instance per sensor) ....................... 353
ArSerialConnection (For connecting to devices through a serial port) ............................................. 358
ArSick (The sick driver) .............................................. 364
ArSickLogger (This class can be used to create log files for the laser mapper) .................................. 376
ArSickPacket (Represents the packets sent to the sick as well as those received from it) ....................... 378
ArSickPacketReceiver (Given a device connection it receives packets from the sick through it) ............. 381
ArSignalHandler (Signal handling class) .............................. 384
ArSocket (Socket communication wrapper) ......................... 390
ArSonarDevice (A class for keeping track of sonar) ........................................ 396
ArSonyPacket (A class for for making commands to send to the sony) ............. 398
ArSonyPTZ (A class to use the sony pan tilt zoom unit) ........................................ 400
ArSyncTask (Class used internally to manage the functions that are called every cycle) ...................... 403
ArTaskPool (A thread pool for running functions or ArPeriodicTasks) ........................................... 408
ArTaskState (Class with the different states a task can be in) ..................................................... 410
ArTcpConnection (For connecting to a device through a socket) ..................................................... 411
ArThread (POSIX/WIN32 thread wrapper class) ......................... 417
ArTime (A class for time readings) .................................. 421
ArTransform (A class to handle transforms between different coordinates) .................................... 423
ArTypes (Contains platform independent sized variable types) .................................................. 426
ArUtil (This class has utility functions) ........................................................................ 427
ArVCC4 (Driver for the VCC4) ..................................... 436
ArVCC4Commands (A class with the commands for the VCC4) ................................................ 440
3.1 Aria Compound List

ArVCC4Packet (A class for for making commands to send to the VCC4) .................................................. 442
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** (void)
  
  *Destructor.*

- virtual bool **isActive** (void)
  
  *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)
  
  *Find the number of arguments this action takes.*

• virtual **ArArg** ∗ **getArg** (int number)
  
  *Gets the numbered argument.*

• virtual std::string **getName** (void)
  
  *Gets the name of the action.*

• virtual std::string **getDescription** (void)
  
  *Gets the long description of the action.*

• virtual **ArActionDesired** ∗ **getDesired** (void)
  
  *Gets what this action wants to do (for display purposes).*

• virtual void **print** (void)
  
  **ArLog::log** (p. 201) *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. 42), ArActionAvoidSide (p. 44), ArActionBumpers (p. 46), ArActionConstantVelocity (p. 48), ArActionGoto (p. 57), ArActionInput (p. 66), ArActionJoydrive (p. 69), ArActionKeydrive (p. 71), ArActionLimiterBackwards (p. 74), ArActionLimiterForwards (p. 76), ArActionLimiterTableSensor (p. 78), ArActionStallRecover (p. 80), ArActionStop (p. 82), and ArActionTurn (p. 84).

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

- 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::

```
ArAction
   |
   v
ArActionAvoidFront
```

Public Methods

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

  Constructor.

- virtual ~ArActionAvoidFront (void)

  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 useTableSensingIRAvail 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)
- **avoidVelocity** Speed at which to go while avoiding an obstacle. (mm/sec)
- **turnAmount** Degrees to turn relative to current heading while avoiding obstacle (deg)
- **useTableIRIfAvail** 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. 40).

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

- ArActionAvoidFront.h
- ArActionAvoidFront.cpp
4.3 ArActionAvoidSide Class Reference

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

#include <ArActionAvoidSide.h>

Inheritance diagram for ArActionAvoidSide:

```
ArAction
    ├── ArActionAvoidSide
```

Public Methods

- **ArActionAvoidSide** (const char *name=”Avoid side”, double obstacleDistance=300, double turnAmount=5)
  
  Constructor.

- virtual **~ArActionAvoidSide** (void)
  
  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 firing into walls at a shallow angle.

This action watches the sensors to see if it is close to firing 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

\(\text{ArActionDesired currentDesired} \) [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. 40).

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

- ArActionAvoidSide.h
- ArActionAvoidSide.cpp
4.4 ArActionBumpers Class Reference

Action to deal with if the bumpers trigger.

```
#include <ArActionBumpers.h>
```

Inheritance diagram for ArActionBumpers:

```
ArAction
   `-- ArActionBumpers
```

Public Methods

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

- virtual **~ArActionBumpers** (void)
  
  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

```cpp
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 allow 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. 40).

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

- ArActionBumpers.h
- ArActionBumpers.cpp
4.5 ArActionConstantVelocity Class Reference

Action for going straight at a constant velocity.

#include <ArActionConstantVelocity.h>

Inheritance diagram for ArActionConstantVelocity::

```
<table>
<thead>
<tr>
<th>ArActionConstantVelocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArAction</td>
</tr>
</tbody>
</table>
```

Public Methods

- **ArActionConstantVelocity** (const char *name="Constant Velocity", double velocity=400)
  Constructor.
- virtual ~**ArActionConstantVelocity** (void)
  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. 40).

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

- ArActionConstantVelocity.h
- ArActionConstantVelocity.cpp
4.6 ArActionDesired Class Reference

Class used to say what movement is desired.

#include <ArActionDesired.h>

Public Methods

- **ArActionDesired** (void)
  Constructor.

- virtual **~ArActionDesired** (void)
  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).

- virtual void **setMaxVel** (double maxVel, double strength=MAX_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 averaging together different desireds.

• virtual void **addAverage** (ArActionDesired *actDesired)
  
  Adds another actionDesired into the mix to average.
4.6 ArActionDesired Class Reference

- 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.*

- 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 separate. 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. 269)) 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. 269).
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), in-
ternal.
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.

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

- ArActionDesired.h
- ArActionDesired.cpp
4.7 ArActionDesiredChannel Class Reference

Class used by ArActionDesired (p. 49) for each channel, internal.

#include <ArActionDesired.h>

4.7.1 Detailed Description

Class used by ArActionDesired (p. 49) 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

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

- ArActionDesired.h
- ArActionDesired.cpp
4.8 ArActionGoto Class Reference

This action goes to a given ArPose (p. 231) very naively.

```cpp
#include <ArActionGoto.h>
```

Inheritance diagram for ArActionGoto:

```
ArActionGoto
|     |
|     | ArAction
```

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.
4.8 ArActionGoto Class Reference

- 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. 231) very naively.

This action naively drives straight towards a given ArPose (p. 231)... 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. 40).

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

- 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:

```
ArActionGroup
   ArActionGroupInput ArActionGroupStop ArActionGroupTeleop ArActionGroupWander
```

Public Methods

- **ArActionGroup (ArRobot *robot)**
  
  Constructor.

- virtual ~ArActionGroup (void)
  
  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 ArActionGroup Class Reference

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. 306)

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::remAction (p. 327)

The documentation for this class was generated from the following files:
• ArActionGroup.h
• ArActionGroup.cpp
4.10 ArActionGroupInput Class Reference

Input to drive the robot.

#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupInput::

```
ArActionGroup

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.

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

- ArActionGroups.h
- ArActionGroups.cpp
4.11 ArActionGroupStop Class Reference

Stop the robot.
#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupStop:

```
ArActionGroupStop
   `-> ArActionGroup
```

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.

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

- ArActionGroups.h
- ArActionGroups.cpp
4.12 ArActionGroupTeleop Class Reference

Teleop the robot.
#include <ArActionGroups.h>

Inheritance diagram for ArActionGroupTeleop:

```
ArActionGroup

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.

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

- ArActionGroups.h
- ArActionGroups.cpp
4.13 **ArActionGroupWander Class Reference**

Has the robot wander.

```cpp
#include <ArActionGroups.h>
```

Inheritance diagram for ArActionGroupWander:

```
ArActionGroupWander
  |    |
  |    ArActionGroup
```

### 4.13.1 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.

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

- ArActionGroups.h
- ArActionGroups.cpp
4.14 **ArActionInput Class Reference**

Action for stopping the robot.

```cpp
#include <ArActionInput.h>
```

Inheritance diagram for ArActionInput:

```
ArAction
   |    
   v    
ArActionInput
```

**Public Methods**

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

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

- 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.
4.14.1 Detailed Description

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

4.14.2 Constructor & Destructor Documentation

4.14.2.1 ArActionInput::ArActionInput (const char ∗ name = ”Input”)

Constructor.

Parameters:
  name name of the action

4.14.3 Member Function Documentation

4.14.3.1 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. 40).

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

- ArActionInput.h
- ArActionInput.cpp
4.15 **ArActionJoydrive Class Reference**

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

```c
#include <ArActionJoydrive.h>
```

Inheritance diagram for ArActionJoydrive:

```
ArAction
  ^
  | ArActionJoydrive
```

**Public Methods**

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

- virtual ~**ArActionJoydrive** (void)
  
  **Destructor.**

- virtual **ArActionDesired** *fire** (**ArActionDesired** 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 **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.

- virtual ArActionDesired * getDesired (void)
  
  Gets what this action wants to do (for display purposes).

### 4.15.1 Detailed Description

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

This class creates its own ArJoyHandler (p. 190) 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.

### 4.15.2 Constructor & Destructor Documentation

#### 4.15.2.1 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. 190).

Parameters:

- name  the name of this action
- transVelMax the maximum velocity the joydrive action will go, it reaches this when the joystick is all the way forwards
- turnAmountMax the maximum amount the joydrive action will turn, it reaches 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)
4.15 ArActionJoydrive Class Reference

See also:
   ArJoyHandler::setUseOSCal (p. 193)

4.15.3 Member Function Documentation

4.15.3.1 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. 40).

4.15.3.2 bool ArActionJoydrive::getUseOSCal (void)

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

See also:
   ArJoyHandler::getUseOSCal (p. 193)

4.15.3.3 void ArActionJoydrive::setUseOSCal (bool useOSCal)

Sets whether to use OSCalibration the joystick or not.

See also:
   ArJoyHandler::setUseOSCal (p. 193)

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

- ArActionJoydrive.h
- ArActionJoydrive.cpp
4.16 ArActionKeydrive Class Reference

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

```cpp
#include <ArActionKeydrive.h>
```

Inheritance diagram for ArActionKeydrive:

```
ArAction

ArActionKeydrive
```

Public Methods

- **ArActionKeydrive** (const char *name="keydrive", double transVelMax=400, double turnAmountMax=24, double velIncrement=25, double turnIncrement=8)
  
  Constructor.

- virtual **~ArActionKeydrive** (void)
  
  Destructor.

- virtual **ArActionDesired * fire** (ArActionDesired 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)
4.16 ArActionKeydrive Class Reference

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.

4.16.1 Detailed Description

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

4.16.2 Member Function Documentation

4.16.2.1 `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. 40).

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

- ArActionKeydrive.h
- ArActionKeydrive.cpp
4.17 ArActionLimiterBackwards Class Reference

Action to limit the backwards motion of the robot.

#include <ArActionLimiterBackwards.h>

Inheritance diagram for ArActionLimiterBackwards:

```
          ArAction
            |
        ArActionLimiterBackwards
```

Public Methods

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

  Constructor.

- virtual ~ArActionLimiterBackwards (void)

  Destructor.

- virtual ArAction Desired * fire (ArAction Desired currentDesired)

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

- virtual ArAction Desired * getDesired (void)

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

4.17.1 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.
4.17.2 Constructor & Destructor Documentation

4.17.2.1 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)

4.17.3 Member Function Documentation

4.17.3.1 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. 40).

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

- ArActionLimiterBackwards.h
- ArActionLimiterBackwards.cpp
4.18  ArActionLimiterForwards Class Reference

Action to limit the forwards motion of the robot.

```c
#include "ArActionLimiterForwards.h"
```

Inheritance diagram for ArActionLimiterForwards:

```
    ArAction
     |     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** (void)

  Destructor.

- virtual **ArAction Desired * fire** (ArAction Desired currentDesired)

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

- virtual **ArAction Desired * getDesired** (void)

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

4.18.1  Detailed Description

Action to limit the forwards motion of the robot.

This action uses the sensors to find a maximum speed to travel at
4.18.2 Constructor & Destructor Documentation

4.18.2.1 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)

4.18.3 Member Function Documentation

4.18.3.1 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.40).

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

- ArActionLimiterForwards.h
- ArActionLimiterForwards.cpp
4.19  ArActionLimiterTableSensor Class Reference

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

```c
#include <ArActionLimiterTableSensor.h>
```

Inheritance diagram for ArActionLimiterTableSensor:

```
ArAction
  ↓
ArActionLimiterTableSensor
```

**Public Methods**

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

- **virtual ~ArActionLimiterTableSensor** (void)
  
  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 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.
4.19.2 Member Function Documentation

4.19.2.1 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. 40).

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

- ArActionLimiterTableSensor.h
- ArActionLimiterTableSensor.cpp
4.20 ArActionStallRecover Class Reference

Action to recover from a stall.

#include <ArActionStallRecover.h>

Inheritance diagram for ArActionStallRecover::

```
+--- ArAction
|   +--- ArActionStallRecover
```

Public Methods

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

- virtual **~ArActionStallRecover** (void)
  
  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 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.
4.20.2 Constructor & Destructor Documentation

4.20.2.1 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)

4.20.3 Member Function Documentation

4.20.3.1 AREXPORT ArActionDesired * ArActionStallRecover::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. 40).

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

- ArActionStallRecover.h
- ArActionStallRecover.cpp
4.21 ArActionStop Class Reference

Action for stopping the robot.

```cpp
#include "ArActionStop.h"
```

Inheritance diagram for ArActionStop::

```
ArActionStop
|    |
|---|--
|   | ArAction
```

Public Methods

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

- virtual **~ArActionStop** (void)
  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 for stopping the robot.

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

4.21.2 Constructor & Destructor Documentation

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

Constructor.

Parameters:

- **name** name of the action
4.21.3 Member Function Documentation

4.21.3.1 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. 40).

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

- ArActionStop.h
- ArActionStop.cpp
4.22 ArActionTurn Class Reference

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

```c
#include <ArActionTurn.h>
```

Inheritance diagram for ArActionTurn:

```
+---------------------+-------------------+
| ArAction            | ArActionTurn      |
'---------------------'-------------------'
```

Public Methods

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

  *Constructor.*

- virtual **~ArActionTurn** (void)

  *Destructor.*

- virtual **ArAction Desired * fire** (ArAction Desired currentDesired)

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

- virtual **ArAction Desired * getDesired** (void)

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

4.22.1 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.
4.22.2 Member Function Documentation

4.22.2.1 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. 40).

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

- ArActionTurn.h
- ArActionTurn.cpp
4.23 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** (void)
  
  Constructor.

- virtual ~ArACTS_1.2 (void)
  
  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 channel.

• 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 internal function to strip out the information from some bytes.

4.23.1 Detailed Description

Driver for ACTS.

4.23.2 Member Enumeration Documentation

4.23.2.1 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.
4.23.3 Member Function Documentation

4.23.3.1 \texttt{bool ArACTS\_1\_2::closePort (void)}

Closes the connection.
Closes the port to the ACTS server

\textbf{Returns:}
true if the connection was closed properly, false otherwise

4.23.3.2 \texttt{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.

\textbf{Parameters:}
channel the channel to get the blob from
blobNumber the number of the blob to get from the given channel
blob the blob instance to fill in with the data about the requested blob

\textbf{Returns:}
true if the blob instance could be filled in from the

4.23.3.3 \texttt{int ArACTS\_1\_2::getNumBlobs (int channel)}

Gets the number of blobs for the given channel.

\textbf{Returns:}
the number of blobs on the channel, or -1 if the channel is invalid

4.23.3.4 \texttt{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.

\textbf{Parameters:}
width the width of the images acts is grabbing (pixels)
height the height of the images acts is grabbing (pixels)
4.23.3.5 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

4.23.3.6 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 succesfully

4.23.3.7 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

4.23.3.8 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 successfully, false otherwise

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

- ArACTS.h
- ArACTS.cpp
4.24 ArACTSBlob Class Reference

A class for the acts blob.

```cpp
#include <ArACTS.h>
```

Public Methods

- **ArACTSBlob** (void)
  Constructor.

- virtual **~ArACTSBlob** (void)
  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.

- int **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 of 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 print** (void)
  
  *Prints the stats of the blob.*

### 4.24.1 Detailed Description

A class for the acts blob.

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

- ArACTS.h
# ArAMPTU Class Reference

Driver for the AMPUT.

```cpp
#include <ArAMPTU.h>
```

Inheritance diagram for ArAMPTU::

```
ArPTZ
ArAMPTU
```

## Public Methods

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

- virtual **~ArAMPTU** *(void)*
  
  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)  
  *Returns true if camera can zoom (or rather, if it is controlled by this).*

• virtual int **getMaxPosPan** (void)  
  *Gets the highest positive degree the camera can pan to.*

• virtual int **getMaxNegPan** (void)  
  *Gets the lowest negative degree the camera can pan to.*

• virtual int **getMaxPosTilt** (void)  
  *Gets the highest positive degree the camera can tilt to.*

• virtual int **getMaxNegTilt** (void)  
  *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)  
  *Gets the angle the camera is panned to.*

• virtual int **getTilt** (void)  
  *Gets the angle the camera is tilted to.*

### 4.25.1 Detailed Description

Driver for the AMPUT.
4.25.2 Constructor & Destructor Documentation

4.25.2.1 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

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

- ArAMPTU.h
- ArAMPTU.cpp
4.26 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, PANTILTDCCW = 0x2A,
  PANTILTUCW = 0x2B, PANTILTDCCW = 0x2C, ZOOM = 0x3F,
  PAUSE = 0x39, CONT = 0x3A, PURGE = 0x3B, STATUS = 0x3C,
  INIT = 0x3D, RESP = 0x3E, PANSLEW = 0x34, TILTSLEW = 0x38
}

4.26.1 Detailed Description

A class with the commands for the AMPTU.

4.26.2 Member Enumeration Documentation

4.26.2.1 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.
  PANTILTDCCW Relative tilt down, pan clockwise.
  PANTILTUCCW Relative tilt up, pan counter-clockwise.
  PANTILTDCCCW 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:

- ArAMPTU.h
4.27 ArAMPTUPacket Class Reference

A class for making commands to send to the AMPTU.

```cpp
#include <ArAMPTU.h>
```

Inheritance diagram for ArAMPTUPacket::

- ArBasePacket
- ArAMPTUPacket

Public Methods

- **ArAMPTUPacket (ArTypes::UByte2 bufferSize=30)**
  
  Constructor.

- **virtual ~ArAMPTUPacket (void)**
  
  Destructor.

- **unsigned char getUnitNumber (void)**
  
  Gets the unit number this packet is for.

- **bool setUnitNumber (unsigned char unitNumber)**
  
  Sets the unit number this packet is for.

- **virtual void byteToBuf (ArTypes::Byte val)**
  
  Puts ArTypes::Byte (p. 426) into packets buffer.

- **virtual void byte2ToBuf (ArTypes::Byte2 val)**
  
  Puts ArTypes::Byte2 (p. 426) into packets buffer.

- **virtual void finalize (void)**
  
  Finalizes the packet in preparation for sending, must be done.

4.27.1 Detailed Description

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

See also:
- getUnitNumber (p.98), setUnitNumber (p.98)

### 4.27.2 Member Function Documentation

#### 4.27.2.1 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

#### 4.27.2.2 bool ArAMPTUPacket::setUnitNumber (unsigned char unitNumber)

Sets 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.

**Parameters:**
- `unitNumber` the unit number for this packet, this needs to be 0-7

**Returns:**
- true if the number is acceptable, false otherwise

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

- ArAMPTU.h
- ArAMPTU.cpp
4.28 ArArg Class Reference

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

#include <ArArg.h>

Public Types

- enum Type { INVALID, INT, DOUBLE, STRING, BOOL, POSE }

Public Methods

- ArArg (void)
  Default empty constructor.

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

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

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

- ArArg (std::string name, bool *pointer, std::string description="")
  Constructor for making a boolean argument.

- ArArg (std::string name, ArPose *pointer, std::string description="")
  Constructor for making a position argument.

- ArArg (const ArArg &arg)
  Copy constructor.

- virtual ~ArArg (void)
  Destructor.

- Type getType (void)
  Gets the type of the argument.

- std::string getName (void)
  Gets the name of the argument.
• std::string **getDescription** (void)
  
  *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** (std::string 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. 231) arguments.*

• int **getInt** (void)
  
  *Gets the argument value, for int arguments.*

• double **getDouble** (void)
  
  *Gets the argument value, for double arguments.*

• std::string **getString** (void)
  
  *Gets the argument value, for string arguments.*

• bool **getBool** (void)
  
  *Gets the argument value, for bool arguments.*

• ArPose **getPose** (void)
  
  *Gets the argument value, for pose arguments.*

• void **print** (void)
  
  *Logs the type, name, and value of this argument.*

• void **clearPointers** (void)
  
  *Internal helper function.*
4.28.1 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 (p. 100). You should probably also add an

See also:
newType to the documentation for ArArg::getType (p. 101).

4.28.2 Member Enumeration Documentation

4.28.2.1 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. 231) argument.

4.28.3 Member Function Documentation

4.28.3.1 ArArg::Type ArArg::getType (void)

Gets the type of the argument.

See also:

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

- ArArg.h
- ArArg.cpp
4.29 ArArgumentBuilder Class Reference

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

#include <ariaUtil.h>

Public Methods

- **ArArgumentBuilder** (size_t argvLen=256)
  Constructor.

- virtual ~ArArgumentBuilder (void)
  Destructor.

- void **add** (char *str, ...)
  Adds the given string, with varargs, separates if there are spaces.

- void **print** (void)
  Prints out the arguments.

- size_t **getArgc** (void)
  Gets the argc.

- char **getArgv** (void)
  Gets the argv.

4.29.1 Detailed Description

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

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

- ariaUtil.h
- ariaUtil.cpp
4.30 ArArgumentParser Class Reference

Class for parsing arguments.
#include <ariaUtil.h>

Public Methods

- **ArArgumentParser** (int *argc, char **argv)
  Constructor, takes the argc argv.

- **∼ArArgumentParser** (void)
  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.

4.30.1 Detailed Description

Class for parsing arguments.
This class is made for parsing arguments from an argc/argv set... if you’re using a winmain you can first toss your string at the **ArArgumentBuilder** (p. 102) above class ArArgumentParser and then use this parser on it.

4.30.2 Constructor & Destructor Documentation

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

Constructor, takes the argc argv.

Parameters:
- **argc** a pointer to the argc used
- **argv** argv
4.30.3 Member Function Documentation

4.30.3.1 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

4.30.3.2 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

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

- ariaUtil.h
- ariaUtil.cpp
4.31 ArASyncTask Class Reference

Asynchronous task (runs in its own thread).

#include <ArASyncTask.h>

Inheritance diagram for ArASyncTask:

```
ArThread
    ArRecurrentTask
    ArRangeDeviceThreaded
    ArSick
    ArSignalHandler
    ArASyncTask
    ArSyncLoop
    ArTimer
```

Public Methods

- **ArASyncTask ()**
  
  Constructor.

- virtual ~**ArASyncTask ()**
  
  Destructor.

- virtual void * **runThread** (void *arg)=0
  
  The main run loop.

- 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 synchronously.

4.31.1 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. 106) function. They then need to create an instance of their task and call **create()** (p. 105). The standard way to stop a task is to call **stopRunning()** (p. 417) which sets **ArThread::myRunning** (p. 419) to false. In their run loop, they should pay attention to the **getRunning()** (p. 418).
or the \texttt{ArThread::myRunning} (p. 419) variable. If this value goes to false, the task should clean up after itself and exit its \texttt{runThread()} (p. 106) function.

4.31.2 Member Function Documentation

4.31.2.1 \texttt{void \* ArASyncTask::runInThisThread (void \* arg = 0)} [virtual]

Run the code of the task synchronously.

This will run the code of the ArASyncTask without creating a new thread to run it in. It performs the needed setup then calls \texttt{runThread()} (p. 106). 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.

\textbf{Parameters:}

- \textit{arg} the argument to pass to the \texttt{runThread()} (p. 106)

4.31.2.2 \texttt{virtual void\* ArASyncTask::runThread (void \* arg)} [pure virtual]

The main run loop.

Override this function and put your tasks run loop here. Check the value of \texttt{getRunning()} (p. 418) or myRunning periodically in your loop. If the value goes false, the loop should exit and \texttt{runThread()} (p. 106) should return.

Reimplemented in \texttt{ArRangeDeviceThreaded} (p. 265), \texttt{ArRecurrentTask} (p. 268), \texttt{ArSick} (p. 367), and \texttt{ArSignalHandler} (p. 388).

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

- ArASyncTask.h
- ArASyncTask.cpp
4.32   ArBasePacket Class Reference

Base packet class.

#include <ArBasePacket.h>

Inheritance diagram for ArBasePacket::

```
ArBasePacket ArAMPTUPacket ArDPPTUPacket ArRobotPacket ArSickPacket ArSonyPacket ArVCC4Packet
```

Public Methods

- **ArBasePacket** (ArTypes::UByte2 bufferSize=0, ArTypes::UByte2 headerLength=0, char *buf=NULL)
  Constructor.

- virtual ~ArBasePacket (void)
  Destructor.

- virtual void empty (void)
  resets the length for more data to be added.

- virtual void finalize (void)
  Finalizes the packet in preparation for sending, must be done.

- virtual void print (void)
  ArLogs the contents of the packet.

- virtual void printHex (void)
  ArLogs the contents of the packet in hex.

- virtual void byteToBuf (ArTypes::Byte val)
  Puts ArTypes::Byte (p. 426) into packets buffer.

- virtual void byte2ToBuf (ArTypes::Byte2 val)
  Puts ArTypes::Byte2 (p. 426) into packets buffer.

- virtual void byte4ToBuf (ArTypes::Byte4 val)
  Puts ArTypes::Byte4 (p. 426) into packets buffer.
virtual void uByteToBuf (ArTypes::UByte val)

Puts ArTypes::UByte (p. 426) into packets buffer.

virtual void uByte2ToBuf (ArTypes::UByte2 val)

Puts ArTypes::UByte2 (p. 426) into packet buffer.

virtual void uByte4ToBuf (ArTypes::UByte4 val)

Puts ArTypes::UByte (p. 426) 4 into packet buffer.

virtual void strToBuf (std::string 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. 426) from the buffer.

virtual ArTypes::Byte2 bufToByte2 (void)

Gets a ArTypes::Byte2 (p. 426) from the buffer.

virtual ArTypes::Byte4 bufToByte4 (void)

Gets a ArTypes::Byte4 (p. 426) from the buffer.

virtual ArTypes::UByte bufToUByte (void)

Gets a ArTypes::UByte (p. 426) from the buffer.

virtual ArTypes::UByte2 bufToUByte2 (void)

Gets a ArTypes::UByte2 (p. 426) from the buffer.

virtual ArTypes::UByte4 bufToUByte4 (void)

Gets a ArTypes::UByte4 (p. 426) from the buffer.

virtual std::string bufToStr ()

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** **getReadLength** (void)
  
  *Gets how the length of the packet that has been read.*

• virtual **ArTypes::UByte2** **getHeaderLength** (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 bool **setHeaderLength** (**ArTypes::UByte2** length)
  
  *Sets the length of the header.*

• virtual void **duplicatePacket** (**ArBasePacket** *packet)
  
  *Makes this packet a duplicate of another packet.*

4.32.1 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 finalize...
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 finalize method.

The base class and most classes of this kind will have an integer before the string, denoting the string's 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.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 ArBasePacket::ArBasePacket (ArTypes::UByte2
bufferSize = 0, ArTypes::UByte2 headerLength = 0, char
* buf = NULL)

Constructor.

Parameters:

bufferSize  size of the buffer

headerLength  length of the header

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

4.32.3 Member Function Documentation

4.32.3.1 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
4.32 ArBasePacket Class Reference

4.32.3.2 std::string ArBasePacket::bufToStr (void) [virtual]

Gets a string from the buffer.

first gets a length from the buffer, then gets a string from the buffer of the
length fetched

Returns:
  string from the buffer

4.32.3.3 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 character array to copy into buffer
  length how many botes to copy from data into packet

4.32.3.4 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

4.32.3.5 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

4.32.3.6 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 ArRobotPacket (p. 341), and ArSickPacket (p. 380).
4.32.3.7  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
  length how many bytes to copy from the str into packet

4.32.3.8  void ArBasePacket::strToBuf (std::string 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

4.32.3.9  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

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

- ArBasePacket.h
- ArBasePacket.cpp
4.33 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, SETSIMORIGNTH = 68, RESETSIMTOORIGIN = 69, SOUND = 90, PLAYLIST = 91, SOUNDTOG = 92 }

4.33.1 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.

4.33.2 Member Enumeration Documentation

4.33.2.1 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 incr) 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 ipackets 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 aux.

GETAUX int, requests 1-200 bytes from aux 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 robot's position 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 disable(0) sound.

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

- ArCommands.h
4.34 ArCondition Class Reference

Threading condition wrapper class.

```c
#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 ()
  Destructor.```
- ```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.```
- ```std::string getError (int messageNumber)
  Translate error into string.```

4.34.1 Detailed Description

Threading condition wrapper class.
4.34.2 Member Enumeration Documentation

4.34.2.1 enum ArCondition::typedef

Enumeration values:

- **STATUS_FAILED**  General failure.
- **STATUS_FAILED_DESTROY**  Another thread is waiting on this condition so it cannot be destroyed.
- **STATUS_FAILED_INIT**  Failed to initialize thread. Requested action is impossible.
- **STATUS_WAIT_TIMEDOUT**  The timedwait timed out before signaling.
- **STATUS_WAIT_INTR**  The wait was interrupted by a signal.
- **STATUS_MUTEX_FAILED_INIT**  The underlying mutex failed to init.
- **STATUS_MUTEX_FAILED**  The underlying mutex failed in some fashion.

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

- ArCondition.h
- ArCondition_LIN.cpp
- ArCondition_WIN.cpp
4.35 ArDeviceConnection Class Reference

Base class for device connections.

#include <ArDeviceConnection.h>

Inheritance diagram for ArDeviceConnection::

![Inheritance Diagram]

Public Types

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

Public Methods

- ArDeviceConnection (void)  
  constructor.

- virtual ~ArDeviceConnection (void)  
  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.

- std::string getStatusMessage (int messageNumber)  
  Gets the description string associated with the status.
4.35 ArDeviceConnection Class Reference

- virtual bool openSimple (void)=0
  
  \textit{Opens the connection again, using the values from setLocation or.}

- virtual bool close (void)
  
  \textit{Closes the connection.}

- virtual std::string getOpenMessage (int messageNumber)=0
  
  \textit{Gets the string of the message associated with opening the device.}

- virtual ArTime getTimeRead (int index)=0
  
  \textit{Gets the time data was read in.}

- virtual bool isTimeStamping (void)=0
  
  \textit{sees if timestamping is really going on or not.}

4.35.1 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.

4.35.2 Member Enumeration Documentation

4.35.2.1 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.
4.35.3 Member Function Documentation

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

Closes the connection.

**Returns:**
whether the close succeeded or not

Reimplemented in `ArSerialConnection` (p. 360), and `ArTcpConnection` (p. 413).

4.35.3.2 virtual std::string 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 `ArSerialConnection` (p. 360), and `ArTcpConnection` (p. 413).

4.35.3.3 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. 121)

Reimplemented in `ArSerialConnection` (p. 361), and `ArTcpConnection` (p. 414).
4.35.3.4 std::string ArDeviceConnection::getStatusMessage (int messageNumber)

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. 120)

4.35.3.5 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 ArSerialConnection (p. 361), and ArTcpConnection (p. 414).

4.35.3.6 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 ArSerialConnection (p. 361), and ArTcpConnection (p. 414).
4.35.3.7 virtual int ArDeviceConnection::read (const char *data, unsigned int size, unsigned int msWait = 0) [pure virtual]

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. 122), writePacket (p. 123)

Reimplemented in ArSerialConnection (p. 362), and ArTcpConnection (p. 415).

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

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. 122), writePacket (p. 123)

Reimplemented in ArSerialConnection (p. 363), and ArTcpConnection (p. 416).
4.35.3.9 virtual int ArDeviceConnection::writePacket
(ArBasePacket * packet) [inline, virtual]

Writes data to connection.

Parameters:
   packet pointer to a packet to write the data from

Returns:
   number of bytes read, or -1 for failure

See also:
   read (p. 122), write (p. 122)

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

- ArDeviceConnection.h
- ArDeviceConnection.cpp
4.36  ArDPPTU Class Reference

Driver for the DPPTU.

#include <ArDPPTU.h>

Inheritance diagram for ArDPPTU:

```
ArPTZ

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** (void)
  **Destructor.**

- bool **init** (void)
  *Initializes the camera.*

- bool **canZoom** (void)
  *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)
4.36 ArDPPTU Class Reference

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.

- **int getMaxPosPan (void)**
  Gets the highest positive degree the camera can pan to.

- **int getMaxNegPan (void)**
  Gets the lowest negative degree the camera can pan to.

- **int getMaxPosTilt (void)**
  Gets the highest positive degree the camera can tilt to.

- **int getMaxNegTilt (void)**
  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-motion power mode.

- bool `lowMotPower` (void)
  Sets low in-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.

- **int** `getPan` (void)
  
  The angle the camera was last told to pan to.

- **int** `getTilt` (void)
  
  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.

### 4.36.1 Detailed Description

Driver for the DPPTU.
4.36.2 Member Enumeration Documentation

4.36.2.1 anonymous enum

Enumeration values:
- **MIN_PAN** Maximum pan range of 3090 positions.
- **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.

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

- ArDPPTU.h
- ArDPPTU.cpp
4.37 ArDPPTUCarcomands 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 }

4.37.1 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. 244) 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.

4.37.2 Member Enumeration Documentation

4.37.2.1 anonymous enum

Enumeration values:
  DELIM Space - Carriage return delimiter.
  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.
UPPER  Upper speed limit.
VELOCITY  Velocity control mode.

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

- ArDPPTU.h
4.38 ArDPPTUPacket Class Reference

A class for making commands to send to the DPPTU.
#include <ArDPPTU.h>

Inheritance diagram for ArDPPTUPacket:

```
ArBasePacket
ArDPPTUPacket
```

Public Methods

- **ArDPPTUPacket (ArTypes::UByte2 bufferSize=30)**
  Constructor.

- virtual ~ArDPPTUPacket (void)
  Destructor.

- virtual void finalize (void)
  Finalizes the packet in preparation for sending, must be done.

4.38.1 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:

- ArDPPTU.h
- ArDPPTU.cpp
4.39 ArFunctor Class Reference

Base class for functors.

```
#include <ArFunctor.h>
```

Inheritance diagram for ArFunctor::

```
ArFunctor
  ArFunctor1 ArFunctor2 ArFunctorC ArGlobalFunctor ArRetFunctor
  ArFunctor1C ArGlobalFunctor1 ArFunctor2C ArGlobalFunctor2 ArGlobalRetFunctor
  ArRetFunctor1 ArRetFunctor2 ArRetFunctorC
```

Public Methods

- virtual ~ArFunctor (void)
  Destructor.

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

- virtual void operator() ()=0
  Invokes the functor.

4.39.1 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.153), ArGlobalRetFunctor (p.163)) 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.181) makes use of functors by using them as callback functions. Since Aria (p.181) 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 ArFunctor, 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.135)<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.135), ArFunctor2 (p.141), ArRetFunctor (p.271), ArRetFunctor1 (p.273), ArRetFunctor2 (p.279)

These 6 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.150), ArFunctor1C (p.137), ArFunctor2C (p.144), ArRetFunctorC (p.287), ArRetFunctor1C (p.275), ArRetFunctor2C (p.281)

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.

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:

- ArFunctor.h
4.40  ArFunctor1 Class Template Reference

Base class for functors with 1 parameter.
#include <ArFunctor.h>

Inheritance diagram for ArFunctor1::

```
ArFunctor1
 |   |
|---|--
ArGlobalFunctor1

Public Methods

- virtual ~ArFunctor1 (void)
  Destructor.

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

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

- virtual void operator() ()=0
  Invokes the functor.

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

4.40.1  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 \texttt{ArFunctor} (p. 133).

### 4.40.2 Member Function Documentation

#### 4.40.2.1 template <class P1> virtual void ArFunctor1< P1>

> ::invoke (P1 \texttt{p1})  [pure virtual]

Invokes the functor.

**Parameters:**

- \texttt{p1} first parameter

Reimplemented in \texttt{ArGlobalFunctor1} (p. 156), and \texttt{ArFunctor1C} (p. 139).

#### 4.40.2.2 template <class P1> virtual void ArFunctor1< P1>

> ::operator() (P1 \texttt{p1})  [pure virtual]

Invokes the functor.

**Parameters:**

- \texttt{p1} first parameter

Reimplemented in \texttt{ArGlobalFunctor1} (p. 157), and \texttt{ArFunctor1C} (p. 139).

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

- \texttt{ArFunctor.h}
4.41 ArFunctor1C Class Template Reference

Functor for a member function with 1 parameter.

```cpp
#include <ArFunctor.h>
```

Inheritance diagram for ArFunctor1C:

![Inheritance Diagram](image_url)

**Public Methods**

- **ArFunctor1C** (void)
  
  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 (void)
  
  Destructor.

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

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

- virtual void **operator()** ()

---

Invokes the functor.

- virtual void \texttt{operator()} (P1 p1)
  
  Invokes the functor.

- virtual void \texttt{setThis} (T *obj)
  
  Set the 'this' pointer.

- virtual void \texttt{setThis} (T &obj)
  
  Set the 'this' pointer.

- virtual void \texttt{setP1} (P1 p1)
  
  Set the default parameter.

### 4.41.1 Detailed Description

\texttt{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 \texttt{ArFunctor} (p. 133).

### 4.41.2 Constructor & Destructor Documentation

#### 4.41.2.1 \texttt{template<class T, class P1> ArFunctor1C<T, P1> ::ArFunctor1C (T & obj, void(T::* func)(P1)) [inline]}

Constructor - supply function pointer.

\textbf{Parameters:}

\texttt{func} member function pointer

#### 4.41.2.2 \texttt{template<class T, class P1> ArFunctor1C<T, P1> ::ArFunctor1C (T & obj, void(T::* func)(P1), P1 p1) [inline]}

Constructor - supply function pointer, default parameters.
4.41.2.3  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

4.41.2.4  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

4.41.3  Member Function Documentation

4.41.3.1  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.136).

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

Invokes the functor.

Parameters:
  *p1* first parameter

Reimplemented from *ArFunctor1* (p.136).
4.41.3.3  template<class T, class P1> virtual void ArFunctor1C<T, P1>::setP1 (P1 p1)  [inline, virtual]

Set the default parameter.

Parameters:

  \textit{p1}  default first parameter

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

Set the 'this' pointer.

Parameters:

  \textit{obj}  the 'this' pointer

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

Set the 'this' pointer.

Parameters:

  \textit{obj}  the 'this' pointer

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

- ArFunctor.h
Base class for functors with 2 parameters.
#include <ArFunctor.h>

Inheritance diagram for ArFunctor2:

Public Methods

- virtual ~ArFunctor2 (void)  
  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 operator() ()=0  
  Invokes the functor.

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

- virtual void operator() (P1 p1, P2 p2)=0  
  Invokes the functor.
4.42.1 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.133).

4.42.2 Member Function Documentation

4.42.2.1 template<class P1, class P2> virtual void ArFunctor2<P1, P2>::invoke (P1 p1, P2 p2) [pure virtual]

Invokes the functor.

Parameters:
  p1 first parameter
  p2 second parameter

Reimplemented in ArGlobalFunctor2 (p.160), and ArFunctor2C (p.147).

4.42.2.2 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.160), and ArFunctor2C (p.147).

4.42.2.3 template<class P1, class P2> virtual void ArFunctor2<P1, P2>::operator() (P1 p1, P2 p2) [pure virtual]

Invokes the functor.

Parameters:
  p1 first parameter
  p2 second parameter

Reimplemented in ArGlobalFunctor2 (p.161), and ArFunctor2C (p.148).
template<class P1, class P2> virtual void ArFunctor2<P1, P2>::operator() (P1 p1) [pure virtual]

Invokes the functor.

Parameters:

\texttt{p1} first parameter

Reimplemented in \texttt{ArGlobalFunctor2} (p. 161), and \texttt{ArFunctor2C} (p. 148).
4.43 ArFunctor2C Class Template Reference

Functor for a member function with 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArFunctor2C:

```
ArFunctor

ArFunctor2< P1, P2 >

ArFunctor2C
```

Public Methods

- **ArFunctor2C (void)**
  
  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 (void)
  
  Destructor.

- virtual void invoke (void)
4.43 ArFunctor2C Class Template Reference

Invokes the functor.

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

- virtual void `invoke` (P1 p1, P2 p2)
  Invokes the functor.

- virtual void `operator()` ()
  Invokes the functor.

- virtual void `operator()` (P1 p1)
  Invokes the functor.

- virtual void `operator()` (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.

4.43.1 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. 133).
4.43.2 Constructor & Destructor Documentation

4.43.2.1 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:

\texttt{func} member function pointer

4.43.2.2 template<class T, class P1, class P2> ArFunctor2C< T, 
P1, P2 >::ArFunctor2C (T & obj, void(T::* func)(P1, P2), 
P1 \texttt{p1}) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

\texttt{func} member function pointer
\texttt{p1} default first parameter

4.43.2.3 template<class T, class P1, class P2> ArFunctor2C< T, 
P1, P2 >::ArFunctor2C (T & obj, void(T::* func)(P1, P2), 
P1 \texttt{p1}, P2 \texttt{p2}) [inline]

Constructor - supply function pointer, default parameters.

Parameters:

\texttt{func} member function pointer
\texttt{p1} default first parameter
\texttt{p2} default second parameter

4.43.2.4 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:

\texttt{func} member function pointer
4.43.2.5 \template<\text{T}, \text{P1}, \text{P2}> \text{ArFunctor2C}< \text{T}, \text{P1}, \text{P2} >::\text{ArFunctor2C} (\text{T} \ast \text{obj}, \text{void(}\text{T}::\text{func})(\text{P1}, \text{P2}), \text{P1 p1}) \quad [\text{inline}]

Constructor - supply function pointer, default parameters.

Parameters:
- \text{func} member function pointer
- \text{p1} default first parameter

4.43.2.6 \template<\text{T}, \text{P1}, \text{P2}> \text{ArFunctor2C}< \text{T}, \text{P1}, \text{P2} >::\text{ArFunctor2C} (\text{T} \ast \text{obj}, \text{void(}\text{T}::\text{func})(\text{P1}, \text{P2}), \text{P1 p1}, \text{P2 p2}) \quad [\text{inline}]

Constructor - supply function pointer, default parameters.

Parameters:
- \text{func} member function pointer
- \text{p1} default first parameter
- \text{p2} default second parameter

4.43.3 Member Function Documentation

4.43.3.1 \template<\text{T}, \text{P1}, \text{P2}> \text{virtual void ArFunctor2C}< \text{T}, \text{P1}, \text{P2} >::\text{invoke} (\text{P1 p1}, \text{P2 p2}) \quad [\text{inline, virtual}]

Invokes the functor.

Parameters:
- \text{p1} first parameter
- \text{p2} second parameter

Reimplemented from \text{ArFunctor2} (p. 142).

4.43.3.2 \template<\text{T}, \text{P1}, \text{P2}> \text{virtual void ArFunctor2C}< \text{T}, \text{P1}, \text{P2} >::\text{invoke} (\text{P1 p1}) \quad [\text{inline, virtual}]

Invokes the functor.
Parameters:
   \texttt{p1} first parameter

Reimplemented from \texttt{ArFunctor2} (p. 142).

4.43.3.3 \texttt{template<class T, class P1, class P2> virtual void}
       \texttt{ArFunctor2C< T, P1, P2 >::operator() (P1 \ p1, P2 \ p2)}
       \texttt{[inline, virtual]}

Invokes the functor.

Parameters:
   \texttt{p1} first parameter
   \texttt{p2} second parameter

Reimplemented from \texttt{ArFunctor2} (p. 142).

4.43.3.4 \texttt{template<class T, class P1, class P2> virtual void}
       \texttt{ArFunctor2C< T, P1, P2 >::operator() (P1 \ p1)}
       \texttt{[inline, virtual]}

Invokes the functor.

Parameters:
   \texttt{p1} first parameter

Reimplemented from \texttt{ArFunctor2} (p. 143).

4.43.3.5 \texttt{template<class T, class P1, class P2> virtual void}
       \texttt{ArFunctor2C< T, P1, P2 >::setP1 (P1 \ p1)}
       \texttt{[inline, virtual]}

Set the default parameter.

Parameters:
   \texttt{p1} default first parameter

4.43.3.6 \texttt{template<class T, class P1, class P2> virtual void}
       \texttt{ArFunctor2C< T, P1, P2 >::setP2 (P2 \ p2)}
       \texttt{[inline, virtual]}

Set the default 2nd parameter.
Parameters:
   \( p2 \) default second parameter

4.43.3.7 \textbf{template <class T, class P1, class P2> virtual void} \newline \hspace{1cm} \texttt{ArFunctor2C<T, P1, P2>::setThis (T & obj)} \ [\text{inline, virtual}] \newline
Set the 'this' pointer.

\textbf{Parameters:} \newline
   \textit{obj} the 'this' pointer

4.43.3.8 \textbf{template <class T, class P1, class P2> virtual void} \newline \hspace{1cm} \texttt{ArFunctor2C<T, P1, P2>::setThis (T * obj)} \ [\text{inline, virtual}] \newline
Set the 'this' pointer.

\textbf{Parameters:} \newline
   \textit{obj} the 'this' pointer

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

- ArFunctor.h
4.44  ArFunctorC Class Template Reference

Functor for a member function.

#include <ArFunctor.h>

Inheritance diagram for ArFunctorC::

Public Methods

- **ArFunctorC** (void)
  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** (void)
  Destructor.

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

- virtual void **operator()** ()
  Invokes the functor.

- virtual void **setThis** (T *obj)
  Set the 'this' pointer.

- virtual void **setThis** (T &obj)
  Set the 'this' pointer.
4.44.1 Detailed Description

```cpp
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. 133).

4.44.2 Constructor & Destructor Documentation

4.44.2.1 template<class T> ArFunctorC<T> ::ArFunctorC (T & obj, void(T::* func)(void)) [inline]

Constructor - supply function pointer.

**Parameters:**

- `func` member function pointer

4.44.2.2 template<class T> ArFunctorC<T> ::ArFunctorC (T * obj, void(T::* func)(void)) [inline]

Constructor - supply function pointer.

**Parameters:**

- `func` member function pointer

4.44.3 Member Function Documentation

4.44.3.1 template<class T> virtual void ArFunctorC<T> ::setThis (T & obj) [inline, virtual]

Set the 'this' pointer.

**Parameters:**

- `obj` the 'this' pointer
4.44.3.2  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
4.45  **ArGlobalFunctor Class Reference**

Functor for a global function with no parameters.

```cpp
#include <ArFunctor.h>
```

Inheritance diagram for `ArGlobalFunctor`:

```
    ArFunctor
     |           |
     v           v
ArGlobalFunctor
```

**Public Methods**

- **`ArGlobalFunctor (void)`**  
  *Constructor.*

- **`ArGlobalFunctor (void(*func)(void))`**  
  *Constructor - supply function pointer.*

- **virtual `~ArGlobalFunctor (void)`**  
  *Destructor.*

- **virtual void `invoke (void)`**  
  *Invokes the functor.*

- **virtual void `operator() ()`**  
  *Invokes the functor.*

### 4.45.1 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 convenience. Code that has a reference to this class and treat it as an `ArFunctor` (p. 133) can use it like any other functor.

For an overall description of functors, see `ArFunctor` (p. 133).
4.45.2 Constructor & Destructor Documentation

4.45.2.1 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
4.46 ArGlobalFunctor1 Class Template Reference

Functor for a global function with 1 parameter.

```cpp
#include <ArFunctor.h>
```

Inheritance diagram for ArGlobalFunctor1::

```
```

Public Methods

- **ArGlobalFunctor1** (void)
  Constructor.
- **ArGlobalFunctor1** (void(*func)(P1))
  Constructor - supply function pointer.
- **ArGlobalFunctor1** (void(*func)(P1), P1 p1)
  Constructor - supply function pointer, default parameters.
- virtual **ArGlobalFunctor1** (void)
  Destructor.
- virtual void **invoke** (void)
  Invokes the functor.
- virtual void **invoke** (P1 p1)
  Invokes the functor.
- virtual void **operator()** ()
  Invokes the functor.
- virtual void **operator()** (P1 p1)
  Invokes the functor.
4.46.1 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. 133) can use it like
any other functor.

For an overall description of functors, see ArFunctor (p. 133).

4.46.2 Constructor & Destructor Documentation

4.46.2.1 template<class P1> ArGlobalFunctor1< P1 >
 >::ArGlobalFunctor1 (void(* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.46.2.2 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

4.46.3 Member Function Documentation

4.46.3.1 template<class P1> virtual void ArGlobalFunctor1< P1 >
 >::invoke (P1 p1) [inline, virtual]

Invokes the functor.
Parameters:
  \( p1 \) first parameter

Reimplemented from \texttt{ArFunctor1} \hbox{\cite{p.136}}.

4.46.3.2 \texttt{template<class P1> virtual void ArGlobalFunctor1\< P1 \> ::operator() (P1 \ p1) \ [inline, virtual]}

Invokes the functor.

Parameters:
  \( p1 \) first parameter

Reimplemented from \texttt{ArFunctor1} \hbox{\cite{p.136}}.

4.46.3.3 \texttt{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:

- \texttt{ArFunctor.h}
4.47 ArGlobalFunctor2 Class Template Reference

Functor for a global function with 2 parameters.
#include <ArFunctor.h>

Inheritance diagram for ArGlobalFunctor2::

```
ArFunctor
  ↓
ArFunctor2< P1, P2 >
  ↓
ArGlobalFunctor2
```

Public Methods

- **ArGlobalFunctor2** (void)
  Constructor.

- **ArGlobalFunctor2** (void(*)(P1, P2))
  Constructor - supply function pointer.

- **ArGlobalFunctor2** (void(*)(P1, P2), P1 p1)
  Constructor - supply function pointer, default parameters.

- **ArGlobalFunctor2** (void(*)(P1, P2), P1 p1, P2 p2)
  Constructor - supply function pointer, default parameters.

- virtual ~**ArGlobalFunctor2** (void)
  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 operator() ()
  *Invokes the functor.*

• virtual void operator() (P1 p1)
  *Invokes the functor.*

• virtual void operator() (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.*

### 4.47.1 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 convenience. Code that has a reference to this class and treat it as an `ArFunctor` (p. 133) can use it like any other functor.

For an overall description of functors, see `ArFunctor` (p. 133).

### 4.47.2 Constructor & Destructor Documentation

**4.47.2.1 template<class P1, class P2> ArGlobalFunctor2<P1, P2> ::ArGlobalFunctor2 (void(* func)(P1, P2))** [inline]

Constructor - supply function pointer.

**Parameters:**

*func* global function pointer
4.47.2.2  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

4.47.2.3  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 pointer
- p1 default first parameter
- p2 default second parameter

4.47.3  Member Function Documentation

4.47.3.1  template<class P1, class P2> virtual void ArGlobalFunctor2< P1, P2 >::invoke (P1 p1, P2 p2) [inline, virtual]

Invokes the functor.

Parameters:
- p1 first parameter
- p2 second parameter

Reimplemented from ArFunctor2 (p. 142).

4.47.3.2  template<class P1, class P2> virtual void ArGlobalFunctor2< P1, P2 >::invoke (P1 p1) [inline, virtual]

Invokes the functor.
Parameters:

- \textit{p1} first parameter

Reimplemented from \texttt{ArFunctor2} (p.142).

\subsection*{4.47.3.3 template<class P1, class P2> virtual void \texttt{ArGlobalFunctor2<P1, P2>::operator() (P1 \ p1, P2 \ p2)}} [inline, virtual]

Invokes the functor.

Parameters:

- \textit{p1} first parameter
- \textit{p2} second parameter

Reimplemented from \texttt{ArFunctor2} (p.142).

\subsection*{4.47.3.4 template<class P1, class P2> virtual void \texttt{ArGlobalFunctor2<P1, P2>::operator() (P1 \ p1)}} [inline, virtual]

Invokes the functor.

Parameters:

- \textit{p1} first parameter

Reimplemented from \texttt{ArFunctor2} (p.143).

\subsection*{4.47.3.5 template<class P1, class P2> virtual void \texttt{ArGlobalFunctor2<P1, P2>::setP1 (P1 \ p1)}} [inline, virtual]

Set the default parameter.

Parameters:

- \textit{p1} default first parameter

\subsection*{4.47.3.6 template<class P1, class P2> virtual void \texttt{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:

- ArFunctor.h
4.48  

ArGlobalRetFunctor Class Template Reference

Functor for a global function with return value.

```
#include <ArFunctor.h>
```

Inheritance diagram for ArGlobalRetFunctor::

```
ArFunctor
   ↓
ArRetFunctor< Ret >
   ↓
ArGlobalRetFunctor
```

Public Methods

- **ArGlobalRetFunctor** (void)
  
  Constructor.

- **ArGlobalRetFunctor** (Ret(*func)(void))
  
  Constructor - supply function pointer.

- virtual **~ArGlobalRetFunctor** (void)
  
  Destructor.

- virtual Ret **invokeR** (void)
  
  Invokes the functor with return value.

4.48.1  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. 133) can use it like any other functor.

For an overall description of functors, see ArFunctor (p. 133).
4.48.2 Constructor & Destructor Documentation

4.48.2.1 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
4.49 ArGlobalRetFunctor1 Class Template Reference

Functor for a global function with 1 parameter and return value.

```
#include <ArFunctor.h>
```

Inheritance diagram for ArGlobalRetFunctor1::

```
ArFunctor
    
ArRetFunctor< Ret >
    
ArRetFunctor1< Ret, P1 >
    
ArGlobalRetFunctor1
```

Public Methods

- **ArGlobalRetFunctor1 (void)**
  *Constructor.*

- **ArGlobalRetFunctor1 (Ret(*func)(P1))**
  *Constructor - supply function pointer.*

- **ArGlobalRetFunctor1 (Ret(*func)(P1), P1 p1)**
  *Constructor - supply function pointer, default parameters.*

- virtual **~ArGlobalRetFunctor1 (void)**
  *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.*
4.49.1 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 convenience. Code that has a reference to this class and treat it as an ArFunctor (p.133) can use it like any other functor.

For an overall description of functors, see ArFunctor (p.133).

4.49.2 Constructor & Destructor Documentation

4.49.2.1 template<class Ret, class P1> ArGlobalRetFunctor1< Ret, P1 >::ArGlobalRetFunctor1 (Ret(* func)(P1)) [inline]

Constructor - supply function pointer.

Parameters:

func global function pointer

4.49.2.2 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

4.49.3 Member Function Documentation

4.49.3.1 template<class Ret, class P1> virtual Ret ArGlobalRetFunctor1< Ret, P1 >::invokeR (P1 p1) [inline, virtual]

Invokes the functor with return value.
Parameters:
   \textit{p1} first parameter

Reimplemented from \texttt{ARRetFunctor1} (p. 274).

\subsection{4.49.3.2 template\texttt{<class Ret, class P1> virtual void ArGlobalRetFunctor1\texttt{< Ret, P1 >::setP1 (P1 p1) [inline, virtual]}}

Set the default parameter.

Parameters:
   \textit{p1} default first parameter

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

- \texttt{ArFunctor.h}
4.50 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** (void)
  
  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** (void)
  
  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.*

### 4.50.1 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 convenience. Code that has a reference to this class and treat it as an ArFunctor (p. 133) can use it like any other functor.

For an overall description of functors, see ArFunctor (p. 133).

### 4.50.2 Constructor & Destructor Documentation

#### 4.50.2.1 template<class Ret, class P1, class P2> ArGlobalRetFunctor2<
  Ret, P1, P2>::ArGlobalRetFunctor2 (Ret(* func)(P1, P2)) [inline]

Constructor - supply function pointer.

**Parameters:**

- *func* global function pointer

#### 4.50.2.2 template<class Ret, class P1, class P2> ArGlobalRetFunctor2<
  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
4.50.2.3  template<class Ret, class P1, class P2> ArGlobalRetFunctor2< Ret, P1, P2 >::ArGlobalRetFunctor2 (Ret(* func)(P1, P2), P1 p1, P2 p2) [inline]

Constructor - supply function pointer, default parameters.

Parameters:
  *func*  global function pointer
  *p2*  default second parameter

4.50.3  Member Function Documentation

4.50.3.1  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 parameter
  *p2*  second parameter

Reimplemented from *ArRetFunctor2* (p. 280).

4.50.3.2  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. 280).

4.50.3.3  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
4.50.3.4 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:

- ArFunctor.h
4.51 ArGripper Class Reference

A class of convenience functions for using the gripper.

```cpp
#include <ArGripper.h>
```

Public Types

- `enum Type { QUERYTYPE, GENIO, USERIO, GRIPPAC, NOGRIPPER }`
  
  *These are the types for the gripper.*

Public Methods

- `ArGripper (ArRobot *robot, int gripperType=QUERYTYPE)`
  
  *Constructor.*

- `virtual ~ArGripper (void)`
  
  *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.*

- `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)
  
  Returns true if the gripper paddles are moving.

• bool isLiftMoving (void)
  
  Returns true if the lift is moving.

• int getGripState (void)
  
  Returns the state of the gripper paddles.

• int getPaddleState (void)
  
  Returns the state of each gripper paddle.

• int getBreakBeamState (void)
  
  Returns the state of the gripper’s breakbeams.

• bool isLiftMaxed (void)
  
  Returns the state of the lift.

• int getType (void)
  
  Gets the type of the gripper.

• void setType (int type)
  
  Sets the type of the gripper.

• long getMSecSinceLastPacket (void)
  
  Gets the number of mSec since the last gripper packet.

• int getGraspTime (void)
  
  Gets the grasp time.
• void printState (void)
  logs the gripper state.

• bool packetHandler (ArRobotPacket *packet)
  Parses the gripper packet.

• void connectHandler (void)
  The handler for when the robot connects.

4.51.1 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.

4.51.2 Member Enumeration Documentation

4.51.2.1 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.

4.51.3 Constructor & Destructor Documentation

4.51.3.1 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
4.51 ArGripper Class Reference

4.51.4 Member Function Documentation

4.51.4.1 int ArGripper::getBreakBeamState (void)

Returns the state of the gripper’s breakbeams.

Returns:
0 if no breakbeams broken, 1 if inner breakbeam broken, 2 if outer breakbeam broken, 3 if both breakbeams broken

4.51.4.2 int ArGripper::getGraspTime (void)

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

4.51.4.3 int ArGripper::getGripState (void)

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

4.51.4.4 long ArGripper::getMSecSinceLastPacket (void)

Gets the number of mSec since the last gripper packet.

Returns:
the number of milliseconds since the last packet

4.51.4.5 int ArGripper::getPaddleState (void)

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

4.51.4.6  int ArGripper::getType (void)

Gets the type of the gripper.

Returns:
the gripper type

See also:
Type (p. 174)

4.51.4.7  bool ArGripper::gripClose (void)

Closes the gripper paddles.

Returns:
whether the command was sent to the robot or not

4.51.4.8  bool ArGripper::gripOpen (void)

Opens the gripper paddles.

Returns:
whether the command was sent to the robot or not

4.51.4.9  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

4.51.4.10 bool ArGripper::gripperHalt (void)

Halts the lift and the gripper paddles.

Returns:
whether the command was sent to the robot or not
4.51.4.11 bool ArGripper::gripperStore (void)

Puts the gripper in a storage position.

Returns:
whether the command was sent to the robot or not

4.51.4.12 bool ArGripper::gripPressure (int mSecIntervals)

Sets the amount of pressure the gripper applies.

Returns:
whether the command was sent to the robot or not

4.51.4.13 bool ArGripper::gripStop (void)

Stops the gripper paddles.

Returns:
whether the command was sent to the robot or not

4.51.4.14 bool ArGripper::isGripMoving (void)

Returns true if the gripper paddles are moving.

Returns:
true if the gripper paddles are moving

4.51.4.15 bool ArGripper::isLiftMaxed (void)

Returns the state of the lift.

Returns:
false if lift is between up and down, true is either all the way up or down

4.51.4.16 bool ArGripper::isLiftMoving (void)

Returns true if the lift is moving.

Returns:
ture if the lift is moving
4.51.4.17  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

4.51.4.18  bool ArGripper::liftDown (void)

Lowers the lift to the bottom.

**Returns:**

whether the command was sent to the robot or not

4.51.4.19  bool ArGripper::liftStop (void)

Stops the lift.

**Returns:**

whether the command was sent to the robot or not

4.51.4.20  bool ArGripper::liftUp (void)

Raises the lift to the top.

**Returns:**

whether the command was sent to the robot or not

4.51.4.21  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
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_HAL = 15, GRIPPRESSURE = 16, LIFT_CARRY = 17 }

4.52.1 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.

4.52.2 Member Enumeration Documentation

4.52.2.1 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 lift simultaneously, this is for getting ready to grasp an object, not for object drops.
- GRIPPER_HAL stops the gripper paddles and lift from moving.
GRIPPRESSURE sets the time delay in 20 msec increments after the gripper paddles first grasp an object before they stop moving, regulates grasp pressure.

LIFTCARRY raises or lowers the lieft, the argument is the number of 20 msec increments to raise or lower the lift, positive arguments for raise, negative for lower.

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

- ArGripper.h
4.53 Aria Class Reference

This class performs global initialization and deinitialization.

```cpp
#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.

- 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 ()
  Gets the directory that ARIA resides in.

• void setKeyHandler (ArKeyHandler *keyHandler)
  Sets the key handler, so that other classes can find it.

• ArKeyHandler * getKeyHandler ()
  Gets the key handler if one has been set.

• void signalHandlerCB (int sig)
  Internal, the callback for the signal handling.

4.53.1 Detailed Description

This class performs global initialization and deinitialization.

4.53.2 Member Enumeration Documentation

4.53.2.1 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.

4.53.3 Member Function Documentation

4.53.3.1 void Aria::addInitCallBack (ArFunctor * cb,
             ArListPos::Pos position) [static]

Adds a callback to call when Aria is initied.
This will add a callback to the list of callbacks to call when Aria has been initialized. It can be called before anything else.
4.53.3.2 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 un-
initialized. It can be called before anything else. This facilitates code that
in operating system signal handlers simply calls Aria::uninit() (p. 185) and
packages that are based on Aria are uninited as well. It simplifies the entire
uninit process.

4.53.3.3 void Aria::exit () [static]

Force an exit of all Aria processes/threads.
This calls cancel() on all AtThread’s and ArASyncTask (p. 105)’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.

4.53.3.4 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

4.53.3.5 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. 184)
4.53.3.6 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 initializes 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. 290) 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:
methode the method in which to handle signals. Defaulted to SIGHANDLE_SINGLE.

initSockets specify whether or not to initialize the socket layer. This is only meaningful for Windows. Defaulted to true.

See also:
ArSignalHandler (p. 384) , ArSocket (p. 390)

4.53.3.7 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. 184) is done. So for setDirectory to be effective, it must be done after the Aria::init (p. 184).

Parameters:
directory the directory Aria is located in

See also:
getDirectory (p. 183)

4.53.3.8 void Aria::shutdown () [static]

Shutdown all Aria processes/threads.
This calls stop() on all ArThread (p. 417)’s and ArASyncTask (p. 105)’s. It will block until all ArThread (p. 417)’s and ArASyncTask (p. 105)’s exit. It is expected that all the tasks will obey the ArThread::myRunning (p. 419) variable and exit when it is false.

4.53.3.9 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:

- ariaInternal.h
- Aria.cpp
4.54 ArInterpolation Class Reference

#include <ArInterpolation.h>

Public Methods

- **ArInterpolation** (size_t numberOfReadings=100)
  Constructor.

- virtual ~ArInterpolation (void)
  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)
  Gets the number of readings this instance holds back in time.

- void reset (void)
  Empties the interpolated positions.

4.54.1 Detailed Description

This class takes care of storing in readings of position vs time, and then interpo-
lating 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 main-
tain 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.
4.54.2 Member Function Documentation

4.54.2.1 int ArInterpolation::getPose (ArTime time\_Stamp, ArPose * position)

Finds a position.

**Parameters:**

- `time\_Stamp` 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
4.55  ArIrrfDevice Class Reference

A class for connecting to a PB-9 and managing the resulting data.

#include <ArIrrfDevice.h>

Inheritance diagram for ArIrrfDevice::

```
ArRangeDevice
  ArIrrfDevice
```

Public Methods

- **ArIrrfDevice** (size_t currentBufferSize=91, size_t cumulativeBufferSize=273, const char *name="irrf")
  Constructor.

- **~ArIrrfDevice** (void)
  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.

4.55.1  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 packet handler 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.
4.55.2 Member Function Documentation

4.55.2.1 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
4.56 ArJoyHandler Class Reference

Interfaces to a joystick.

#include <ArJoyHandler.h>

Public Methods

- **ArJoyHandler** (bool useOSCal=true)
  
  Constructor.

- **~ArJoyHandler** (void)
  
  Destructor.

- **bool init** (void)
  
  Initializes the joystick, returns true if successful.

- **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.

- **bool haveJoystick** (void)
  
  Returns if the joystick was successfully initialized or not.

- **void setSpeeds** (int x, int y)
  
  Sets the max that X or Y will return.

- **void getAdjusted** (int *x, int *y)
  
  Gets the adjusted reading, as integers, based on the setSpeed.

- **void getDoubles** (double *x, double *y)
  
  Gets the adjusted reading, as floats, between -1.0 and 1.0.

- **bool getButton** (int button)
  
  Checks whether the given button is pressed or not.

- **void startCal** (void)
  
  Starts the calibration process.

- **void endCal** (void)
  
  Ends the calibration process.
• void **getUnfiltered** (int *x, int *y)
  
  *Gets the unfiltered 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)
  
  *Gets the speeds that X and Y are set to.*

### 4.56.1 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 'getAdjusted', 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.

### 4.56.2 Constructor & Destructor Documentation

#### 4.56.2.1 ArJoyHandler::ArJoyHandler (bool useOSCal = true)

Constructor.
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.

### 4.56.3 Member Function Documentation

#### 4.56.3.1 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. 194)

#### 4.56.3.2 void ArJoyHandler::getAdjusted (int * x, int * y)

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 to the class, and returns the values computed from this.

**Parameters:**

- `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

#### 4.56.3.3 bool ArJoyHandler::getButton (int button)

Checks whether the given button is pressed or not.

**Parameters:**

- `button` button to test for pressed, buttons are 1 through 4

**Returns:**

- true if the button is pressed, false otherwise
4.56.3.4 void ArJoyHandler::getDoubles (double * x, double * y)

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:
- 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

4.56.3.5 void ArJoyHandler::getUnfiltered (int * x, int * y)

Gets the unfiltered 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 positive 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:
- x pointer to an integer in which to store x value
- y pointer to an integer in which to store y value

4.56.3.6 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

4.56.3.7 void ArJoyHandler::setUseOSCal (bool useOSCal)

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

4.56.3.8 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. 192)

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

- ArJoyHandler.h
- ArJoyHandler.cpp
- ArJoyHandler_LIN.cpp
- ArJoyHandler_WIN.cpp
4.57 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 (void)
  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 any-
  more.

- void checkKeys (void)
  int internal, use addKeyHandler, Checks for keys and handles them.

- int getKey (void)
  int internal, use addKeyHandler instead... Gets a key from the stdin if ones
  available, -1 if there aren’t any available.
4.57.1 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. 182), and before you create a key handler you should see if one is already there with Aria::getKeyHandler (p. 182).

You can attach a key handler to a robot with ArRobot::attachKeyHandler (p. 311) 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.

4.57.2 Member Enumeration Documentation

4.57.2.1 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.

4.57.3 Constructor & Destructor Documentation

4.57.3.1 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 checkKeys yourself like after you start a robot run or in its own thread or something along those lines

4.57.4 Member Function Documentation

4.57.4.1 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

4.57.4.2 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 was found and removed... false means that the remove failed because there was no key for that

4.57.4.3 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 was found
and removed... false means that the remove failed because there was no
key for that.

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

- ArKeyHandler.h
- ArKeyHandler.cpp
4.58 ArListPos Class Reference

has enum for position in list.

#include <ariaTypedefs.h>

Public Types

- enum Pos { FIRST = 1, LAST = 2 }

4.58.1 Detailed Description

has enum for position in list.

4.58.2 Member Enumeration Documentation

4.58.2.1 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
4.59 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.
- bool init (LogType type, LogLevel level, const char *fileName="")
  Initialize the logging utility.
- void close ()
  Close the logging utility.

4.59.1 Detailed Description

Logging utility class.

ArLog is a utility class to log all messages from Aria (p.181) to a chosen destination. 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.

4.59.2 Member Enumeration Documentation

4.59.2.1 enum ArLog::LogLevel

Enumeration values:

- Terse  Use terse logging.
- Normal Use normal logging.
- Verbose Use verbose logging.
4.59.2.2 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.

4.59.3 Member Function Documentation

4.59.3.1 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 Logging
- level level of logging
- fileName the name of the file for File type of logging

4.59.3.2 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
4.60  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.
4.60.1 Detailed Description

This class has static members to do common math operations.

4.60.2 Member Function Documentation

4.60.2.1 double ArMath::addAngle (double \textit{ang1}, double \textit{ang2}) [inline, static]

This adds two angles together and fixes the result to [-180, 180].

**Parameters:**
- \textit{ang1} first angle
- \textit{ang2} second angle, added to first

**Returns:**
- sum of the angles, in range [-180,180]

**See also:**
- subAngle (p. 205) , fixAngle (p. 204)

4.60.2.2 double ArMath::atan2 (double \textit{y}, double \textit{x}) [inline, static]

Finds the arctan of the given \textit{y/x} pair.

**Parameters:**
- \textit{y} the y distance
- \textit{x} the x distance

**Returns:**
- the angle \textit{y} and \textit{x} form

4.60.2.3 double ArMath::cos (double \textit{angle}) [inline, static]

Finds the cos, from angles in degrees.

**Parameters:**
- \textit{angle} angle to find the cos of, in degrees

**Returns:**
- the cos of the angle
See also:

\texttt{sin} (p. 205)

4.60.2.4  \texttt{double \ ArMath::degToRad (double} \ deg \texttt{)} \ [inline, static]  

Converts an angle in degrees to an angle in radians.

Parameters:

\texttt{deg}  the angle in degrees

Returns:

the angle in radians

See also:

\texttt{radToDeg} (p. 205)

4.60.2.5  \texttt{double \ ArMath::fabs (double} \ val \texttt{)} \ [inline, static]  

Finds the absolute value of a double.

Parameters:

\texttt{val}  the number to find the absolute value of

Returns:

the absolute value of the number

4.60.2.6  \texttt{double \ ArMath::fixAngle (double} \ angle \texttt{)} \ [inline, static]  

Takes an angle and returns the angle in range (-180,180].

Parameters:

\texttt{angle}  the angle to fix

Returns:

the angle in range (-180,180]

See also:

\texttt{addAngle} (p. 203) , \texttt{subAngle} (p. 205)
4.60.2.7 double ArMath::radToDeg (double \textit{rad})  [inline, static]

Converts an angle in radians to an angle in degrees.

**Parameters:**
\textit{rad} the angle in radians

**Returns:**
the angle in degrees

**See also:**
degToRad (p. 204)

4.60.2.8 int ArMath::roundInt (double \textit{val})  [inline, static]

Finds the closest integer to double given.

**Parameters:**
\textit{val} the double to find the nearest integer to

**Returns:**
the integer the value is nearest to

4.60.2.9 double ArMath::sin (double \textit{angle})  [inline, static]

Finds the sin, from angles in degrees.

**Parameters:**
\textit{angle} angle to find the sin of, in degrees

**Returns:**
the sin of the angle

**See also:**
cos (p. 203)

4.60.2.10 double ArMath::subAngle (double \textit{ang1}, double \textit{ang2})  [inline, static]

This subtracts one angle from another and fixes the result to [-180,180].
Parameters:

\textit{ang1} first angle
\textit{ang2} second angle, subtracted from first angle

Returns:

resulting angle, in range \([-180,180]\)

See also:

\textit{addAngle} (p. 203), \textit{fixAngle} (p. 204)

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

- ariaUtil.h
4.61 ArMode Class Reference

A class for different modes, mostly as related to keyboard input.

#include <ArMode.h>

Inheritance diagram for ArMode::

```
ArMode
    ├── ArModeBumps
    │    ├── ArModeCamera
    │    │    ├── ArModeGripper
    │    │    │    ├── ArModeIO
    │    │    │    │    ├── ArModeLaser
    │    │    │    │    │    ├── ArModePosition
    │    │    │    │    │    │    ├── ArModeSonar
    │    │    │    │    │    │    │    └── ArModeTeleop
    │    │    │    │    │    │    │    └── ArModeWander
    │    └── ArMode
    └── ArMode
```

Public Methods

- **ArMode** (ArRobot *robot, const char *name, char key, char key2)
  Constructor.

- virtual **~ArMode** (void)
  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.

4.61.1 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.

### 4.61.2 Constructor & Destructor Documentation

#### 4.61.2.1 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
- `key2` the first key to switch to this mode on... it can be '\0' if you don’t want to use this

### 4.61.3 Member Function Documentation

#### 4.61.3.1 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

#### 4.61.3.2 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

#### 4.61.3.3 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. 217), ArModeWander (p. 219), ArModeGripper (p. 214), ArModeCamera (p. 212), and ArModeSonar (p. 216).

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

- ArMode.h
- ArMode.cpp
4.62 ArModeCamera Class Reference

Mode for controlling the gripper.

#include <ArModes.h>

Inheritance diagram for ArModeCamera::

```
ArMode

ArModeCamera
```

Public Methods

- **ArModeCamera** (ArRobot *robot, const char *name, char key, char key2)
  
  Constructor.

- virtual ~ArModeCamera (void)
  
  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. 207)’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.

4.62.1 Detailed Description

Mode for controlling the gripper.
4.62.2 Member Function Documentation

4.62.2.1 `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. 209).
The documentation for this class was generated from the following files:

- ArModes.h
- ArModes.cpp
4.63  ArModeGripper Class Reference

Mode for controlling the gripper.
#include <ArModes.h>

Inheritance diagram for ArModeGripper::

```
  ArModeGripper
  |     |
  |     |
ArMode
```

Public Methods

- **ArModeGripper** (ArRobot *robot, const char *name, char key, char key2)
  Constructor.

- virtual ~ArModeGripper (void)
  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. 207)’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.

4.63.1 Detailed Description

Mode for controlling the gripper.
4.63.2 Member Function Documentation

4.63.2.1 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. 209).
The documentation for this class was generated from the following files:

- ArModes.h
- ArModes.cpp
4.64  ArModeSonar Class Reference

Mode for displaying the sonar.
#include <ArModes.h>

Inheritance diagram for ArModeSonar::

```
ArModeSonar
ArModeSonar
ArMode
```

Public Methods

- **ArModeSonar** (ArRobot *robot, const char *name, char key, char key2)
  Constructor.

- virtual ~ArModeSonar (void)
  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. 207)'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.

4.64.1  Detailed Description

Mode for displaying the sonar.
4.64.2 Member Function Documentation

4.64.2.1 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. 209).
The documentation for this class was generated from the following files:

- ArModes.h
- ArModes.cpp
4.65  ArModeTeleop Class Reference

Mode for teleoping the robot with joystick + keyboard.

#include <ArModes.h>

Inheritance diagram for ArModeTeleop:

```
  ArMode
  |      |
  v      v
ArModeTeleop
```

Public Methods

- **ArModeTeleop** (ArRobot *robot, const char *name, char key, char key2)
  
  Constructor.

- virtual ~ArModeTeleop (void)
  
  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.*

4.65.1  Detailed Description

Mode for teleoping the robot with joystick + keyboard.

4.65.2  Member Function Documentation

4.65.2.1  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. 209).

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

- ArModes.h
- ArModes.cpp
4.66 ArModeWander Class Reference

Mode for wandering around.

#include <ArModes.h>

Inheritance diagram for ArModeWander::

```
  ArMode
  |  
  v  
ArModeWander
```

Public Methods

- **ArModeWander** (ArRobot *robot, const char *name, char key, char key2)
  
  Constructor.

- virtual ~ArModeWander (void)

  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.

4.66.1 Detailed Description

Mode for wandering around.

4.66.2 Member Function Documentation

4.66.2.1 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. 209).

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

- ArModes.h
- ArModes.cpp
4.67 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. 290) pointer.

- **virtual bool exit ()=0**
  Close down the module and have it exit.

- **ArRobot * getRobot ()**
  Get the ArRobot (p. 290) pointer the module should be using.

- **void setRobot (ArRobot *robot)**
  Set the ArRobot (p. 290) pointer.

Protected Attributes

- **ArRobot * myRobot**
  Stored ArRobot (p. 290) pointer that the module should use.

4.67.1 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 dynamically loaded module of code. This is useful 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. 37) and the modules init() (p. 222) could instantiate the ArAction (p. 37) and add it to the supplied ArRobot (p. 290). The init() (p. 222) takes a reference to an ArRobot (p. 290). 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. 182) to find all the ArRobot (p. 290) instantiated. The module should do all its clean up in exit() (p. 221).

The user should derive their own class from ArModule and implement the init() (p. 222) and exit() (p. 221) functions. The users code should always clean up when exit() (p. 221) is called. exit() (p. 221) 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 ArModuleLoader (p. 224) to find the users module class and invoke it.

One thing to note about the use of code wrapped in ArModules and statically linking in that code. To be able to statically 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 statically link in multiple modules since they all have symbols with the same name.

See also ArModuleLoader (p. 224) 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.

4.67.2 Member Function Documentation

4.67.2.1 virtual bool ArModule::init (ArRobot ∗ robot, void ∗ argument = NULL) [pure virtual]

Initialize the module. The module should use the supplied ArRobot (p. 290) 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

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

- ArModule.h
- ArModule.cpp
4.68 ArModuleLoader Class Reference

Dynamic ArModule (p. 221) 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. 221).

- Status reload (const char *modName, ArRobot *robot, void *modArgument=NULL, bool quiet=false)
  
  Close and then reload an ArModule (p. 221).

- Status close (const char *modName, bool quiet=false)
  
  Close an ArModule (p. 221).

- void closeAll ()
  
  Close all open ArModule (p. 221).

4.68.1 Detailed Description

Dynamic ArModule (p. 221) loader.

The ArModuleLoader is used to load ArModules into a program and invoke them.

See also ArModule (p. 221) to see how to define an ArModule (p. 221).

See also the example programs simpleMod.cpp and simpleModule.cpp. For a more complete example, see the example programs joydriveActionMod.cpp and joydriveActionModule.cpp.
4.68 ArModuleLoader Class Reference

4.68.2 Member Enumeration Documentation

4.68.2.1 enum ArModuleLoader::Status

Enumeration values:
- STATUS_SUCCESS Load succeeded.
- 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.

4.68.3 Member Function Documentation

4.68.3.1 `ArModuleLoader::Status ArModuleLoader::close (const char * modName, bool quiet = false) [static]`

Close an ArModule (p. 221).

Calls ArModule::exit() (p. 221) 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

4.68.3.2 `ArModuleLoader::Status ArModuleLoader::load (const char * modName, ArRobot * robot, void * modArgument = NULL, bool quiet = false) [static]`

Load an ArModule (p. 221).

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 whether 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
will be passed the \texttt{ArRobot} reference that \texttt{load()} takes. This is the \texttt{ArRobot} that the \texttt{ArModule} will use for its processing.

**Parameters:**

- \texttt{modName} filename of the module without the extension (.dll or .so)
- \texttt{robot} \texttt{ArRobot} reference which the module is to use, this can be \texttt{NULL}
- \texttt{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 \texttt{NULL}
- \texttt{quiet} whether to print out a message if this fails or not, defaults to false

4.68.3.3 \texttt{ArModuleLoader::Status ArModuleLoader::reload} (const char * \texttt{modName}, \texttt{ArRobot} * \texttt{robot}, void * \texttt{modArgument} = \texttt{NULL}, bool \texttt{quiet} = false) [static]

Close and then reload an \texttt{ArModule}.

\texttt{reload()} is similar to \texttt{load()} (p. 225), except that it will call \texttt{close()} (p. 225) on the module and then call \texttt{load()} (p. 225).

**Parameters:**

- \texttt{modName} filename of the module without the extension (.dll or .so)

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

- \texttt{ArModuleLoader.h}
- \texttt{ArModuleLoader.cpp}
4.69 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.

- virtual ~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 std::string getError (int messageNumber)
  Get a human readable error message from an error code.

- virtual MutexType & getMutex ()
  Get a reference to the underlying mutex variable.

4.69.1 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 useful 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.
4.69.2 Member Enumeration Documentation

4.69.2.1 enum ArMutex::Status

Enumeration values:
- STATUS_FAILED_INIT - Failed to initialize.
- STATUS_FAILED - General failure.
- STATUS_ALREADY_LOCKED - Mutex already locked.

4.69.3 Member Function Documentation

4.69.3.1 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. 227).

4.69.3.2 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.

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

- ArMutex.h
- ArMutex_LIN.cpp
- ArMutex_WIN.cpp
4.70 ArPeriodicTask Class Reference

A periodic task base class.

#include <ArPeriodicTask.h>

Public Methods

- **ArPeriodicTask ()**
  Constructor.

- virtual **~ArPeriodicTask ()**
  Destructor.

- virtual bool **invoke ()**
  Invoke the task.

- virtual void **runTask ()=0**
  Function to override for the task.

- void **setRunning** (bool yesno)
  Function to set the task’s running variable.

- bool **isRunning ()**
  Function to tell if the task is running.

4.70.1 Detailed Description

A periodic task base class.

This class is a base class for running periodic tasks that need to be run in their own thread. Since the tasks are periodic and don’t need to be run constantly, they don’t need their own thread all the time. Since threads can be a restricted resource, there is this periodic task. The user simply needs to derive their own class from ArPeriodicTask and override the **runTask()** (p. 229) function. The function is guaranteed to be run in its own thread. It can run as long as it wants and when its done the thread will be released to be used by another ArPeriodicTask that may need it. To run the task, simply call **invoke()** (p. 229) on it. The **invoke()** (p. 229) can be called multiple times.

See also **ArTaskPool** (p. 408).

The documentation for this class was generated from the following files:
• ArPeriodicTask.h
• ArPeriodicTask.cpp
4.71 ArPose Class Reference

The class which represents a position.

```
#include <ariaUtil.h>
```

Inheritance diagram for ArPose::

```
ArPose
ArPoseWithTime
```

Public Methods

- **ArPose** (double x=0, double y=0, double th=0)
  Constructor, with optional initial values.

- **ArPose** (const ArPose &pose)
  Copy Constructor.

- virtual **~ArPose** (void)
  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)
  
  Gets the x position.

• double **getY** (void)
  
  Gets the y position.

• double **getTh** (void)
  
  Gets the heading.

• double **getThRad** (void)
  
  Gets the heading, in radians.

• void **getPose** (double ∗x, double ∗y, double ∗th=NULL)
  
  Gets the whole position in one function call.

• virtual double **findDistanceTo** (ArPose position)
  
  Finds the distance from this position to the given position.

• virtual double **findAngleTo** (ArPose position)
  
  Finds the angle between this position and the given position.

• virtual void **print** (void)
  
  Logs the coordinates using ArLog (p. 200).

### 4.71.1 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.

### 4.71.2 Constructor & Destructor Documentation

#### 4.71.2.1 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:
   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

4.71.3 Member Function Documentation

4.71.3.1 virtual double ArPose::findAngleTo (ArPose position)  
   [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

4.71.3.2 virtual double ArPose::findDistanceTo (ArPose position)  
   [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

4.71.3.3 void ArPose::getPose (double ∗ x, double ∗ y, double ∗ th  
   = NULL) [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  
th defaults to NULL so you can use this with just x and y.

Parameters:
   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
4.71.3.4 virtual void ArPose::setPose (ArPose \textit{position}) [inline, virtual]

Sets the position equal to the given position.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{position} the position value this instance should be set to
\end{itemize}

4.71.3.5 virtual void ArPose::setPose (double \textit{x}, double \textit{y}, double \textit{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.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{x} the position to set the x position to
  \item \textit{y} the position to set the y position to
  \item \textit{th} the position to set the th position to, default of 0
\end{itemize}

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

\begin{itemize}
  \item ariaUtil.h
\end{itemize}
4.72 ArPoseWithTime Class Reference

A subclass of pose that also has the time the pose was taken.

#include <ariaUtil.h>

Inheritance diagram for ArPoseWithTime:::

```
  ArPose
  ↑
  ArPoseWithTime
```

4.72.1 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:

- ariaUtil.h
4.73 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 double.
• 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.

4.73.1 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. 238), getInt() (p. 239), getDouble() (p. 239), getString() (p. 240), setBool() (p. 240), setInt() (p. 241), setDouble() (p. 241),
setString() (p. 242). A preference can also have a set of values of all the same
type. In the file would look like:

<k key> <int> <int> <int> ...
The 'set' accessors can deal with an arbitrary amount of values: `getBoolSet()` (p. 238), `getIntSet()` (p. 239), `getDoubleSet()` (p. 240), `getStringSet()` (p. 240), `setBoolSet()` (p. 241), `setIntSet()` (p. 242), `setDoubleSet()` (p. 241).

A preference can have an array of valid values. When the file is loaded, `ArPreferences` checks all the values from the file against the supplied valid values. The check is done with a string compare. It is most useful for string values. So it will apply to numbers as long as they are formatted in the correct way.

### 4.73.2 Member Enumeration Documentation

#### 4.73.2.1 `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.

### 4.73.3 Member Function Documentation

#### 4.73.3.1 `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.

#### 4.73.3.2 `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
4.73.3.3  double ArPref::getDouble (void)  [virtual]

Get the value as a double.
Get the value, formatting it correctly. If the preference is not of the double type or not found, it will return 0.0.

4.73.3.4  int ArPref::getDoubleSet (double * doubleArray, int size)  [virtual]

Get the value as multiple doubles.
Get the value, formatting 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

4.73.3.5  int ArPref::getInt (void)  [virtual]

Get the value as an integer.
Get the value, formatting it correctly. If the preference is not of the integer type or not found, it will return 0.

4.73.3.6  int ArPref::getIntSet (int * intArray, int size)  [virtual]

Get the value as multiple integers.
Get the value, formatting 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
4.73.3.7  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.

4.73.3.8  std::string ArPref::getString (void) [virtual]

Get the value as a std::string.
Get the value, formatting it correctly. If the preference is not of the string type or not found, it will return "".

4.73.3.9  int ArPref::getStringSet (std::string ∗ stringArray, int size) [virtual]

Get the value as multiple std::strings.
Get the value, formatting 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

4.73.3.10  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:
- val the value to set the preference to
- append create a new instance of this preference
4.73 ArPref Class Reference

4.73.3.11 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

4.73.3.12 bool ArPref::setDouble (double val, bool append = false)
   [virtual]

Set the value to be the supplied double.
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 to
   append create a new instance of this preference

4.73.3.13 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

4.73.3.14 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:

- \textit{val} the value to set the preference to
- \textit{append} create a new instance of this preference

4.73.3.15 \hspace{1em} \textbf{bool ArPref::setIntSet (bool \ append, \int \ count, \ldots)}
\hspace{1em} \textbf{[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:

- \textit{append} create a new instance of this preference
- \textit{count} the number of values in the parameter list

4.73.3.16 \hspace{1em} \textbf{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:

- \textit{val} the value to set the preference to
- \textit{append} create a new instance of this preference

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

- ArPref.h
- ArPref.cpp
4.74  ArPriorityResolver Class Reference

(Default resolver), takes the action list and uses the priority to resolve.

`#include <ArPriorityResolver.h>`

Inheritance diagram for ArPriorityResolver::

```
  ArResolver
  
  ArPriorityResolver
```

Public Methods

- **ArPriorityResolver**(void)
  
  Constructor.

- virtual **~ArPriorityResolver**(void)
  
  Destructor.

4.74.1  Detailed Description

(Default resolver), takes the action list and uses the priority to resolve.

This is the default resolver for **ArRobot** (p. 290), 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
4.75 ArPTZ Class Reference

Base class which handles the PTZ cameras.

#include <ArPTZ.h>

Inheritance diagram for ArPTZ:

```
ArPTZ
   ArAMPTU ArDPPTU ArSonyPTZ ArVCC4
```

Public Methods

- **ArPTZ** ([ArRobot *robot])
  - virtual ~ArPTZ (void)
    - 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)=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)=0
  The angle the camera was last told to pan to.

• virtual int getTilt (void)=0
  The angle the camera was last told to tilt to.

• virtual int getZoom (void)
  The value the camera was last told to zoom to.

• virtual bool canGetRealPanTilt (void)
  If this driver can tell the real pan/tilt angle.

• virtual int getRealPan (void)
  The angle the camera says its at.

• virtual int getRealTilt (void)
  The angle the camera says its at.

• virtual bool canGetRealZoom (void)
  If this driver can tell the real zoom.

• virtual int getRealZoom (void)
  The zoom the camera says its at.

• virtual int getMaxPosPan (void)=0
  Gets the highest positive degree the camera can pan to.

• virtual int getMaxNegPan (void)=0
  Gets the lowest negative degree the camera can pan to.

• virtual int getMaxPosTilt (void)=0
  Gets the highest positive degree the camera can tilt to.

• virtual int getMaxNegTilt (void)=0
  Gets the lowest negative degree the camera can tilt to.
• virtual int getMaxZoom (void)

  Gets the maximum value for the zoom on this camera.

• virtual int getMinZoom (void)

  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 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.

### 4.75.1 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**
pointer is passed in to the constructor, 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. 118) 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. 114), 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. 118) to setDeviceConnection 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 setDeviceConnection, 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).

### 4.75.2 Constructor & Destructor Documentation

#### 4.75.2.1 ArPTZ::ArPTZ (ArRobot * robot)

**Parameters:**

- **robot** The robot this camera is attached to, can be NULL

### 4.75.3 Member Function Documentation

#### 4.75.3.1 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 readPacket (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

### 4.75.3.2 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 one 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

### 4.75.3.3 virtual bool `ArPTZ::robotPacketHandler (ArRobotPacket * packet)` [inline, 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

### 4.75.3.4 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
4.75.3.5 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 will 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:

- ArPTZ.h
- ArPTZ.cpp
4.76  ArRangeBuffer Class Reference

This class is a buffer that holds ranging information.

#include <ArRangeBuffer.h>

Public Methods

• **ArRangeBuffer** (int size)
  Constructor.

• virtual **~ArRangeBuffer** (void)
  Destructor.

• size_t **getSize** (void)
  Gets the size of the buffer.

• void **setSize** (size_t size)
  Sets the size of the buffer.

• **ArPose getPosTaken** ()
  Gets the pose of the robot when readings were taken.

• void **setPosTaken** (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< ArPose *>::iterator readingIt)
  While doing an invalidation sweep a reading to the list to be invalidated.

• void **endInvalidationSweep** (void)
  Ends the invalidation sweep.

• std::list< ArPose*>* **getBuffer** (void)
  Gets a pointer to a list of readings.

• double **getClosestPolar** (double startAngle, double endAngle, ArPose position, unsigned int maxRange, double *angle=NULL)
4.76 ArRangeBuffer Class Reference

4.76.1 Detailed Description

This class is a buffer that holds ranging information.

4.76.2 Constructor & Destructor Documentation

4.76.2.1 ArRangeBuffer::ArRangeBuffer (int size)

Constructor.

Parameters:

size The size of the buffer, in number of readings

4.76.3 Member Function Documentation

4.76.3.1 void ArRangeBuffer::addReading (double x, double y)

Adds a new reading to the buffer.

Parameters:

x the x position of the reading
4.76.3.2  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

4.76.3.3  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. 254), endInvalidationSweep (p. 252)

4.76.3.4  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. 252); then for each reading you want to be in the buffer, call redoReading(double x, double y) (p. 255), then when you are done, call endRedoBuffer() (p. 253);

4.76.3.5  void ArRangeBuffer::endInvalidationSweep (void)

Ends the invalidation sweep.
See the description of beginInvalidationSweep, it describes how to use this function.
See also:
  beginInvalidationSweep (p. 252), invalidateReading (p. 254)

### 4.76.3.6 void ArRangeBuffer::endRedoBuffer (void)

End redoing the buffer.

For a description of how to use this, see beginRedoBuffer

### 4.76.3.7 std::list< ArPose * > * 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, i.e., 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

### 4.76.3.8 double ArRangeBuffer::getClosestBox (double \( x_1 \), double \( y_1 \), double \( x_2 \), double \( y_2 \), ArPose startPos, unsigned int maxRange, ArPose * readingPos = NULL)

Gets the closest reading, from a rectangular box, in robot LOCAL coords.

Gets the closest reading in a region defined by two points (opposite points of a rectangle).

**Parameters:**
- \( x_1 \) the x coordinate of one of the rectangle points
- \( y_1 \) the y coordinate of one of the rectangle points
- \( x_2 \) the x coordinate of the other rectangle point
- \( y_2 \) the y coordinate of the other rectangle point
- startPos the position to find the closest reading to (usually the robot's 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 coods for the definition of the coordinates, normally just ArRobot::getPosition

Returns:
if the return is $\geq 0$ and $\leq \text{maxRange}$ then this is the distance to the closest reading, if it is $\geq \text{maxRange}$, then there was no reading in the given section

4.76.3.9  double ArRangeBuffer::getClosestPolar (double startAngle, double endAngle, ArPose startPos, unsigned int maxRange, double * angle = NULL)

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 positive... 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 $\geq 0$ and $\leq \text{maxRange}$ then this is the distance to the closest reading, if it is $\geq \text{maxRange}$, then there was no reading in the given section

4.76.3.10  void ArRangeBuffer::invalidateReading (std::list<ArPose>::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:**
- *readingIt* the ITERATOR to the reading you want to get rid of

See also:
- beginInvalidationSweep, endInvalidationSweep (p. 252)

### 4.76.3.11 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:**
- *x* the x param of the coord to add to the buffer
- *y* the x param of the coord to add to the buffer

### 4.76.3.12 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:**
- *size* number of readings to set the buffer to

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

- ArRangeBuffer.h
- ArRangeBuffer.cpp
4.77  ArRangeDevice Class Reference

The class for all devices which return range info (laser, sonar).

#include <ArRangeDevice.h>

Inheritance diagram for ArRangeDevice:

```
ArRangeDevice
   ^
   |  ArIrrfDevice  ArRangeDeviceThreaded ArSonarDevice
   v
ArSick
```

Public Methods

- **ArRangeDevice** (size_t currentBufferSize, size_t cumulativeBufferSize, const char *name, unsigned int maxRange)
  Constructor.

- virtual ~ArRangeDevice (void)
  Destructor.

- virtual std::string getName (void)
  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 endAngle, double *angle=NULL)
  
  *Gets the closest current reading in the given polar region.*

• virtual double **cumulativeReadingPolar**(double startAngle, double endAngle, double *angle=NULL)
  
  *Gets the closest cumulative reading in the given polar region.*

• virtual double **currentReadingBox**(double x1, double y1, double x2, double y2, ArPose *readingPos=NULL)
  
  *Gets the closest current reading from the given box region.*

• virtual double **cumulativeReadingBox**(double x1, double y1, double x2, double y2, ArPose *readingPos=NULL)
  
  *Gets the closest current reading from the given box region.*

• virtual **ArRangeBuffer** *getCurrentRangeBuffer* (void)
  
  *Gets the current range buffer.*

• virtual **ArRangeBuffer** *getCumulativeRangeBuffer* (void)
  
  *Gets the cumulative range buffer.*

• virtual std::list< ArPose *>* **getCurrentBuffer** (void)
  
  *Gets the current buffer of readings.*

• virtual std::list< ArPose *>* **getCumulativeBuffer** (void)
  
  *Gets the current buffer of readings.*

• virtual std::list< ArSensorReading *>* **getRawReadings** (void)
  
  *Gets the raw unfiltered readings from the device.*

• 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 doCumulative=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.

### 4.77.1 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. 290) structure which check all of the ArRangeDevice s attached to a robot.

### 4.77.2 Constructor & Destructor Documentation

4.77.2.1 ArRangeDevice::ArRangeDevice (size_t currentBufferSize, size_t cumulativeBufferSize, const char * name, unsigned int maxRange)

Constructor.

**Parameters:**

- `currentBufferSize` number of readings to store in the current buffer
- `cumulativeBufferSize` 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
4.77.3 Member Function Documentation

4.77.3.1 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

4.77.3.2 double ArRangeDevice::cumulativeReadingBox (double x1, double y1, double x2, double y2, ArPose *pose = NULL) [virtual]

Gets the closest current reading from the given box region.
Gets the closest reading in a region defined by two points (opposite 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

4.77.3.3 double ArRangeDevice::cumulativeReadingPolar (double startAngle, double endAngle, double *angle = NULL) [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 posetive... with how the robot is set up,
that's 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 $\geq 0$ and $\leq$ `maxRange` then this is the distance to the closest reading, if it is $\geq$ `maxRange`, then there was no reading in the given section

### 4.77.3.4 double ArRangeDevice::currentReadingBox (double `x1`, double `y1`, double `x2`, double `y2`, ArPose ∗ `pose` = NULL) [virtual]

Gets the closest current reading from the given box region.

Gets the closest reading in a region defined by two points (opposite 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 $\geq 0$ and $\leq$ `maxRange` then this is the distance to the closest reading, if it is $\geq$ `maxRange`, then there was no reading in the given section
4.77.3.5  double ArRangeDevice::currentReadingPolar (double startAngle, double endAngle, double * angle = NULL)  [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 positive... 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

4.77.3.6  virtual std::list<ArSensorReading *> * ArRangeDevice::getRawReadings (void)  [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.

4.77.3.7  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. 105)

Reimplemented in ArRangeDeviceThreaded (p. 265).
4.77.3.8  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

4.77.3.9  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

4.77.3.10  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. 105)
Reimplemented in `ArRangeDeviceThreaded` (p. 266).

4.77.3.11  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. 105)
Reimplemented in `ArRangeDeviceThreaded` (p. 266).

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

- ArRangeDevice.h
• ArRangeDevice.cpp
4.78  ArRangeDeviceThreaded Class Reference

A range device which can run in its own thread.
#include <ArRangeDeviceThreaded.h>

Inheritance diagram for ArRangeDeviceThreaded:

```
+-----------------+            +------------+
|  ArThread       |   ↓       |  ArASyncTask|
|                 |       ↓   |             |
| ArRangeDevice   |   ↓       | ArRangeDeviceThreaded |
|                 |       ↓   |                   |
| ArASyncTask     |            | ArSick         |
```

Public Methods

- **ArRangeDeviceThreaded** (size_t currentBufferSize, size_t cumulativeBufferSize, const char *name, unsigned int maxRange)
  
  Constructor.

- virtual ~ArRangeDeviceThreaded (void)
  
  Destructor.

- virtual void *runThread (void *arg)=0
  
  The main run loop.

- void run (void)
  
  Runs the device in this thread.

- void runAsync (void)
  
  Runs the device in its own thread.

- virtual int lockDevice (void)
  
  Lock this device.

- virtual int tryLockDevice (void)
  
  Try to lock this device.

- virtual int unlockDevice (void)
Unlock this device.

4.78.1 Detailed Description
A range device which can run in its own thread.

4.78.2 Member Function Documentation

4.78.2.1 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. 105)
Reimplemented from ArRangeDevice (p. 261).

4.78.2.2 void ArRangeDeviceThreaded::run (void) [inline]
Runs the device in this thread.
Runs the device in the calling thread.
See also:
ArThread::stopRunning (p. 417)

4.78.2.3 void ArRangeDeviceThreaded::runAsync (void) [inline]
Runs the device in its own thread.
Runs the device in its own thread.
See also:
ArThread::stopRunning (p. 417)

4.78.2.4 virtual void* ArRangeDeviceThreaded::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. 418) or myRunning periodically in your loop. If the value goes false, the loop should exit and runThread() (p. 265) should return.
Reimplemented from `ArAsyncTask` (p. 106).

Reimplemented in `ArSick` (p. 367).

**4.78.2.5 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. 105)

Reimplemented from `ArRangeDevice` (p. 262).

**4.78.2.6 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 `ArAsyncTask` (p. 105)

Reimplemented from `ArRangeDevice` (p. 262).

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

- `ArRangeDeviceThreaded.h`
- `ArRangeDeviceThreaded.cpp`
4.79 ArRecurrentTask Class Reference

Recurrent task (runs in its own thread).

```
#include <ArRecurrentTask.h>
```

Inheritance diagram for ArRecurrentTask::

```
ArRecurrentTask
  ↓
ArASyncTask
  ↓
ArThread
```

Public Methods

- **ArRecurrentTask()**
  
  Constructor.

- **~ArRecurrentTask()**
  
  Destructor.

- **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.
4.79.1 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 ArRecurrentTask and define the task() (p. 268) 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 go() (p. 267) function to start the task. The status of the task can be checked with the done() (p. 268) function, which returns 0 if running, 1 if completed, and 2 if killed. go() (p. 267) can be called whenever the task is done to restart it. To stop the task in midstream, call reset() (p. 267). kill() kills off the thread, shouldn’t be used unless exiting the async task permanently.

4.79.2 Member Function Documentation

4.79.2.1 int ArRecurrentTask::done ()

Check if the task is running or not. 0 = running, 1 = finished normally, 2 = canceled

4.79.2.2 void ∗ ArRecurrentTask::runThread (void ∗ ptr) [virtual]

The main run loop. Override this function and put your task’s run loop here. Check the value of getRunning() (p. 418) or myRunning periodically in your loop. If the value goes false, the loop should exit and runThread() (p. 268) should return.

Reimplemented from ArASyncTask (p. 106).

4.79.2.3 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
4.80  ArResolver Class Reference

Resolves a list of actions and returns what to do.
#include <ArResolver.h>

Inheritance diagram for ArResolver::

```
ArResolver

ArPriorityResolver
```

Public Types

- typedef std::multimap<int, ArAction *> ActionMap
  Constructor.

Public Methods

- virtual ~ArResolver (void)
  Destructor.

- virtual ArActionDesired * resolve (ActionMap *actions, ArRobot *robot)=0
  Figure out a single ArActionDesired (p. 49) from a list of ArAction (p. 37) s.

- virtual std::string getName (void)
  Gets the name of the resolver.

- virtual std::string getDescription (void)
  Gets the long description for the resolver.

4.80.1  Detailed Description

Resolves a list of actions and returns what to do.

This class exists just for resolve, which will always have to be overridden. 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:

- ArResolver.h
4.81 ArRetFunctor Class Template Reference

Base class for functors with a return value.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor::

Public Methods

- virtual ~ArRetFunctor (void)
  Destructor.

- virtual void invoke (void)
  Invokes the functor.

- virtual void operator() ()
  Invokes the functor.

- virtual Ret invokeR (void)=0
  Invokes the functor with return value.

4.81.1 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. 133).
The documentation for this class was generated from the following file:

- ArFunctor.h
4.82 ArRetFunctor1 Class Template Reference

Base class for functors with a return value with 1 parameter.

```cpp
#include <ArFunctor.h>
```

Inheritance diagram for ArRetFunctor1::

```
ArFunctor
   |
   v
ArRetFunctor< Ret >
   |
   v
ArRetFunctor1
   |
   v
ArGlobalRetFunctor1 ArRetFunctor1C
```

Public Methods

- virtual ~`ArRetFunctor1` (void)
  
  Destructor.

- virtual Ret `invokeR` (void)=0
  
  Invokes the functor with return value.

- virtual Ret `invokeR` (P1 p1)=0
  
  Invokes the functor with return value.

4.82.1 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. 133).
4.82.2 Member Function Documentation

4.82.2.1 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. 166), and ArRetFunctor1C (p. 277).

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

- ArFunctor.h
4.83 ArRetFunctor1C Class Template Reference

Functor for a member function with return value and 1 parameter.

```cpp
#include <ArFunctor.h>
```

Inheritance diagram for ArRetFunctor1C::

```
ArFunctor
    ArRetFunctor< Ret >
        ArRetFunctor1< Ret, P1 >
            ArRetFunctor1C
```

Public Methods

- **ArRetFunctor1C (void)**
  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 (void)**
  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.*

4.83.1 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.133).

4.83.2 Constructor & Destructor Documentation

4.83.2.1 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

4.83.2.2 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.
4.83 ArRetFunctor1C Class Template Reference

Parameters:

- *func* member function pointer
- *p1* default first parameter

4.83.2.3 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

4.83.2.4 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

4.83.3 Member Function Documentation

4.83.3.1 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. 274).

4.83.3.2 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

### 4.83.3.3 template<class Ret, class T, class P1> virtual void
\[ \text{ArRetFunctor1C}<\text{Ret, T, P1}>::\text{setThis (T & obj)} \]
[inline, virtual]

Set the 'this' pointer.

Parameters:
\[ \text{obj} \] the 'this' pointer

### 4.83.3.4 template<class Ret, class T, class P1> virtual void
\[ \text{ArRetFunctor1C}<\text{Ret, T, P1}>::\text{setThis (T * obj)} \]
[inline, virtual]

Set the 'this' pointer.

Parameters:
\[ \text{obj} \] the 'this' pointer

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

- ArFunctor.h
4.84 ArRetFunctor2 Class Template Reference

Base class for functors with a return value with 2 parameters.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctor2::

```
ArFunctor
  ↓
ArRetFunctor<Ret>
  ↓
ArRetFunctor2
  ↓
ArGlobalRetFunctor2  ArRetFunctor2C
```

Public Methods

- virtual ~ArRetFunctor2 (void)  
  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.

4.84.1 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. 133).

4.84.2 Member Function Documentation

4.84.2.1 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 parameter
  p2 second parameter

Reimplemented in ArGlobalRetFunctor2 (p. 170), and ArRetFunctor2C (p. 284).

4.84.2.2 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. 170), and ArRetFunctor2C (p. 284).

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

- ArFunctor.h
4.85 ArRetFunctor2C Class Template Reference

Functor for a member function with return value and 2 parameters.

```
#include <ArFunctor.h>
```

Inheritance diagram for ArRetFunctor2C::

```
ArFunctor

ArRetFunctor< Ret >

ArRetFunctor2< Ret, P1, P2 >

ArRetFunctor2C
```

Public Methods

- **ArRetFunctor2C (void)**
  
  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 ~\texttt{ArRetFunctor2C} (void)

\emph{Destructor.}

virtual \texttt{Ret invokeR} (void)

\emph{Invokes the functor with return value.}

virtual \texttt{Ret invokeR} (P1 p1)

\emph{Invokes the functor with return value.}

virtual \texttt{Ret invokeR} (P1 p1, P2 p2)

\emph{Invokes the functor with return value.}

virtual void \texttt{setThis} (T *obj)

\emph{Set the 'this' pointer.}

virtual void \texttt{setThis} (T &obj)

\emph{Set the 'this' pointer.}

virtual void \texttt{setP1} (P1 p1)

\emph{Set the default parameter.}

virtual void \texttt{setP2} (P2 p2)

\emph{Set the default 2nd parameter.}

\section*{4.85.1 Detailed Description}

\texttt{template<class Ret, class T, class P1, class P2> class ArRetFunctor2C< 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 \texttt{ArFunctor} (p.133).
4.85.2 Constructor & Destructor Documentation

4.85.2.1 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

4.85.2.2 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

4.85.2.3 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 pointer
    p1 default first parameter
    p2 default second parameter

4.85.2.4 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
4.85.2.5 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

4.85.2.6 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 pointer
   p1 default first parameter
   p2 default second parameter

4.85.3 Member Function Documentation

4.85.3.1 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 parameter
   p2 second parameter

Reimplemented from ArRetFunctor2 (p. 280).

4.85.3.2 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. 280).

4.85.3.3 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

4.85.3.4 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

4.85.3.5 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

4.85.3.6 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
4.86 ArRetFunctorC Class Template Reference

Functor for a member function with return value.

#include <ArFunctor.h>

Inheritance diagram for ArRetFunctorC::

```
ArFunctor
  
ArRetFunctor< Ret >
  
ArRetFunctorC
```

Public Methods

- **ArRetFunctorC (void)**
  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 (void)**
  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.
4.86.1 Detailed Description

template<\text{class Ret, class } T > \text{ class } \text{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 \text{ArFunctor} (p. 133).

4.86.2 Constructor & Destructor Documentation

4.86.2.1 template<\text{class Ret, class } T > \text{ ArRetFunctorC< Ret, T >::ArRetFunctorC (T & obj, Ret(T::* func)(\text{void})) [\text{inline}]}

Constructor - supply function pointer.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{func} member function pointer
\end{itemize}

4.86.2.2 template<\text{class Ret, class } T > \text{ ArRetFunctorC< Ret, T >::ArRetFunctorC (T * obj, Ret(T::* func)(\text{void})) [\text{inline}]}

Constructor - supply function pointer.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{func} member function pointer
\end{itemize}

4.86.3 Member Function Documentation

4.86.3.1 template<\text{class Ret, class } T > \text{ virtual void ArRetFunctorC< Ret, T >::setThis (T & obj) [\text{inline, virtual}]}

Set the 'this' pointer.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{obj} the 'this' pointer
\end{itemize}
4.86.3.2 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:

- ArFunctor.h
4.87 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.

- ~ArRobot (void)
  Destructor.

- void run (bool stopRunIfNotConnected)
  Starts the instance to do processing.

- void runAsync (bool stopRunIfNotConnected)
  Starts the instance to do processing in its own new thread.

- bool isRunning (void)
  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)
  Gets the connection this instance uses.

- bool isConnected (void)
  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)
  
  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. 314).

• void **setVel** (double velocity)
  
  Sets the velocity
  
  See also:
  
  `clearDirectMotion` (p. 314).

• void **setVel2** (double leftVelocity, double rightVelocity)
  
  Sets the velocity of the wheels independently
  
  See also:
  
  `clearDirectMotion` (p. 314).

• void **move** (double distance)
  
  Move the given distance forward/backwards
  
  See also:
  
  `clearDirectMotion` (p. 314).

• 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. 314).

• void `setRotVel` (double velocity)

  Sets the rotational velocity

  See also:

  `clearDirectMotion` (p. 314).

• void `setDeltaHeading` (double deltaHeading)

  Sets the delta heading

  See also:

  `clearDirectMotion` (p. 314).

• bool `isHeadingDone` (double delta=0.0)

  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 `get` (void)

  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)

  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.

- `std::string getRobotName (void)`
  Returns the Robot’s name that is set in its onboard configuration.

- `std::string getRobotType (void)`
  Returns the type of the robot connected to.

- `std::string getRobotSubType (void)`
  Returns the subtype of the robot connected to.

- `double getMaxTransVel (void)`
  Gets the robots maximum translational velocity.

- `bool setMaxTransVel (double maxVel)`
  Sets the robots maximum translational velocity.

- `double getMaxRotVel (void)`
  Gets the robots maximum rotational velocity.

- `bool setMaxRotVel (double myMaxVel)`
  Sets the robots maximum rotational velocity.

- `ArPose getPose (void)`
  Gets the global position of the robot.

- `double getX (void)`
  Gets the global X location of the robot.

- `double getY (void)`
  Gets the global Y location of the robot.

- `double getTh (void)`
  Gets the global theta location of the robot.
* double `getVel` (void)
  Gets the translational velocity of the robot.

* double `getRotVel` (void)
  Gets the rotational velocity of the robot.

* double `getRobotRadius` (void)
  Gets the robot radius (in mm).

* double `getRobotDiagonal` (void)
  Gets the robot diagonal (half-height to diagonal of octagon) (in mm).

* double `getBatteryVoltage` (void)
  Gets the battery voltage of the robot.

* double `getLeftVel` (void)
  Gets the velocity of the left wheel.

* double `getRightVel` (void)
  Gets the velocity of the right wheel.

* int `getStallValue` (void)
  Gets the 2 bytes of stall return from the robot.

* bool `isLeftMotorStalled` (void)
  Returns true if the left motor is stalled.

* bool `isRightMotorStalled` (void)
  Returns true if the left motor is stalled.

* double `getControl` (void)
  Gets the control heading.

* int `getFlags` (void)
  Gets the flags values.

* bool `areMotorsEnabled` (void)
  Returns true if the motors are enabled.

* bool `areSonarsEnabled` (void)
returns true if the motors are enabled.

• double getCompass (void)
  Gets the compass heading from the robot.

• int getAnalogPortSelected (void)
  Gets which analog port is selected.

• unsigned char getAnalog (void)
  Gets the analog value.

• unsigned char getDigIn (void)
  Gets the byte representing digital input status.

• unsigned char getDigOut (void)
  Gets the byte representing digital output status.

• int getIOAnalogSize (void)
  Gets the number of bytes in the analog IO buffer.

• int getIODigInSize (void)
  Gets the number of bytes in the digital input IO buffer.

• int getIODigOutSize (void)
  Gets the number of bytes in the digital output IO buffer.

• int getIOAnalog (int num)
  Gets the n'th byte from the analog input data from the IO packet.

• unsigned char getIODigIn (int num)
  Gets the n'th byte from the digital input data from the IO packet.

• unsigned char getIODigOut (int num)
  Gets the n'th byte from the digital output data from the IO packet.

• bool hasTableSensingIR (void)
  Gets whether the robot has table sensing IR or not (see params in docs).

• bool isLeftTableSensingIRTriggered (void)
  Returns true if the left table sensing IR is triggered.

• bool isRightTableSensingIRTriggered (void)
Returns true if the right table sensing IR is triggered.

- `bool isLeftBreakBeamTriggered (void)`
  Returns true if the left break beam IR is triggered.

- `bool isRightBreakBeamTriggered (void)`
  Returns true if the right break beam IR is triggered.

- `ArTime getIOPacketTime (void)`
  Returns the time received of the last IO packet.

- `bool hasFrontBumpers (void)`
  Gets whether the robot has front bumpers (see params in docs).

- `bool hasRearBumpers (void)`
  Gets whether the robot has rear bumpers (see params in docs).

- `ArRobotParams getParams (void)`
  Gets the raw params for the robot.

- `ArPose getEncoderPose (void)`
  Gets the position of the robot according to the encoders.

- `int getMotorPacCount (void)`
  Gets the number of motor packets received in the last second.

- `int getSonarPacCount (void)`
  Gets the number of sonar returns received in the last second.

- `int getSonarRange (int num)`
  Gets the range of the last sonar reading for the given sonar.

- `bool isSonarNew (int num)`
  Find out if the given sonar has a new reading.

- `int getNumSonar (void)`
  Find the number of sonar there are.

- `ArSensorReading * getSonarReading (int num)`
  Returns the sonar reading for the given sonar.

- `int getClosestSonarRange (double startAngle, double endAngle)`
Returns the closest of the current sonar reading in the given range.

- `int getClosestSonarNumber` (double startAngle, double endAngle)
  Returns the number of the sonar that has the closest current reading in the given range.

- `std::string getName` (void)
  Gets the robot's name in ARIAs list.

- `void setName` (const char *name)
  Sets the robot's 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 globalPos)
  Changes the transform.

- `void setEncoderTransform` (ArPose transformPos)
  Changes the transform directly.

- `ArTransform getToLocalTransform` (void)
  This gets the transform from local coords to global coords.

- `ArTransform getToGlobalTransform` (void)
  This gets the transform for going from global coords to local coords.

- `void applyTransform` (ArTransform trans, bool doCumulative=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.

- **ArRangeDevice * findRangeDevice (const char *name)**
  
  Finds a range device in the robot’s list.

- **std::list< ArRangeDevice *> * getRangeDeviceList (void)**
  
  Gets the range device list.

- **bool hasRangeDevice (ArRangeDevice *device)**
  
  Finds whether a particular range device is attached to this robot or not.

- **double checkRangeDevicesCurrentPolar (double startAngle, double endAngle, double *angle=NULL)**
  
  Goes through all the range devices and checks them.

- **double checkRangeDevicesCumulativePolar (double startAngle, double endAngle, double *angle=NULL)**
  
  Goes through all the range devices and checks them.

- **double checkRangeDevicesCurrentBox (double x1, double y1, double x2, double y2, ArPose *readingPos=NULL)**

- **double checkRangeDevicesCumulativeBox (double x1, double y1, double x2, double y2, ArPose *readingPos=NULL)**

- **void setStateReflectionRefreshTime (int msec)**
  
  Sets the number of milliseconds between state reflection refreshes if the state has not changed.

- **int getStateReflectionRefreshTime (void)**
  
  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)**
4.87 ArRobot Class Reference

Add a connect callback.

- void remConnectCB (ArFunctor *functor)

Add a disconnect callback.

- void addFailedConnectCB (ArFunctor *functor, ArListPos::Pos position)

Add a callback for when a connection to the robot is failed.

- void remFailedConnectCB (ArFunctor *functor)

Remove a callback for when a connection to the robot is failed.

- void addDisconnect NormallyCB (ArFunctor *functor, ArListPos::Pos position)

Add a callback for when disconnect is called while connected.

- void remDisconnect NormallyCB (ArFunctor *functor)

Remove a callback for when disconnect is called while connected.

- void addDisconnectOnErrorCB (ArFunctor *functor, ArListPos::Pos position)

Add a callback for when disconnection happens because of an error.

- void remDisconnectOnErrorCB (ArFunctor *functor)

Remove a callback for when disconnection happens because of an error.

- void addRunExitCB (ArFunctor *functor, ArListPos::Pos position)

Add a callback for when the run loop exits for whatever reason.

- void remRunExitCB (ArFunctor *functor)

Remove a callback for when the run loop exits for whatever 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 tasks.

- void `remUserTask(const char *name)`
  - Removes a user task from the list of synchronous tasks by name.

- void `remUserTask(ArFunctor *functor)`
  - Removes a user task from the list of synchronous tasks by functor.

- ArSyncTask * `findUserTask(const char *name)`
  - Finds a user task by name.

- ArSyncTask * `findUserTask(ArFunctor *functor)`
  - Finds a user task by functor.

- void `printUserTasks(void)`
  - Logs the list of user tasks, strictly for your viewing pleasure.

- void `printAllTasks(void)`
  - 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 synchronous 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 **printActions** (void)
  
  Prints 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, ArPoseWithTime > *functor)
  
  Sets the encoderCorrectionCallback.

- **ArRetFunctor1**< double, ArPoseWithTime > * **getEncoderCorrectionCallback** (void)
  
  Gets the encoderCorrectionCallback.

- void **setCycleTime** (unsigned int ms)
  
  Sets the number of ms between cycles.

- unsigned int **getCycleTime** (void)
Gets the number of ms between cycles.

- **void setCycleChained**(bool cycleChained)
  Sets whether to chain the robot cycle to when we get in SIP packets.

- **bool isCycleChained**(void)
  Gets whether we chain the robot cycle to when we get in SIP packets.

- **void setConnectionTimeoutTime**(int nSecs)
  Sets the time without a response until connection assumed lost.

- **int getConnectionTimeoutTime**(void)
  Gets the time without a response until connection assumed lost.

- **ArTime getLastPacketTime**(void)
  Gets the time the last packet was received.

- **void setPoseInterpNumReadings**(size_t numReadings)
  Sets the number of packets back in time the ArInterpolation (p. 186) goes.

- **size_t getPoseInterpNumReadings**(void)
  Sets the number of packets back in time the position interpol goes.

- **int getPoseInterpPosition**(ArTime time,[ArPose *]position)
  Gets the position the robot was at at the given timestamp.

- **unsigned int getCounter**(void)
  Gets the Counter for the time through the loop.

- **ArRobotParams * getRobotParams**(void)
  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 exitOnEscape=true)
  Attaches a key handler.

- **ArKeyHandler * getKeyHandler**(void)
  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 synchronous 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.

## 4.87.1 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. 358) 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. 358) , **ArTcpConnection** (p. 411)

### 4.87.2 Member Enumeration Documentation

#### 4.87.2.1 enum ArRobot::WaitState

Enumeration values:

- **WAIT_CONNECTED** The robot has connected.
- **WAIT_FAILEDCONN** 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 interrupted by a signal.
- **WAIT_FAIL** The wait failed due to an error.

### 4.87.3 Constructor & Destructor Documentation

#### 4.87.3.1 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 stateReflection 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 initializes 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. 336) when the program is signaled.
4.87.4  Member Function Documentation

4.87.4.1  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. 305) 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. 306) , remAction (p. 327)

4.87.4.2  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. 243)) 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

4.87.4.3  void ArRobot::addConnectCB (ArFunctor * functor,
                             ArListPos::Pos position)

Adds a connect callback.
Adds a connect callback, which is an ArFunctor (p.133), created as an ArFunctorC (p.150). 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:
    functor  created from ArFunctorC (p.150) which refers to the function to call.
    position whether to place the functor first or last

See also:
remConnectCB (p.327)
4.87.4.4  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. 133), created as an ArFunctorC (p. 150). This whole list of disconnect normally callbacks is called when something calls disconnect if the instance isConnected. 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. 150) which refers to the function to call.

position  whether to place the functor first or last

See also:

remFailedConnectCB (p. 328)

4.87.4.5  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. 133), created as an ArFunctorC (p. 150). 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. 150) which refers to the function to call.

position  whether to place the functor first or last

See also:

remFailedConnectCB (p. 328)
4.87.4.6 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. 133), created as an ArFunctorC (p. 150). 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. 150) which refers to the function to call.

position whether to place the functor first or last

See also:

remFailedConnectCB (p. 328)

4.87.4.7 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. 273), created as an instance of ArRetFunctor1C (p. 275). The return is a boolean, while the functor takes an ArRobotPacket (p. 340) 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. 328)

4.87.4.8 void ArRobot::addRunExitCB (ArFunctor * functor, ArListPos::Pos position)

Adds a callback for when the run loop exits for whatever reason.

Adds a callback that is called when the run loop exits. The functor is which is an ArFunctor (p. 133), created as an ArFunctorC (p. 150). The whole list
of functors is called when the run loop exits. This is most useful for threaded programs that run the robot using `ArRobot::runAsync` (p. 330). This will allow user threads to know when the robot loop has exited.

**Parameters:**
- `functor` functor created from `ArFunctorC` (p. 150) which refers to the function to call.
- `position` whether to place the functor first or last

**See also:**
- `remRunExitCB` (p. 329)

### 4.87.4.9 bool ArRobot::addSensorInterpTask (const char ∗ name, int position, ArFunctor ∗ functor, ArTaskState::State ∗ state = NULL)

Adds a task under the sensor interp part of the synchronous 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. 150) which refers to the function to call.

**See also:**
- `remSensorInterpTask` (p. 329)

### 4.87.4.10 bool ArRobot::addUserTask (const char ∗ name, int position, ArFunctor ∗ functor, ArTaskState::State ∗ state = NULL)

Adds a user task to the list of synchronous 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 position number.

functor functor created from ArFunctorC (p. 150) which refers to the function to call.

See also:
remUserTask (p. 329)

4.87.4.11 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

4.87.4.12 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 weird 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
4.87 ArRobot Class Reference

See also:
   addConnectCB (p. 306)

See also:
   addFailedConnectCB (p. 308)

See also:
   runAsync (p. 330)

4.87.4.13 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 used 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 it’s 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 it’s still trying to connect, 1 if it connected, 2 if it failed

4.87.4.14 void ArRobot::attachKeyHandler (ArKeyHandler * keyHandler, bool exitOnEscape = true)

Attaches 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 attach
   exitOnEscape whether to exit when escape is pressed or not
4.87.4.15  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 is the robot is in a weird state (this is not
often). If the robot is connected via ArSerialConnection (p. 358) 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 separate 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

4.87.4.16  double ArRobot::checkRangeDevicesCumulativeBox
  (double x1, double y1, double x2, double y2, ArPose *
   readingPos = NULL)

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 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

4.87.4.17  double ArRobot::checkRangeDevicesCumulativePolar
        (double startAngle, double endAngle, double * angle = NULL)

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 positive... 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
  angle       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

4.87.4.18  double ArRobot::checkRangeDevicesCurrentBox (double x1, double y1,
                                                   double x2, double y2, ArPose * readingPos
                                                   = NULL)

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
\texttt{y2} the y coordinate of the other rectangle point

\texttt{readingPos} a pointer to a position in which to store the location of the closest position

Returns:
if the return is \( \geq 0 \) then this is the distance to the closest reading, if it is \(< 0\) then there were no readings in the given region

\textbf{4.87.4.19} \hspace{1em} \texttt{double ArRobot::checkRangeDevicesCurrentPolar (double startAngle, double endAngle, double \& angle = NULL)}

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 positive... 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:
\begin{itemize}
  \item \texttt{startAngle} where to start the slice
  \item \texttt{endAngle} where to end the slice, going clockwise from \texttt{startAngle}
  \item \texttt{angle} a pointer return of the angle to the found reading
\end{itemize}

Returns:
if the return is \( \geq 0 \) then this is the distance to the closest reading, if it is \(< 0\) then there were no readings in the given region

\textbf{4.87.4.20} \hspace{1em} \texttt{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{itemize}
  \item \texttt{setDirectMotionPrecedenceTime} (p. 332)
  \item \texttt{getDirectMotionPrecedenceTime} (p. 319)\end{itemize}
4.87.4.21   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

4.87.4.22   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

4.87.4.23   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

4.87.4.24   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

### 4.87.4.25  
`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:

- **command**: the command number to send
- **str**: the character array to send with the command
- **size**: length of the array to send

Returns:

whether the command could be sent or not

### 4.87.4.26  
`void ArRobot::disableMotors ()`

Disables the motors on the robot.

This command disables the motors on the robot, if it is connected.

### 4.87.4.27  
`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)

### 4.87.4.28  
`void ArRobot::enableMotors ()`

Enables the motors on the robot.

This command enables the motors on the robot, if it is connected.
4.87.4.29  **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 finds 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

4.87.4.30  **ArRangeDevice** * ArRobot::findRangeDevice (const char * 
        * name)

Finds a rangeDevice in the robot’s list.

**Parameters:**
  
  * name  remove the first device with this name

**Returns:**
  
  if found, a range device with the given name, if not found NULL

4.87.4.31  **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. 403) for the first task found with that functor

4.87.4.32  **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. 403) for the first task found with that name
4.87.4.33  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. 403) for the first task found with that functor

4.87.4.34  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. 403) for the first task found with that name

4.87.4.35  ArResolver::ActionMap* ArRobot::getActionMap (void)

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. 306) and ArRobot::remAction (p. 327)...
This is just for the things like ArActionGroup (p. 58) that want to deactivate
or activate all the actions (well, only deactivating everything makes sense).

Returns:
the actions the robot is using

4.87.4.36  int ArRobot::getConnectionTimeoutTime (void)

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.
4.87.4.37 double ArRobot::getControl (void) [inline]

Gets the control heading.
Gets the control heading as an offset from the current heading.

See also:  
   getTh (p. 293)

4.87.4.38 unsigned int ArRobot::getCycleTime (void)

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 interpreted, 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

4.87.4.39 ArDeviceConnection * ArRobot::getDeviceConnection (void)

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:
   ArDeviceConnection (p. 118) , ArSerialConnection (p. 358) , ArTcpConnection (p. 411)

4.87.4.40 unsigned int ArRobot::getDirectMotionPrecedenceTime (void)

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:
setDirectMotionPrecedenceTime (p. 332), clearDirectMotion (p. 314)

4.87.4.41 ArRetFunctor1< double, ArPoseWithTime > *
ArRobot::getEncoderCorrectionCallback (void)

Gets the encoderCorrectionCallback.
This gets the encoderCorrectionCB, see setEncoderCorrectionCallback for de-
tails.

Returns:
the callback, or NULL if there isn’t one

4.87.4.42 ArTime ArRobot::getLastPacketTime (void)

Gets the time the last packet was received.
This gets the ArTime (p. 421) that the last packet was received at

Returns:
the time the last packet was received

4.87.4.43 int ArRobot::getPoseInterpPosition (ArTime time Stamp,
ArPose * position) [inline]

Gets the position the robot was at at the given timestamp.

See also:
ArInterpolation::getPose (p. 187)

4.87.4.44 std::list< ArRangeDevice * > * ArRobot::getRange-
DeviceList (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 devices attached to this robot

4.87.4.45  ArRobotParams * ArRobot::getRobotParams (void)

Gets the parameters the robot is using.

Returns:
the ArRobotParams (p. 349) instance the robot is using for its parameters

4.87.4.46  int ArRobot::getSonarRange (int num)

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. 296)

4.87.4.47  ArSensorReading * ArRobot::getSonarReading (int num)

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. 353), 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
4.87.4.48 int ArRobot::getStateReflectionRefreshTime (void)

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 cycle time, it'll simply happen every cycle.

**Returns:**

the state reflection refresh time

4.87.4.49 ArSyncTask * ArRobot::getSyncTaskRoot (void)

This gets the root of the synchronous 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 synchronous task tree

**See also:**

ArSyncTask (p. 403)

4.87.4.50 ArTransform ArRobot::getToGlobalTransform (void)

This gets the transform from local coords to global coords.

**Returns:**

an ArTransform (p. 423) which can be used for transforming a position in local coordinates to one in global coordinates

4.87.4.51 ArTransform ArRobot::getToLocalTransform (void)

This gets the transform for going from global coords to local coords.

**Returns:**

an ArTransform (p. 423) which can be used for transforming a position in global coordinates to one in local coordinates
4.87.4.52 bool ArRobot::hasRangeDevice (ArRangeDevice * device)

Finds whether a particular range device is attached to this robot or not.

Parameters:
  device the device to check for

4.87.4.53 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.

4.87.4.54 bool ArRobot::isConnected (void) [inline]

Questions whether the robot is connected or not.

Returns:
  true if connected to a robot, false if not

4.87.4.55 bool ArRobot::isDirectMotion (void)

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. 314)

4.87.4.56 bool ArRobot::isHeadingDone (double delta = 0.0)

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).

4.87.4.57 bool ArRobot::isMoveDone (double \textit{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 \textit{delta}
= 0 (default), the \textit{delta} distance is what was set with setMoveDoneDist, you
can get the distance with getMoveDoneDist

Parameters:
\textit{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).

4.87.4.58 bool ArRobot::isRunning (void)

Returns whether the robot is currently running or not.

Returns:
true if the robot is currently running in a run or runAsync, otherwise false

4.87.4.59 bool ArRobot::isSonarNew (int \textit{num})

Find out if the given sonar has a new reading.

Parameters:
\textit{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
4.87.4.60 bool ArRobot::loadParamFile (const char * file)

Loads a parameter file (replacing all other params).

Returns:
true if the file could be loaded, false otherwise

4.87.4.61 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.

4.87.4.62 void ArRobot::move (double distance)

Move the given distance forward/backwards

See also:
clearDirectMotion (p. 314).

Tells the robot to move the specified distance forward/backwards, if the con-
structor 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

4.87.4.63 void ArRobot::moveTo (ArPose poseTo, ArPose
poseFrom, bool doCumulative = true)

Moves the robot’s RW position to reflect pose From => pose To.

Parameters:
poseTo the absolute real world position to move to
poseFrom the original absolute real world position
doCumulative whether to update the cumulative buffers or not
4.87.4.64  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 of the robot
- doCumulative whether to update the cumulative buffers or not

4.87.4.65  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. 308), remPacketHandler (p. 328)

4.87.4.66  void ArRobot::printAllTasks (void)

Logs the list of all tasks, strictly for your viewing pleasure.

See also:
- ArLog (p. 200)

4.87.4.67  void ArRobot::printUserTasks (void)

Logs the list of user tasks, strictly for your viewing pleasure.

See also:
- ArLog (p. 200)

4.87.4.68  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

4.87.4.69 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

4.87.4.70 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. 306)

4.87.4.71 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. 307)

4.87.4.72 void ArRobot::remDisconnectOnErrorCB (ArFunctor * functor)

Removes a callback for when disconnection happens because of an error.
Parameters:
  \textit{functor} the functor to remove from the list of connect callbacks

See also:
  \texttt{addDisconnectOnErrorCB} (p. 307)

4.87.4.73 \hspace{.1em} \texttt{void ArRobot::remFailedConnectCB (ArFunctor * functor)}\hspace{.1em}

Removes a callback for when a connection to the robot is failed.

Parameters:
  \textit{functor} the functor to remove from the list of connect callbacks

See also:
  \texttt{addFailedConnectCB} (p. 308)

4.87.4.74 \hspace{.1em} \texttt{void ArRobot::remPacketHandler (ArRetFunctor1<bool, ArRobotPacket *> * functor)}\hspace{.1em}

Removes a packet handler from the list of packet handlers.

Parameters:
  \textit{functor} the functor to remove from the list of packet handlers

See also:
  \texttt{addPacketHandler} (p. 308)

4.87.4.75 \hspace{.1em} \texttt{void ArRobot::remRangeDevice (ArRangeDevice * device)}\hspace{.1em}

Remove a range device from the robot’s list, by instance.

Parameters:
  \textit{device} remove the first device with this pointer value

4.87.4.76 \hspace{.1em} \texttt{void ArRobot::remRangeDevice (const char * name)}\hspace{.1em}

Remove a range device from the robot’s list, by name.

Parameters:
  \textit{name} remove the first device with this name
4.87 ArRobot Class Reference

4.87.4.77 void ArRobot::remRunExitCB (ArFunctor * functor)

Removes a callback for when the run loop exits for whatever reason.

Parameters:
   functor the functor to remove from the list of run exit callbacks

See also:
   addRunExitCB (p. 308)

4.87.4.78 void ArRobot::remSensorInterpTask (ArFunctor * functor)

Removes a sensor interp tasks by functor.

See also:
   addSensorInterpTask (p. 309) , remSensorInterpTask(std::string name)

4.87.4.79 void ArRobot::remSensorInterpTask (const char * name)

Removes a sensor interp tasks by name.

See also:
   addSensorInterpTask (p. 309) , remSensorInterpTask(ArFunctor *functor) (p. 329)

4.87.4.80 void ArRobot::remUserTask (ArFunctor * functor)

Removes a user task from the list of synchronous tasks by functor.

See also:
   addUserTask (p. 309) , remUserTask(std::string name)

4.87.4.81 void ArRobot::remUserTask (const char * name)

Removes a user task from the list of synchronous tasks by name.

See also:
   addUserTask (p. 309) , remUserTask(ArFunctor *functor) (p. 329)
4.87.4.82 void ArRobot::robotLocker(void)

Robot locker, internal.
This just locks the robot, so that its locked for all the user tasks

4.87.4.83 void ArRobot::robotUnlocker(void)

Robot unlocker, internal.
This just unlocks the robot

4.87.4.84 void ArRobot::run(bool stopRunIfNotConnected)

Starts the instance to do processing.
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

4.87.4.85 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

4.87.4.86 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...

### 4.87.4.87 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 interpreted, 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

### 4.87.4.88 void ArRobot::setDeadReconPose (ArPose pose)

Sets the dead recon position of the robot.

**Parameters:**

- **pose** the position to set the dead recon position to

### 4.87.4.89 void ArRobot::setDeltaHeading (double deltaHeading)

Sets the delta heading

**See also:**

- **clearDirectMotion** (p. 314).

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
4.87.4.90  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. 118), ArSerialConnection (p. 358), ArTcpConnection (p. 411)

4.87.4.91  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. 332) , clearDirectMotion (p. 314)

4.87.4.92  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. 273) created as an ArRetFunctor1C (p. 275), that will be the callback... call this function NULL to clear the callback
See also:
getEncoderCorrectionCallback (p. 320)

4.87.4.93 void ArRobot::setEncoderTransform (ArPose
transformPos)

Changes the transform directly.

Parameters:
transformPos the position to transform to

4.87.4.94 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

4.87.4.95 void ArRobot::setHeading (double heading)

Sets the heading

See also:
clearDirectMotion (p. 314).

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

4.87.4.96 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 otherwise

4.87.4.97  
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 otherwise

4.87.4.98  
void ArRobot::setRotVel (double `velocity`)  

Sets the rotational velocity

See also:  
clearDirectMotion (p. 314).

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

4.87.4.99  
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 cycle time, it’ll simply happen every cycle.
Parameters:

\textit{mSec} the refresh time, in milliseconds, non-negative, if negative is given, then the value will be 0

4.87.4.100 \hspace{1em} \textbf{void ArRobot::setVel} (double \textit{velocity})

Sets the velocity

See also:

\texttt{clearDirectMotion} (p. 314).

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:

\textit{velocity} the desired translational velocity of the robot

4.87.4.101 \hspace{1em} \textbf{void ArRobot::setVel2} (double \textit{leftVelocity}, double \textit{rightVelocity})

Sets the velocity of the wheels independently

See also:

\texttt{clearDirectMotion} (p. 314).

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:

\textit{leftVelocity} the desired velocity of the left wheel

\textit{rightVelocity} the desired velocity of the right wheel

4.87.4.102 \hspace{1em} \textbf{void ArRobot::stateReflector} (void)

State Reflector, internal.

If state reflecting (really direct motion command reflecting) was enabled in the constructor \texttt{(ArRobot::ArRobot} (p. 305)) 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.

4.87.4.103  void ArRobot::stop (void)

Stops the robot

See also:
   clearDirectMotion (p. 314).

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:
   setDirectMotionPrecedenceTime (p. 332) , getDirectMotion-
   PrecedenceTime (p. 319) , clearDirectMotion (p. 314)

4.87.4.104  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.

4.87.4.105  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. 337).
Parameters:
  msecs milliseconds in which to wait for the ArRobot to connect

Returns:
  WAIT_CONNECTED for success

See also:
  waitForConnectOrConnFail (p. 337), wakeAllWaitingThreads (p. 338), wakeAllConnWaitingThreads (p. 338), wakeAllRunExitWaitingThreads (p. 338)

4.87.4.106 ArRobot::WaitState ArRobot::waitForConnectOrConnFail (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. 336)

4.87.4.107 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
4.87.4.108 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:
wakeAllWaitingThreads (p. 338) , wakeAllRunExitWaitingThreads (p. 338)

4.87.4.109 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:
wakeAllWaitingThreads (p. 338) , wakeAllRunExitWaitingThreads (p. 338)

4.87.4.110 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:
wakeAllWaitingThreads (p. 338) , wakeAllConnWaitingThreads (p. 338)

4.87.4.111 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. 338) , wakeAllRunExitWaitingThreads (p. 338)

The documentation for this class was generated from the following files:
• ArRobot.h
• ArRobot.cpp
4.88 ArRobotPacket Class Reference

Represents the packets sent to the robot as well as those received from it.

#include <ArRobotPacket.h>

Inheritance diagram for ArRobotPacket::

```
+------------------
| ArBasePacket     
+------------------
|                  
| ArRobotPacket    
```

Public Methods

- **ArRobotPacket** (unsigned char sync1=0xfa, unsigned char sync2=0xfb)
  Constructor.

- virtual ~ArRobotPacket (void)
  Destructor.

- bool verifyCheckSum (void)
  returns true if the checksum matches what it should be.

- ArTypes::UByte getID (void)
  returns the ID of the packet (first byte of data).

- ArTypes::Byte2 calcCheckSum (void)
  returns the checksum, probably used only internally.

- virtual void finalize (void)
  Finalizes 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.
4.88 ArRobotPacket Class Reference

4.88.1 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 opposite
endian from intel. Also has the getID for convenience.

You can just look at the documentation for the ArBasePacket (p. 107) except
for the 4 new functions here, verifyCheckSum, getID, print, and calcCheckSum.

4.88.2 Constructor & Destructor Documentation

4.88.2.1 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

4.88.3 Member Function Documentation

4.88.3.1 void ArRobotPacket::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. 111).

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

- ArRobotPacket.h
- ArRobotPacket.cpp
4.89  **ArRobotPacketReceiver Class Reference**

Given a device connection it receives packets from the robot through it.

```cpp
#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** (void)
  
  Destructor.

- **ArRobotPacket** * receivePacket** (unsigned int msWait=0)
  
  Receives a packet from the robot if there is one available.

- **void** setDeviceConnection** (ArDeviceConnection *deviceConnection)
  
  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.

### 4.89.1 Detailed Description

Given a device connection it receives packets from the robot through it.
4.89.2 Constructor & Destructor Documentation

4.89.2.1 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

4.89.2.2 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

4.89.3 Member Function Documentation

4.89.3.1 ArRobotPacket * ArRobotPacketReceiver::receivePacket (unsigned int msWait = 0)

Receives a packet from the robot if there is one available.
Parameters:
- **msWait** how long to block for the start of a packet, nonblocking if 0

Returns:
NULL if there are no packets in allotted 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
4.90 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 (void)
  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 *deviceConnection)
  Sets the device this instance sends commands to.

- ArDeviceConnection *getDeviceConnection (void)
  Gets the device this instance sends commands to.
4.90.1 Detailed Description

Given a device connection this sends commands through it to the robot.

4.90.2 Constructor & Destructor Documentation

4.90.2.1 ArRobotPacketSender::ArRobotPacketSender (unsigned char \textit{sync1} = 0xfa, unsigned char \textit{sync2} = 0xfb)

Constructor without an already assigned device connection.

Parameters:
- \textit{sync1} first byte of the header this sender will send, this should be left as the default in nearly all cases, i.e. don’t mess with it
- \textit{sync2} second byte of the header this sender will send, this should be left as the default in nearly all cases, i.e. don’t mess with it

4.90.2.2 ArRobotPacketSender::ArRobotPacketSender (ArDeviceConnection \& \textit{deviceConnection}, unsigned char \textit{sync1} = 0xfa, unsigned char \textit{sync2} = 0xfb)

Constructor with assignment of a device connection.

Parameters:
- \textit{sync1} first byte of the header this sender will send, this should be left as the default in nearly all cases, i.e. don’t mess with it
- \textit{sync2} second byte of the header this sender will send, this should be left as the default in nearly all cases, i.e. don’t mess with it

4.90.3 Member Function Documentation

4.90.3.1 bool ArRobotPacketSender::com (unsigned char \textit{number})

Sends a command to the robot with no arguments.

Parameters:
- \textit{command} the command number to send

Returns:
- whether the command could be sent or not
4.90.3.2  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 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

4.90.3.3  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

4.90.3.4  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
4.90.3.5  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:
  command  the command number to send
  str  the character array to send with the command
  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
4.91 ArRobotParams Class Reference

Contains the robot parameters, according to the parameter file.

#include <ArRobotParams.h>

Public Methods

- **ArRobotParams** (void)
  
  Constructor.

- virtual **~ArRobotParams** (void)
  
  Destructor.

- void **init** (ArRobotParamFile *param)
  
  Given the robot parameters in preference form, fills in this instance.

- std::string **getClass** (void)
  
  Returns the class from the parameter file.

- std::string **getSubClass** (void)
  
  Returns the subclass from the parameter file.

- double **getRobotRadius** (void)
  
  Returns the robot’s radius.

- double **getRobotDiagonal** (void)
  
  Returns the robot diagonal (half-height to diagonal of octagon).

- bool **isHolonomic** (void)
  
  Returns whether the robot is holonomic or not.

- bool **hasMoveCommand** (void)
  
  Returns if the robot has a built in move command.

- int **getMaxVelocity** (void)
  
  Returns the max velocity of the robot.

- int **getMaxRotVelocity** (void)
  
  Returns the max rotational velocity of the robot.

- bool **getRequestIOPackets** (void)
Returns true if IO packets are automatically requested upon connection to the robot.

- double `getAngleConvFactor` (void)  
  Returns the angle conversion factor.

- double `getDistConvFactor` (void)  
  Returns the distance conversion factor.

- double `getVelConvFactor` (void)  
  Returns the velocity conversion factor.

- double `getRangeConvFactor` (void)  
  Returns the sonar range conversion factor.

- double `getDiffConvFactor` (void)  
  Returns the wheel velocity difference to angular velocity conv factor.

- double `getVel2Divisor` (void)  
  Returns the multiplier for VEL2 commands.

- bool `haveTableSensingIR` (void)  
  Returns true if the robot has table sensing IR.

- bool `haveNewTableSensingIR` (void)  
  Returns true if the robot’s table sensing IR bits are sent in the 4th-byte of the IO packet.

- bool `haveFrontBumpers` (void)  
  Returns true if the robot has front bumpers.

- int `numFrontBumpers` (void)  
  Returns the number of front bumpers.

- bool `haveRearBumpers` (void)  
  Returns true if the robot has rear bumpers.

- int `numRearBumpers` (void)  
  Returns the number of rear bumpers.

- int `getNumSonar` (void)  
  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)
  Returns if the robot has a laser (according to param file).

• std::string getLaserPort (void)
  What port the laser is on.

• bool getLaserPowerControlled (void)
  If the laser power is controlled by the serial port lines.

• bool getLaserFlipped (void)
  If the laser is flipped on the robot.

• int getLaserX (void)
  The X location of the laser.

• int getLaserY (void)
  The Y location of the laser.

4.91.1 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
4.92 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** (void)
  
  Destructor.

- void **clear** (void)
  
  Clears all quadrants.

- void **update** (double angle)
  
  Updates the appropriate quadrant for the given angle.

- bool **didAll** (void)
  
  Returns true if the all of the quadrants have been gone through.

4.92.1 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
4.93 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 thPos=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 sonar reading.*

- double **getY** (void)
  
  *Gets the Y location of the sonar reading.*

- **ArPose getPose** (void)
  
  *Gets the position of the reading*

  **Returns:**
  
  *the position of the reading (ie where the sonar pinged back).*

- 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.*

- double **getSensorTh** (void)
  
  *Gets the heading of the sensor on the robot.*

- **ArPose getSensorPosition** (void)
  
  *Gets the sensor's 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 location 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.*

• **ArPose getPosTaken** (void)  
  
  *Gets the position of the robot when the reading was received.*

• unsigned int **getCounterTaken** (void)  
  
  *Gets the counter from when the reading arrived.*

• void **newData** (int range, double x, double y, double th, **ArTransform** trans, unsigned int counter, **ArTime** timeTaken)  
  
  *Takes the data and makes the reading reflect it.*

• void **ArSensorReading::newData** (int sx, int sy, double x, double y, double th, **ArTransform** trans, unsigned int counter, **ArTime** timeTaken)  
  
  *Takes the data and makes the reading reflect it.*

• void **resetSensorPosition** (int xPos, int 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.*

4.93.1 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 getRange) will be -1, and the counter it was taken (from getCounterTaken) will be 0, also it will never be new (from isNew)
4.93.2 Constructor & Destructor Documentation

4.93.2.1 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)

4.93.3 Member Function Documentation

4.93.3.1 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

4.93.3.2 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. 356)

4.93.3.3 int ArSensorReading::getRange (void) [inline]

Gets the range of the reading.

Returns:
  the distance return from the sensor (how far from the robot)
4.93.3.4 ArPose ArSensorReading::getSensorPosition (void) [inline]

Gets the sensors position on the robot.

**Returns:**
the position of the sensor on the robot

4.93.3.5 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

4.93.3.6 void ArSensorReading::newData (int range, double x, double y, double th, ArTransform trans, unsigned int counter, ArTime timeTaken)

Takes the data and makes the reading reflect it.

**Parameters:**
*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)
*counter* the counter from the robot when the sensor reading was taken

4.93.3.7 void ArSensorReading::resetSensorPosition (int xPos, int 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:

- ArSensorReading.h
- ArSensorReading.cpp
4.94 ArSerialConnection Class Reference

For connecting to devices through a serial port.

#include <ArSerialConnection.h>

Inheritance diagram for ArSerialConnection::

```
ArDeviceConnection
    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** (void)
  
  Constructor.

- virtual `~ArSerialConnection` (void)
  
  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.

- 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 msWait=0)
  Reads data from connection.
• virtual int write (const char *data, unsigned int size)
  Writes data to connection.
• virtual std::string 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.
• 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.

4.94.1 Detailed Description

For connecting to devices through a serial port.

4.94.2 Member Enumeration Documentation

4.94.2.1 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.
OPEN_COULD_NOT_SET_BAUD  Baud rate valid, but could not set it.
OPEN_ALREADY_OPEN  Connection was already open.

4.94.3  Member Function Documentation

4.94.3.1  bool ArSerialConnection::close (void)  [virtual]

Closes the connection.

Returns:
whether the close succeeded or not

Reimplemented from ArDeviceConnection (p. 120).

4.94.3.2  int ArSerialConnection::getBaud (void)

Gets what the current baud rate is set to.

Returns:
the current baud rate of the connection

4.94.3.3  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

4.94.3.4  std::string 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. 120).
4.94.3.5  int ArSerialConnection::getStatus (void)  [virtual]

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

Returns:
the status of the connection

See also:
getStatusMessage (p. 121)

Reimplemented from ArDeviceConnection (p. 120).

4.94.3.6  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. 121).

4.94.3.7  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. 121).

4.94.3.8  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. 360)

4.94.3.9  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. 363), writePacket (p. 123)

Reimplemented from ArDeviceConnection (p. 122).

4.94.3.10  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. 360)
4.94.3.11  bool ArSerialConnection::setHardwareControl (bool
hardwareControl)

Sets whether to enable or disable the hardware control lines.

**Parameters:**

  *hardwareControl* true to enable hardware control of lines

**Returns:**

  true if the set succeeded

4.94.3.12  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. 360)

4.94.3.13  int ArSerialConnection::write (const char ∗data,
unsigned int size)  [virtual]

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. 362), writePacket (p. 123)

Reimplemented from ArDeviceConnection (p. 122).

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

- ArSerialConnection.h
- ArSerialConnection_LIN.cpp
- ArSerialConnection_WIN.cpp
4.95 ArSick Class Reference

The sick driver.

```cpp
#include <ArSick.h>
```

Inheritance diagram for ArSick::

```
ArThread

ArRangeDevice
```

```
ArASyncTask
```

```
ArRangeDeviceThreaded
```

```
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=1448, const char *name="laser")`
  * Constructor.

- `~ArSick (void)`
  * 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, BaudRate 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.*

- **void** `setSensorPosition` (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 asynchronously.*

- **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.*

- **bool** `runOnRobot` (void)
  
  *Runs the laser off of the robot, advised.*
• 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, ArListPos::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, ArListPos::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.

• void setFilterNearDist (double dist)
  
  Sets the distance which two readings must be less than to be ignored.
• double **getFilterNearDist** (void)
  
  *Gets the distance which two readings must be less than to be ignored.*

• bool **isUsingSim** (void)
  
  *Gets whether the laser is simulated or not.*

• bool **isControllingPower** (void)
  
  *Gets whether the computer is controlling 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. 264).*

• void **processPacket** (ArSickPacket ∗packet, ArPose pose, 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.*

• 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.*

• void **switchState** (State state)
  *Internal function for switching states.*

4.95.1 Detailed Description

The sick driver.

4.95.2 Member Enumeration Documentation

4.95.2.1 enum **ArSick::BaudRate**

**Enumeration values:**

  - **BAUD9600** 9600 Baud.
4.95.2.2  enum ArSick::Degrees

Enumeration values:

- **DEGREES180**  180 Degrees.
- **DEGREES100**  100 Degrees.

4.95.2.3  enum ArSick::Increment

Enumeration values:

- **INCREMENT_ONE**  One degree increments.
- **INCREMENT_HALF**  Half a degree increments.

4.95.2.4  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.
4.95.3 Member Function Documentation

4.95.3.1 void ArSick::addConnectCB (ArFunctor * functor, ArListPos::Pos position)

Adds a connect callback.

Adds a connect callback, which is an ArFunctor (p. 133), created as an ArFunctorC (p. 150). 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:
  functor functor created from ArFunctorC (p. 150) which refers to the function to call.
  position whether to place the functor first or last

See also:
  remConnectCB (p. 374)

4.95.3.2 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. 133), created as an ArFunctorC (p. 150). This whole list of disconnect normally callbacks is called when something calls disconnect if the instance isConnected. 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. 150) which refers to the function to call.
  position whether to place the functor first or last

See also:
  remFailedConnectCB (p. 375)

4.95.3.3 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 \texttt{ArFunctor} (p. 133), created as an \texttt{ArFunctorC} (p. 150). 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 (i.e., 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 \texttt{setConnectionTimeoutTime}. If you have some sort of module that adds a callback, that module must remove the callback when the module removed.

**Parameters:**

- \texttt{functor} functor created from \texttt{ArFunctorC} (p. 150) which refers to the function to call.
- \texttt{position} whether to place the functor first or last

**See also:**

- \texttt{remFailedConnectCB} (p. 375)

### 4.95.3.4 \texttt{void ArSick::addFailedConnectCB (ArFunctor \ast functor, ArListPos::Pos position)}

Adds a callback for when a connection to the robot is failed.

Adds a failed connect callback, which is an \texttt{ArFunctor} (p. 133), created as an \texttt{ArFunctorC} (p. 150). 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:**

- \texttt{functor} functor created from \texttt{ArFunctorC} (p. 150) which refers to the function to call.
- \texttt{position} whether to place the functor first or last

**See also:**

- \texttt{remFailedConnectCB} (p. 375)

### 4.95.3.5 \texttt{bool ArSick::asyncConnect (void)}

Connect to the laser asynchronously.

This does not \texttt{lockDevice} the laser, but you should \texttt{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. 372), ArRangeDeviceThreaded::run (p. 265), Ar-
RangeDeviceThreaded::runAsync (p. 265), runOnRobot (p. 375)

4.95.3.6  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

4.95.3.7  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 lockDevice the laser or not have the laser being poked at by multiple
threads before you use htis function call

4.95.3.8  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
4.95.3.9  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

4.95.3.10  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

4.95.3.11  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.

4.95.3.12  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
4.95.3.13 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

4.95.3.14 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. 370)

4.95.3.15 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. 370)

4.95.3.16 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. 370)
4.95.3.17 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. 371)

4.95.3.18 bool ArSick::runOnRobot (void)

Runs the laser off of the robot, advised.

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.

4.95.3.19 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...

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

- ArSick.h
- ArSick.cpp
4.96  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)
  
  Constructor.

- virtual ~ArSickLogger (void)
  
  Destructor.

- void setDistDiff (double distDiff)
  
  Sets the distance at which the robot will take a new reading.

- double 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 robotTask (void)
  
  The task which gets attached to the robot.

4.96.1 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.
4.96.2 Constructor & Destructor Documentation

4.96.2.1 ArSickLogger::ArSickLogger (ArRobot * robot, ArSick * sick, double distDiff, double degDiff, const char * filename)

Constructor.

Make sure you have called ArSick::configure (p. 372) 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

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

- ArSickLogger.h
- ArSickLogger.cpp
4.97 ArSickPacket Class Reference

Represents the packets sent to the sick as well as those received from it.

```c
#include <ArSickPacket.h>
```

Inheritance diagram for ArSickPacket::

```
ArBasePacket
  
ArSickPacket
```

Public Methods

- **ArSickPacket** (unsigned char sendingAddress=0)
  
  Constructor.

- virtual **~ArSickPacket** (void)
  
  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 **finalize** (void)
  
  Finalizes 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.

4.97.1 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. 107) except for these functions here, setAddress, getAddress, verifyCheckSum, print, getID, and calcCheckSum.

4.97.2 Member Function Documentation

4.97.2.1 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

4.97.2.2 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 getSendingAddress instead.
4.97.2.3 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 finalized for sending, the address is put into the
appropriate spot in the packet.

Returns:
the address of the laser to be addressed

4.97.2.4 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. 111).

4.97.2.5 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 finalized 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
Given a device connection it receives packets from the sick through it.

```cpp
#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 useBase0Address=false)
  
  *Constructor with assignment of a device connection.*

- virtual **~ArSickPacketReceiver** (void)
  
  *Destructor.*

- **ArSickPacket * receivePacket** (unsigned int msWait=0)
  
  *Receives a packet from the robot if there is one available.*

- void **setDeviceConnection** (ArDeviceConnection *deviceConnection)
  
  *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.*

### Detailed Description

Given a device connection it receives packets from the sick through it.
4.98.2 Constructor & Destructor Documentation

4.98.2.1 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

4.98.2.2 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

4.98.3 Member Function Documentation

4.98.3.1 ArSickPacket * ArSickPacketReceiver::receivePacket (unsigned int msWait = 0)

Receives a packet from the robot if there is one available.

Parameters:

msWait 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:

- ArSickPacketReceiver.h
- ArSickPacketReceiver.cpp
4.99 ArSignalHandler Class Reference

Signal handling class.

```cpp
#include <ArSignalHandler.h>
```

Inheritance diagram for ArSignalHandler::

```
ArThread
  ArASyncTask
    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)

  Don't 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.

4.99.1 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 incurs 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. 181) 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 really useful. 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.

### 4.99.2 Member Function Documentation

#### 4.99.2.1 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

#### 4.99.2.2 void ArSignalHandler::block (Signal sig) [static]

Block the given signal.

Block the given signal. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

**Parameters:**

- **sig** the number of the signal
4.99.2.3 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.

4.99.2.4 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 receive the common signals which are SIGHUP, SIGINT, SIGQUIT, and SIGTERM. This function can be called at any time.

4.99.2.5 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 finishes, it calls the signal handler. This uses the system call signal(2). This should not be used if you have a threaded program.

See also:
    createHandlerThreaded (p. 387)

4.99.2.6 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 useful 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 ArThread (p. 417) 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. 387)
4.99.2.7  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.

4.99.2.8  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

4.99.2.9  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 received. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

Parameters:
  sig  the number of the signal

4.99.2.10 void * ArSignalHandler::runThread (void * arg)  [virtual]

The main run loop.

Override this function and put your taskes run loop here. Check the value of getRunning() (p. 418) or myRunning periodically in your loop. If the value goes false, the loop should exit and runThread() (p. 388) should return.

Reimplemented from ArASyncTask (p. 106).
4.99.2.11 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

4.99.2.12 void ArSignalHandler::unblockAll () [static]

Unblock all the signals.

Unblock all the signals. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

4.99.2.13 void ArSignalHandler::unhandle (Signal sig) [static]

Do not handle the given signal.

Do not handle the given signal. Call this before calling createHandlerNonThreaded or createHandlerThreaded.

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
4.100  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.

- **ArSocket (int port, bool doClose, Type type)**
  Constructor which opens a server port.

- **~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.

• const std::string & getErrorStr () const
  
  Get the last error string.

• Error getError () const
  
  Get the last error.

**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.
4.100 ArSocket Class Reference

- const size_t sockAddrLen ()
  Size of the sockaddr.

- const size_t maxHostNameLen ()
  Max host name length.

- int hostToNetOrder (int i)
  Host byte order to network byte order.

- int netToHostOrder (int i)
  Network byte order to host byte order.

4.100.1 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. 394) and call ArSocket::shutdown() (p. 395) when it exits. For programs that use Aria::init() (p. 184) and Aria::uninit() (p. 185) calling the ArSocket::init() (p. 394) and ArSocket::shutdown() (p. 395) is unnecessary. The Aria (p. 181) initialization functions take care of this. These functions do nothing in Linux.

4.100.2 Constructor & Destructor Documentation

4.100.2.1 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

4.100.2.2  

**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** automatically close the port if the socket is destructed
- **type** protocol type to use

4.100.3  

**Member Function Documentation**

4.100.3.1  

**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 descriptor when it is destructed.

4.100.3.2  

**bool ArSocket::init (void) [static]**

Initialize the network layer.
In Windows, the networking subsystem needs to be initialized and shut-
down individually by each program. So when a program starts they will
need to call the static function **ArSocket::init()** (p.394) and call **Ar-
Socket::shutdown()** (p.395) when it exits. For programs that use **Aria::init()**
(p.184) and **Aria::uninit()** (p.185) calling the **ArSocket::init()** (p.394) and
**ArSocket::shutdown()** (p.395) is unnecessary. The **Aria** (p.181) initialization
functions take care of this. These functions do nothing in Linux.

4.100.3.3  

**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
4.100.3.4 void ArSocket::shutdown () [static]

Shutdown the network layer.

In Windows, the networking subsystem needs to be initialized and shutdown individualy by each program. So when a program starts they will need to call the static function `ArSocket::init()` (p. 394) and call `ArSocket::shutdown()` (p. 395) when it exits. For programs that use `Aria::init()` (p. 184) and `Aria::uninit()` (p. 185) calling the `ArSocket::init()` (p. 394) and `ArSocket::shutdown()` (p. 395) is unnecessary. The `Aria` (p. 181) initialization functions take care of this. These functions do nothing in Linux.

4.100.3.5 void ArSocket::transfer (ArSocket * s) [inline]

Transfer ownership of a socket.

`transfer()` (p. 395) will transfer ownership to this socket. The input socket will no longer close the file descriptor when it is destructed.

4.100.3.6 int ArSocket::write (const void * buff, size_t len) [inline]

Write to the socket.

**Parameters:**
- `buff` buffer to write from
- `len` how many bytes to write

**Returns:**
- number of bytes written

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

- ArSocket.h
- ArSocket_LIN.cpp
- ArSocket_WIN.cpp
4.101  ArSonarDevice Class Reference

A class for keeping track of sonar.

#include <ArSonarDevice.h>

Inheritance diagram for ArSonarDevice::

```plaintext
ArRangeDevice

<table>
<thead>
<tr>
<th>ArSonarDevice</th>
</tr>
</thead>
</table>
```

Public Methods

- **ArSonarDevice** (size_t currentBufferSize=24, size_t cumulativeBufferSize=64, const char *name="sonar")
  
  Constructor.

- ~**ArSonarDevice** (void)
  
  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. 256) default action.

- void **setCumulativeMaxRange** (double r)
  
  Maximum range for a reading to be added to the cumulative buffer (mm).

4.101.1  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.
4.101.2 Member Function Documentation

4.101.2.1 void ArSonarDevice::addReading (double x, double y) [virtual]

Adds sonar readings to the current and cumulative buffers Overrides the ArRangeDevice (p. 256) 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. 261) / unlockDevice() (p. 262) if calling this from outside process().

Parameters:
- x the global x coordinate of the reading
- y the global y coordinate of the reading

Reimplemented from ArRangeDevice (p. 256).

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

- ArSonarDevice.h
- ArSonarDevice.cpp
4.102 ArSonyPacket Class Reference

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

#include <ArSonyPTZ.h>

Inheritance diagram for ArSonyPacket::

```
ArBasePacket
  ↓
ArSonyPacket
```

Public Methods

- **ArSonyPacket** (ArTypes::UByte2 bufferSize=15)
  Constructor.

- virtual void **uByteToBuf** (ArTypes::UByte val)
  Puts ArTypes::UByte (p. 426) into packets buffer.

- virtual void **byte2ToBuf** (ArTypes::Byte2 val)
  Puts ArTypes::Byte2 (p. 426) 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.

4.102.1 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 these, if you use anything else your commands won't work. You must use uByteToBuf and byte2ToBuf.

4.102.2 Member Function Documentation

4.102.2.1 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, basically 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

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

- ArSonyPTZ.h
- ArSonyPTZ.cpp
4.103 ArSonyPTZ Class Reference

A class to use the sony pan tilt zoom unit.

#include <ArSonyPTZ.h>

Inheritance diagram for ArSonyPTZ::

```
+----------------+     +-------------------+
| ArPTZ           |     | 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)
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)
  The angle the camera was last told to pan to.

- virtual int getTilt (void)
  The angle the camera was last told to tilt to.

- virtual int getZoom (void)
  The value the camera was last told to zoom to.

- virtual int getMaxPosPan (void)
  Gets the highest positive degree the camera can pan to.

- virtual int getMaxNegPan (void)
  Gets the lowest negative degree the camera can pan to.

- virtual int getMaxPosTilt (void)
  Gets the highest positive degree the camera can tilt to.

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

- virtual int getMaxZoom (void)
  Gets the maximum value for the zoom on this camera.

- virtual int getMinZoom (void)
  Gets the lowest value for the zoom on this camera.

### 4.103.1 Detailed Description

A class to use the sony pan tilt zoom unit.
4.103.2 Member Enumeration Documentation

4.103.2.1 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.

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

- ArSonyPTZ.h
- ArSonyPTZ.cpp
4.104 ArSyncTask Class Reference

Class used internally to manage the functions that are called every cycle.

```cpp
#include <ArSyncTask.h>
```

Public Methods

- **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.

- virtual ~ArSyncTask (void)
  
  Destructor.

- void run (void)
  
  Runs the node, which runs all children of this node as well.

- void print (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, ArTaskState::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.

### 4.104.1 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. 290) 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 addNewBranch 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. 133) 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 appropriate 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.

4.104.2 Constructor & Destructor Documentation

4.104.2.1 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.

4.104.2.2 ArSyncTask::~ArSyncTask (void) [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.

4.104.3 Member Function Documentation

4.104.3.1 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 internal 
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.
4.104.3.2 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 internal 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. 133) which contains the functor to invoke when run is called.

4.104.3.3 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.

4.104.3.4 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.
4.104.3.5 `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.

4.104.3.6 `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.

4.104.3.7 `void ArSyncTask::print (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.

4.104.3.8 `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.

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

- ArSyncTask.h
- ArSyncTask.cpp
4.105 ArTaskPool Class Reference

A thread pool for running functions or ArPeriodicTasks.

#include <ArTaskPool.h>

Public Methods

- ArTaskPool ()
  Constructor.

- virtual ~ArTaskPool ()
  Destructor.

- virtual bool runTask (ArFunctor *task)
  Run a function as a periodic task.

- virtual bool runTask (ArPeriodicTask *task)
  Run a periodic task.

- virtual void init (int minLimit=1, int maxLimit=0)
  Init the pool.

- virtual void uninit ()
  Uninit the pool. Blocks till all tasks exit.

- virtual void forceUninit ()
  Force all tasks to exit, then uninit the pool.

Static Public Methods

- ArTaskPool * getPool ()
  Get a pointer to the 'single' instance of ArTaskPool.

- void setPool (ArTaskPool *pool)
  Set the pointer to the 'single' instance of ArTaskPool.

Static Public Attributes

- ArTaskPool * ourPool = 0
4.105.1 Detailed Description

A thread pool for running functions or ArPeriodicTasks.

4.105.2 Member Data Documentation

4.105.2.1 ArTaskPool * ArTaskPool::ourPool = 0 [static]

When tasks need to be run, threads will be created based upon the need. See setTaskMaxLimit() about this. If minLimit is greater than 0 and there are more threads than minLimit, the extra threads will be destroyed. If minLimit is 0, then all threads will be immediately destroyed after the task is done.

Parameters:

minLimit minimum number of threads that exist at once

See also:

setTaskMaxLimit

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

- ArTaskPool.h
- ArTaskPool.cpp
4.106  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 }

4.106.1 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).

4.106.2 Member Enumeration Documentation

4.106.2.1 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
4.107 ArTcpConnection Class Reference

For connectiong to a device through a socket.

```c
#include <ArTcpConnection.h>
```

Inheritance diagram for ArTcpConnection:

```
ArDeviceConnection ----↓
ArTcpConnection
```

Public Types

- `enum Open { OPEN_NET_FAIL = 1, OPEN_BAD_HOST, OPEN_NO_ROUTE, OPEN_CON_REFUSED }`

Public Methods

- `ArTcpConnection (void)`
  
  Constructor.

- `virtual ~ArTcpConnection (void)`
  
  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 msWait=0)`
  
  Reads data from connection.
• virtual int write (const char *data, unsigned int size)  
  Writes data to connection.

• virtual std::string 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.

• void setStatus (int status)  
  Sets the status of the device, ONLY use this if you’re playing with setSocket  
  and know what you’re doing.

### 4.107.1 Detailed Description

For connectiong to a device through a socket.

### 4.107.2 Member Enumeration Documentation

#### 4.107.2.1 enum ArTcpConnection::Open

Enumeration values:

- OPEN_NET_FAIL  Some critical part of the network isn’t working.
4.107 ArTcpConnection Class Reference

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 connection.

4.107.3 Member Function Documentation

4.107.3.1 bool ArTcpConnection::close (void) [virtual]

Closes the connection.

Returns:
whether the close succeeded or not

Reimplemented from ArDeviceConnection (p. 120).

4.107.3.2 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. 414)

4.107.3.3 std::string 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. 120).
4.107.3.4 int ArTcpConnection::getPort (void)

Gets the number of the port connected to.

Returns:
the number of the port connected to

See also:
getHost (p. 413)

4.107.3.5 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. 121)

Reimplemented from ArDeviceConnection (p. 120).

4.107.3.6 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. 121).

4.107.3.7 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. 121).
4.107.3.8 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. 413)

4.107.3.9 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. 416), writePacket (p. 123)

Reimplemented from ArDeviceConnection (p. 122).

4.107.3.10 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
4.107.3.11  int ArTcpConnection::write (const char * data, unsigned int size) [virtual]

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. 415), writePacket (p. 123)

Reimplemented from ArDeviceConnection (p. 122).

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

- ArTcpConnection.h
- ArTcpConnection.cpp
### 4.108 ArThread Class Reference

POSIX/WIN32 thread wrapper class.

```cpp
#include <ArThread.h>
```

Inheritance diagram for ArThread:

![Inheritance Diagram](inclusion.png)

#### Public Types

- `enum Status { STATUS_FAILED = 1, STATUS_NORESOURCE, STATUS_NO_SUCH_THREAD, STATUS_INVALID, STATUS_ALREADY_DETACHED }`

#### Public Methods

- **ArThread** (bool blockAllSignals=true)
  
  Constructor.

- **ArThread** (ThreadType thread, bool joinable, bool blockAllSignals=true)
  
  Constructor - starts the thread.

- **ArThread** (ArFunctor *func, bool joinable=true, bool blockAllSignals=true)
  
  Constructor - starts the thread.

- virtual ~ArThread (void)
  
  Destructor.

- virtual int create (ArFunctor *func, bool joinable=true, bool lowerPriority=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)
  *Detach the thread so it can't be joined.*

- **virtual void cancel**(void)
  *Cancel 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 bool getJoinable**(void)
  *Get the joinable status of the thread.*

- **virtual ThreadType *getThread**(void)
  *Get the underlying thread type.*

- **virtual ArFunctor *getFunc**(void)
  *Get the functor that the thread runs.*

- **virtual void setRunning**(bool yesno)
  *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.*

- **std::string getError**(int err)
  *Translate error into string.*

- **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 yield (void)
  
  *Yield the processor to another thread.*

Protected Attributes

- bool myRunning
  
  *State variable to denote when the thread should continue or exit.*

4.108.1 Detailed Description

POSIX/WIN32 thread wrapper class.

create() (p. 417) 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. 418)’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. 418) 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. 418) will cancel the thread.

The static function self() (p. 420) will return a thread.
4.108.2 Member Enumeration Documentation

4.108.2.1 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_DETACHED**  Thread is already detached.

4.108.3 Member Function Documentation

4.108.3.1 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. 184), 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 other words, it should be called by **main()**.

4.108.3.2 ArThread * ArThread::self (void) [static]

Returns the instance of your own thread.

If a newly created thread calls **self()** (p. 420) 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. 417) then updates the list of threads with the new thread ID. There is just not much that can be done about that. The user should be aware of this caveat.

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

- ArThread.h
- ArThread.cpp
- ArThread_LIN.cpp
- ArThread_WIN.cpp
4.109 ArTime Class Reference

A class for time readings.
#include <ariaUtil.h>

Public Methods

• **ArTime** (void)
  Constructor.

• **~ArTime** (void)
  Destructor.

• **long mSecSince** (ArTime since)
  Gets the number of milliseconds since the given timestamp to this one.

• **long secSince** (ArTime since)
  Gets the number of seconds since the given timestamp to this one.

• **long mSecTo** (void)
  Finds the number of millisecs from when this timestamp is set to to now.

• **long secTo** (void)
  Finds the number of seconds from when this timestamp is set to to now.

• **long mSecSince** (void)
  Finds the number of milliseconds from this timestamp to now.

• **long secSince** (void)
  Finds the number of seconds from when this timestamp was set to now.

• **bool isBefore** (ArTime testTime)
  returns whether the given time is before this one or not.

• **bool isAt** (ArTime testTime)
  returns whether the given time is equal to this time or not.

• **bool isAfter** (ArTime testTime)
  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)

  Gets the seconds since 1970.

• time_t getMSec (void)

  Gets the milliseconds.

• void print (void)

  Logs the time.

### 4.109.1 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
4.110 ArTransform Class Reference

A class to handle transforms between different coordinates.

#include <ArTransform.h>

Public Methods

- **ArTransform** (void)
  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** (void)
  Destructor.

- **ArPose doTransform** (ArPose 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.

- void **doTransform** (std::list< ArPose * > *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).
4.110.1 Detailed Description

A class to handle transforms between different coordinates.

4.110.2 Member Function Documentation

4.110.2.1 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:

source the parameter to transform

Returns:

the source transformed from absolute into local coords

4.110.2.2 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

4.110.2.3 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
4.110.2.4 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

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

- ArTransform.h
- ArTransform.cpp
4.111  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.*

4.111.1  Detailed Description

Contains platform independent sized variable types.

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

- ariaTypedefs.h
4.112 ArUtil Class Reference

This class has utility functions.

```c
#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_CONFIG,
  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.

4.112.1 Detailed Description

This class has utility functions.
4.112.2  Member Enumeration Documentation

4.112.2.1  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.

4.112.2.2  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_MACHINE.
  REGKEY_USERS  use HKEY_USERS.
4.112.3 Member Function Documentation

4.112.3.1 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

4.112.3.2 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.

4.112.3.3 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.

4.112.3.4 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.102)... make sure you have enough space in the arrays that you're passing as dest... this allocates no memory

4.112.3.5 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

4.112.3.6 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

4.112.3.7 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

4.112.3.8 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

4.112.3.9 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.181) 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
4.112.3.10 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

4.112.3.11 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

4.112.3.12 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
4.112.3.13  void ArUtil::sleep (unsigned int \textit{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:
\begin{itemize}
  \item \textit{ms} the number of milliseconds to sleep for
\end{itemize}

4.112.3.14  void ArUtil::splitString (std::string \textit{inString}, std::list<
\textit{std::string}> \& \textit{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:
\begin{itemize}
  \item \textit{inString} the input string to split
  \item \textit{outList} the list in which to store the words that are found
\end{itemize}

4.112.3.15  int ArUtil::strcmp (const char * \textit{str}, const char * \textit{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 \textit{str} is found, respectively, to be less than, to match, or be greater than \textit{str2}.

Parameters:
\begin{itemize}
  \item \textit{str} the string to compare
  \item \textit{str2} the second string to compare
\end{itemize}

Returns:
\begin{itemize}
  \item an integer less than, equal to, or greater than zero if \textit{str} is found, respectively, to be less than, to match, or be greater than \textit{str2}.
\end{itemize}
4.112.3.16  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, respec-
tively, to be less than, to match, or be greater than str2.

4.112.3.17  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, respec-
tively, to be less than, to match, or be greater than str2.

4.112.3.18  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.

4.112.3.19 bool ArUtil::stripDir (std::string fileIn, std::string & fileOut) [static]

OS-independent way of stripping the directory from the filename.
Works for \ 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

4.112.3.20 bool ArUtil::stripFile (std::string fileIn, std::string & fileOut) [static]

OS-independent way of stripping the filename from the directory.
Works for \ 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:

• ariaUtil.h
• ariaUtil.cpp
4.113 ArVCC4 Class Reference

Driver for the VCC4.

```
#include <ArVCC4.h>
```

Inheritance diagram for ArVCC4:

```
ArPTZ
  
ArVCC4
```

Public Methods

- **ArVCC4** (ArRobot *robot, bool myInverted=false)  
  - Turn on/off the camera.

- bool **power** (int status)
  - Initializes the camera.

- bool **init** (void)
  - Initializes the camera.

- void **connectHandler** (void)
  - Internal, attached to robot, inits the camera when robot connects.

- 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.

- int **getMaxPosPan** (void)
  - Gets the highest positive degree the camera can pan to.
• int getMaxNegPan (void)
  Gets the lowest negative degree the camera can pan to.

• int getMaxPosTilt (void)
  Gets the highest positive degree the camera can tilt to.

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

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

• bool panTilt (int pdeg, int tdeg)
  Pans and tilts to the given degrees.

• bool zoom (int deg)
  Zooms to the given value.

• 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)
  Sends this packet to the unit.

• int getPan (void)
  The angle the camera was last told to pan to.

• int getTilt (void)
  The angle the camera was last told to tilt to.

• int getZoom (void)
  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.

• int getMaxZoom (void)
  
  Gets the maximum value for the zoom on this camera.

• int getMinZoom (void)
  
  Gets the lowest value for the zoom on this camera.

4.113.1 Detailed Description

Driver for the VCC4.

4.113.2 Constructor & Destructor Documentation

4.113.2.1 ArVCC4::ArVCC4 (ArRobot * robot, bool inverted = false)

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

The documentation for this class was generated from the following files:
- ArVCC4.h
- ArVCC4.cpp
4.114 ArVCC4Commands Class Reference

A class with the commands for the VCC4.

#include <ArVCC4.h>

Public Types

- enum { DELIM = 0x00, DEVICEID = 0x30, PANSLEW = 0x50, TILTSLEW = 0x51, STOP = 0x53, INIT = 0x58, PANTILT = 0x62, SETRANGE = 0x64, CONTROL = 0x90, POWER = 0xA0, ZOOMSTOP = 0xA2, ZOOM = 0xA3, FOOTER = 0xEF, HEADER = 0xFF }

4.114.1 Detailed Description

A class with the commands for the VCC4.

This class is for controlling the Canon VC-C4 camera.

Note that currently the pan/tilt commands send a stop to the camera, halting the previous command. The stop command waits for 200ms for processing. The same is true for the zoom commands.

Also note that the camera takes about 4 seconds to complete the power-on cycle, and 2 seconds for an init. Commands sent during these periods will be ignored, and may confuse the camera.

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.

4.114.2 Member Enumeration Documentation

4.114.2.1 anonymous enum

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.
PANTILT  Pan/tilt command.
SETRANGE  Pan/tilt min/max range assignment.
CONTROL  Puts camera in Control mode.
POWER  Turns on/off power.
ZOOMSTOP  Stops zoom motion.
ZOOM  Zooms camera lens.
FOOTER  Packet Footer.
HEADER  Packet Header.

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

- ArVCC4.h
4.115 ArVCC4Packet Class Reference

A class for for making commands to send to the VCC4.

#include <ArVCC4.h>

Inheritance diagram for ArVCC4Packet::

```
ArBasePacket
   
ArVCC4Packet
```

Public Methods

- ArVCC4Packet (ArTypes::UByte2 bufferSize=30)  
  Constructor.

- virtual ~ArVCC4Packet (void)  
  Destructor.

- virtual void finalize (void)  
  Finalizes the packet in preparation for sending, must be done.

4.115.1 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 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:

- ArVCC4.h
- ArVCC4.cpp
Index

~ArACTSBlob
ArACTSBlob, 90
~ArACTS_1_2
ArACTS_1_2, 85
~ArAMPTU
ArAMPTU, 92
~ArAMPTUPacket
ArAMPTUPacket, 97
~ArAsyncTask
ArAsyncTask, 105
~ArAction
ArAction, 38
~ArActionAvoidFront
ArActionAvoidFront, 41
~ArActionAvoidSide
ArActionAvoidSide, 43
~ArActionBumpers
ArActionBumpers, 45
~ArActionConstantVelocity
ArActionConstantVelocity, 47
~ArActionDesired
ArActionDesired, 49
~ArActionGroup
ArActionGroup, 58
~ArActionInput
ArActionInput, 65
~ArActionJoydrive
ArActionJoydrive, 67
~ArActionKeydrive
ArActionKeydrive, 70
~ArActionLimiterBackwards
ArActionLimiterBackwards, 73
~ArActionLimiterForwards
ArActionLimiterForwards, 75
~ArActionLimiterTableSensor
ArActionLimiterTableSensor, 77
~ArActionStallRecover
ArActionStallRecover, 79
~ArActionStop
ArActionStop, 81
~ArActionTurn
ArActionTurn, 83
~ArArg
ArArg, 99
~ArArgumentBuilder
ArArgumentBuilder, 102
~ArArgumentParser
ArArgumentParser, 103
~ArBasePacket
ArBasePacket, 107
~ArCondition
ArCondition, 116
~ArDPPTU
ArDPPTU, 124
~ArDPPTUPacket
ArDPPTUPacket, 132
~ArDeviceConnection
ArDeviceConnection, 118
~ArFunctor
ArFunctor, 133
~ArFunctor1
ArFunctor1, 135
~ArFunctor1C
ArFunctor1C, 137
~ArFunctor2
ArFunctor2, 141
~ArFunctor2C
ArFunctor2C, 144
~ArFunctorC
ArFunctorC, 150
~ArGlobalFunctor
  ArGlobalFunctor, 153
~ArGlobalFunctor1
  ArGlobalFunctor1, 155
~ArGlobalFunctor2
  ArGlobalFunctor2, 158
~ArGlobalRetFunctor
  ArGlobalRetFunctor, 163
~ArGlobalRetFunctor1
  ArGlobalRetFunctor1, 165
~ArGlobalRetFunctor2
  ArGlobalRetFunctor2, 168
~ArGripper
  ArGripper, 172
~ArInterpolation
  ArInterpolation, 186
~ArIrrfDevice
  ArIrrfDevice, 188
~ArJoyHandler
  ArJoyHandler, 190
~ArKeyHandler
  ArKeyHandler, 195
~ArMode
  ArMode, 207
~ArModeCamera
  ArModeCamera, 211
~ArModeGripper
  ArModeGripper, 213
~ArModeSonar
  ArModeSonar, 215
~ArModeTeleop
  ArModeTeleop, 217
~ArModeWander
  ArModeWander, 219
~ArModule
  ArModule, 221
~ArMutex
  ArMutex, 227
~ArPTZ
  ArPTZ, 244
~ArPeriodicTask
  ArPeriodicTask, 229
~ArPose
  ArPose, 231
~ArPref
  ArPref, 236
~ArPriorityResolver
  ArPriorityResolver, 243
~ArRangeBuffer
  ArRangeBuffer, 250
~ArRangeDevice
  ArRangeDevice, 256
~ArRangeDeviceThreaded
  ArRangeDeviceThreaded, 264
~ArRecurrentTask
  ArRecurrentTask, 267
~ArResolver
  ArResolver, 269
~ArRetFunctor
  ArRetFunctor, 271
~ArRetFunctor1
  ArRetFunctor1, 273
~ArRetFunctor1C
  ArRetFunctor1C, 275
~ArRetFunctor2
  ArRetFunctor2, 279
~ArRetFunctor2C
  ArRetFunctor2C, 282
~ArRetFunctorC
  ArRetFunctorC, 287
~ArRobot
  ArRobot, 290
~ArRobotPacket
  ArRobotPacket, 340
~ArRobotPacketReceiver
  ArRobotPacketReceiver, 342
~ArRobotPacketSender
  ArRobotPacketSender, 345
~ArRobotParams
  ArRobotParams, 349
~ArSectors
  ArSectors, 352
~ArSerialConnection
  ArSerialConnection, 358
~ArSick
  ArSick, 364
~ArSickLogger
  ArSickLogger, 376
~ArSickPacket
  ArSickPacket, 378
~ArSickPacketReceiver
  ArSickPacketReceiver, 381
## INDEX

<table>
<thead>
<tr>
<th>Class/Method</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArSignalHandler</td>
<td>384</td>
</tr>
<tr>
<td>ArSocket</td>
<td>390</td>
</tr>
<tr>
<td>ArSonarDevice</td>
<td>396</td>
</tr>
<tr>
<td>ArSyncTask</td>
<td>405</td>
</tr>
<tr>
<td>ArTaskPool</td>
<td>408</td>
</tr>
<tr>
<td>ArTcpConnection</td>
<td>411</td>
</tr>
<tr>
<td>ArThread</td>
<td>417</td>
</tr>
<tr>
<td>ArTime</td>
<td>421</td>
</tr>
<tr>
<td>ArTransform</td>
<td>423</td>
</tr>
<tr>
<td>ArVCC4Packet</td>
<td>442</td>
</tr>
<tr>
<td>ABSPAN</td>
<td>ArAMPTUCommands, 95</td>
</tr>
<tr>
<td>ABSTILT</td>
<td>ArAMPTUCommands, 95</td>
</tr>
<tr>
<td>ACCEL</td>
<td>ArDPPTUCommands, 130</td>
</tr>
<tr>
<td>accept</td>
<td>ArSocket, 391</td>
</tr>
<tr>
<td>accountForRobotHeading</td>
<td>ArActionDesired, 52</td>
</tr>
<tr>
<td>actionHandler</td>
<td>ArRobot, 306</td>
</tr>
<tr>
<td>ActionMap</td>
<td>ArResolver, 269</td>
</tr>
<tr>
<td>activate</td>
<td>ArAction, 38</td>
</tr>
<tr>
<td>ArActionGroup</td>
<td>58</td>
</tr>
<tr>
<td>ArActionInput</td>
<td>65</td>
</tr>
<tr>
<td>ArActionKeydrive</td>
<td>70</td>
</tr>
<tr>
<td>ArMode</td>
<td>208</td>
</tr>
<tr>
<td>ArModeCamera</td>
<td>211</td>
</tr>
<tr>
<td>ArModeGripper</td>
<td>213</td>
</tr>
<tr>
<td>ArModeSonar</td>
<td>215</td>
</tr>
<tr>
<td>ArModeTeleop</td>
<td>217</td>
</tr>
<tr>
<td>ArModeWander</td>
<td>219</td>
</tr>
<tr>
<td>activateExclusive</td>
<td>ArActionGroup, 58</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>ArTaskState, 410</td>
</tr>
<tr>
<td>ActsConstants</td>
<td>ArACTS, 86</td>
</tr>
<tr>
<td>actsHandler</td>
<td>ArACTS, 86</td>
</tr>
<tr>
<td>add</td>
<td>ArArgumentBuilder, 102</td>
</tr>
<tr>
<td>addAction</td>
<td>ArActionGroup, 59</td>
</tr>
<tr>
<td>ArRobot</td>
<td>306</td>
</tr>
<tr>
<td>addAngle</td>
<td>ArMath, 203</td>
</tr>
<tr>
<td>addAverage</td>
<td>ArActionDesired, 52</td>
</tr>
<tr>
<td>addConnectCB</td>
<td>ArRobot, 306</td>
</tr>
<tr>
<td>ArSick, 370</td>
<td></td>
</tr>
<tr>
<td>addDisconnectNormallyCB</td>
<td>ArRobot, 306</td>
</tr>
<tr>
<td>ArSick, 370</td>
<td></td>
</tr>
<tr>
<td>addFailedConnectCB</td>
<td>ArRobot, 307</td>
</tr>
<tr>
<td>ArSick, 371</td>
<td></td>
</tr>
<tr>
<td>addHandlerCB</td>
<td>ArSignalHandler, 386</td>
</tr>
<tr>
<td>addInitCallBack</td>
<td>Aria, 182</td>
</tr>
<tr>
<td>ArKeyHandler</td>
<td>197</td>
</tr>
<tr>
<td>addMSec</td>
<td>ArTime, 422</td>
</tr>
<tr>
<td>addNewBranch</td>
<td>ArSyncTask, 405</td>
</tr>
<tr>
<td>addNewLeaf</td>
<td>ArSyncTask, 405</td>
</tr>
<tr>
<td>addPacketHandler</td>
<td>ArRobot, 308</td>
</tr>
<tr>
<td>addRangeDevice</td>
<td>ArRobot, 297</td>
</tr>
<tr>
<td>addReading</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

ArInterpolation, 186
ArRangeBuffer, 251
ArRangeDevice, 256
ArSonarDevice, 397
addrHost
   ArSocket, 392
addRobot
   Aria, 181
addRunExitCB
   ArRobot, 308
addSensorInterpTask
   ArRobot, 309
addUninitCallBack
   Aria, 182
addUserTask
   ArRobot, 309
ADSEL
   ArCommands, 114
angleBetween
   ArMath, 202
appendSlash
   ArUtil, 430
applyTransform
   ArRangeBuffer, 252
   ArRangeDevice, 259
   ArRobot, 310
   ArSensorReading, 355
ArAction
   ~ArAction, 38
   activate, 38
   ArAction, 38
   deactivate, 39
   getArg, 39
   getDescription, 39
   getDesired, 39
   getName, 39
   getNumArgs, 39
   print, 39
   setNextArgument, 39
   setRobot, 39
ArAction, 37
   fire, 40
ArActionAvoidFront
   ~ArActionAvoidFront, 41
   ArActionAvoidFront, 42
getDesired, 41
ArActionAvoidFront, 41
ArActionAvoidFront, 42
fire, 42
ArActionAvoidSide
   ~ArActionAvoidSide, 43
   ArActionAvoidSide, 43
   getDesired, 43
ArActionAvoidSide, 43
ArActionAvoidSide, 43
fire, 44
ArActionBumpers
   ~ArActionBumpers, 45
   ArActionBumpers, 46
   getDesired, 45
ArActionBumpers, 45
ArActionBumpers, 45
fire, 46
ArActionConstantVelocity
   ~ArActionConstantVelocity, 47
   ArActionConstantVelocity, 47
   getDesired, 47
ArActionConstantVelocity, 47
ArActionConstantVelocity, 47
fire, 48
ArActionDesired
   ~ArActionDesired, 49
   ArActionDesired, 49
   getDeltaHeading, 50
   getDeltaHeadingDesiredChannel, 51
   getDeltaHeadingStrength, 50
   getHeading, 50
   getHeadingStrength, 50
   getMaxNegVel, 50
   getMaxNegVelDesiredChannel, 51
   getMaxNegVelStrength, 50
   getMaxRotVel, 50
   getMaxRotVelDesiredChannel, 51
   getMaxRotVelStrength, 50
   getMaxVel, 50
   getMaxVelDesiredChannel, 51
   getMaxVelStrength, 50
<table>
<thead>
<tr>
<th>Function/Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getVel</td>
<td>49</td>
</tr>
<tr>
<td>getVelDesiredChannel</td>
<td>51</td>
</tr>
<tr>
<td>getVelStrength</td>
<td>50</td>
</tr>
<tr>
<td>reset</td>
<td>49</td>
</tr>
<tr>
<td>ArActionDesired</td>
<td>49</td>
</tr>
<tr>
<td>accountForRobotHeading</td>
<td>52</td>
</tr>
<tr>
<td>addAverage</td>
<td>52</td>
</tr>
<tr>
<td>endAverage</td>
<td>52</td>
</tr>
<tr>
<td>merge</td>
<td>52</td>
</tr>
<tr>
<td>setDeltaHeading</td>
<td>52</td>
</tr>
<tr>
<td>setHeading</td>
<td>53</td>
</tr>
<tr>
<td>setMaxNegVel</td>
<td>53</td>
</tr>
<tr>
<td>setMaxRotVel</td>
<td>53</td>
</tr>
<tr>
<td>setMaxVel</td>
<td>53</td>
</tr>
<tr>
<td>setVel</td>
<td>54</td>
</tr>
<tr>
<td>startAverage</td>
<td>54</td>
</tr>
<tr>
<td>ArActionDesiredChannel</td>
<td>55</td>
</tr>
<tr>
<td>MAX_STRENGTH</td>
<td>55</td>
</tr>
<tr>
<td>MIN_STRENGTH</td>
<td>55</td>
</tr>
<tr>
<td>NO_STRENGTH</td>
<td>55</td>
</tr>
<tr>
<td>ArActionGoto</td>
<td>56</td>
</tr>
<tr>
<td>cancelGoal</td>
<td>56</td>
</tr>
<tr>
<td>getCloseDist</td>
<td>56</td>
</tr>
<tr>
<td>getDesired</td>
<td>57</td>
</tr>
<tr>
<td>getGoal</td>
<td>56</td>
</tr>
<tr>
<td>getSpeed</td>
<td>56</td>
</tr>
<tr>
<td>haveAchievedGoal</td>
<td>56</td>
</tr>
<tr>
<td>setCloseDist</td>
<td>56</td>
</tr>
<tr>
<td>setGoal</td>
<td>56</td>
</tr>
<tr>
<td>setSpeed</td>
<td>56</td>
</tr>
<tr>
<td>ArActionGoto</td>
<td>56</td>
</tr>
<tr>
<td>fire</td>
<td>57</td>
</tr>
<tr>
<td>ArActionGroup</td>
<td>58</td>
</tr>
<tr>
<td>~ArActionGroup</td>
<td>58</td>
</tr>
<tr>
<td>activate</td>
<td>58</td>
</tr>
<tr>
<td>activateExclusive</td>
<td>58</td>
</tr>
<tr>
<td>ArActionGroup</td>
<td>59</td>
</tr>
<tr>
<td>deactivate</td>
<td>58</td>
</tr>
<tr>
<td>getActionList</td>
<td>58</td>
</tr>
<tr>
<td>removeActions</td>
<td>58</td>
</tr>
<tr>
<td>ArActionGroup</td>
<td>58</td>
</tr>
<tr>
<td>addAction</td>
<td>59</td>
</tr>
<tr>
<td>ArActionGroup</td>
<td>59</td>
</tr>
<tr>
<td>remAction</td>
<td>59</td>
</tr>
<tr>
<td>ArActionGroupInput</td>
<td>61</td>
</tr>
<tr>
<td>ArActionGroupStop</td>
<td>62</td>
</tr>
<tr>
<td>ArActionGroupTeleop</td>
<td>63</td>
</tr>
<tr>
<td>ArActionGroupWander</td>
<td>64</td>
</tr>
<tr>
<td>ArActionInput</td>
<td>65</td>
</tr>
<tr>
<td>~ArActionInput</td>
<td>65</td>
</tr>
<tr>
<td>activate</td>
<td>65</td>
</tr>
<tr>
<td>ArActionInput</td>
<td>66</td>
</tr>
<tr>
<td>deltaHeading</td>
<td>65</td>
</tr>
<tr>
<td>deltaVel</td>
<td>65</td>
</tr>
<tr>
<td>getDesired</td>
<td>65</td>
</tr>
<tr>
<td>setVel</td>
<td>65</td>
</tr>
<tr>
<td>ArActionInput</td>
<td>65</td>
</tr>
<tr>
<td>ArActionInput</td>
<td>66</td>
</tr>
<tr>
<td>fire</td>
<td>66</td>
</tr>
<tr>
<td>ArActionJoydrive</td>
<td>67</td>
</tr>
<tr>
<td>~ArActionJoydrive</td>
<td>67</td>
</tr>
<tr>
<td>ArActionJoydrive</td>
<td>68</td>
</tr>
<tr>
<td>getDesired</td>
<td>68</td>
</tr>
<tr>
<td>getStopIfNoButtonPressed</td>
<td>67</td>
</tr>
<tr>
<td>joystickInited</td>
<td>67</td>
</tr>
<tr>
<td>setSpeeds</td>
<td>67</td>
</tr>
<tr>
<td>setStopIfNoButtonPressed</td>
<td>67</td>
</tr>
<tr>
<td>ArActionJoydrive</td>
<td>67</td>
</tr>
<tr>
<td>ArActionJoydrive</td>
<td>68</td>
</tr>
<tr>
<td>fire</td>
<td>69</td>
</tr>
<tr>
<td>getUseOSCal</td>
<td>69</td>
</tr>
<tr>
<td>setUseOSCal</td>
<td>69</td>
</tr>
<tr>
<td>ArActionKeydrive</td>
<td>70</td>
</tr>
<tr>
<td>~ArActionKeydrive</td>
<td>70</td>
</tr>
<tr>
<td>ArActionKeydrive</td>
<td>70</td>
</tr>
<tr>
<td>activate</td>
<td>70</td>
</tr>
<tr>
<td>deactivate</td>
<td>70</td>
</tr>
<tr>
<td>down</td>
<td>71</td>
</tr>
<tr>
<td>getDesired</td>
<td>70</td>
</tr>
<tr>
<td>giveUpKeys</td>
<td>71</td>
</tr>
<tr>
<td>left</td>
<td>71</td>
</tr>
<tr>
<td>right</td>
<td>71</td>
</tr>
<tr>
<td>setIncrements</td>
<td>70</td>
</tr>
<tr>
<td>setRobot</td>
<td>70</td>
</tr>
<tr>
<td>setSpeeds</td>
<td>70</td>
</tr>
<tr>
<td>space</td>
<td>71</td>
</tr>
<tr>
<td>takeKeys</td>
<td>71</td>
</tr>
<tr>
<td>up</td>
<td>71</td>
</tr>
<tr>
<td>ArActionKeydrive</td>
<td>70</td>
</tr>
<tr>
<td>fire</td>
<td>71</td>
</tr>
<tr>
<td>ArActionLimiterBackwards</td>
<td></td>
</tr>
</tbody>
</table>
ArActionLimiterBackwards, 73
ArActionLimiterBackwards, 74
getDesired, 73
ArActionLimiterBackwards, 73
ArActionLimiterBackwards, 74
fire, 74
ArActionLimiterForwards
ArActionLimiterForwards, 75
ArActionLimiterForwards, 76
getDesired, 75
ArActionLimiterForwards, 75
ArActionLimiterForwards, 76
fire, 76
ArActionLimiterTableSensor
ArActionLimiterTableSensor, 77
ArActionLimiterTableSensor, 77
getDesired, 77
ArActionLimiterTableSensor, 77
fire, 78
ArActionStallRecover
ArActionStallRecover, 79
ArActionStallRecover, 80
getDesired, 79
ArActionStallRecover, 79
ArActionStallRecover, 80
fire, 80
ArActionStop
ArActionStop, 81
ArActionStop, 81
getDesired, 81
ArActionStop, 81
ArActionStop, 81
fire, 82
ArActionTurn
ArActionTurn, 83
ArActionTurn, 83
getDesired, 83
ArActionTurn, 83
fire, 84
ArACTS_1_2
ArACTS_1_2, 85
actsHandler, 86
ArACTS_1_2, 85
BLOB_DATA_SIZE, 86
DATA_HEADER, 86
getData, 86
getRobot, 85
isConnected, 85
MAX_BLOBS, 86
MAX_DATA, 86
NUM_CHANNELS, 86
setRobot, 85
ArACTS_1_2, 85
ActsConstants, 86
closePort, 87
getBlob, 87
getNumBlobs, 87
invert, 87
openPort, 87
receiveBlobInfo, 88
requestPacket, 88
requestQuit, 88
ArACTSBlob
ArACTSBlob, 90
ArACTSBlob, 90
getArea, 90
getBottom, 90
getLeft, 90
getRight, 90
getTop, 90
getXCG, 90
getYCG, 90
print, 91
setArea, 90
setBottom, 91
setLeft, 91
setRight, 91
setTop, 91
setXCG, 90
setYCG, 90
ArACTSBlob, 90
ArAMPTU
ArAMPTU, 92
ArAMPTU, 94
canZoom, 93
getMaxNegPan, 93
INDEX

getMaxNegTilt, 93
getMaxPosPan, 93
getMaxPosTilt, 93
getPan, 93
getTilt, 93
init, 92
pan, 92
panRel, 92
panSlew, 93
panTilt, 92
panTiltRel, 92
pause, 93
purge, 93
requestStatus, 93
resume, 93
tilt, 92
tiltRel, 92
tiltSlew, 93
ArAMPTU, 92
ArAMPTU, 94
ArAMPTUCommands
ABSPAN, 95
ABSTILT, 95
CONT, 95
INIT, 96
PANSLEW, 96
PANTILT, 95
PANTILTDCCW, 95
PANTILTDCCW, 95
PANTILTUCCW, 95
PANTILTUCW, 95
PAUSE, 95
PURGE, 95
RELPANCCW, 95
RELPANCW, 95
RELTILTD, 95
RELTILTU, 95
RESP, 96
STATUS, 95
TILTSLEW, 96
ZOOM, 95
ArAMPTUCommands, 95
ArAMPTUCommand, 95
ArAMPTUPacket, 97
getUnitNumber, 98
setUnitNumber, 98
ArArg
∼ArArg, 99
ArArg, 99
BOOL, 101
clearPointers, 100
DOUBLE, 101
getBool, 100
getDescription, 100
getDouble, 100
getDouble, 100
getInt, 100
getName, 99
getPose, 100
getString, 100
getUnitNumber, 98
ArAMPTUPacket, 97
byteToBuf, 97
finalize, 97
ArArgumentBuilder
∼ArArgumentBuilder, 102
add, 102
ArArgumentBuilder, 102
getArgs, 102
gategory, 102
print, 102
ArArgumentParser
∼ArArgumentParser, 103
checkArgument, 104
checkParameters, 104

ArASyncTask
~ArASyncTask, 105
ArASyncTask, 105
create, 105
ArASyncTask, 105
runInThisThread, 106
runThread, 106
ArBasePacket
~ArBasePacket, 107
ArBasePacket, 110
bufToByte, 108
bufToByte2, 108
bufToByte4, 108
bufToUByte, 108
bufToUByte2, 108
bufToUByte4, 108
byte2ToBuf, 107
byte4ToBuf, 107
dataToBuf, 107
finalize, 107
dataToBuf, 110
dataToStr, 110
dataToBuf, 111
duplicatePacket, 111
empty, 111
resetRead, 111
strNToBuf, 111
strToBuf, 112
strToBufPadded, 112
ArCommands
ADSEL, 114
BUMPSTALL, 114
CALCOMP, 115
CLOSE, 113
CONFIG, 114
DCHEAD, 114
DHEAD, 114
DIGOUT, 114
ENABLE, 113
ENCODER, 114
ENDSIM, 114
ESTOP, 114
GETAUX, 114
Gripper, 114
GripperPacRequest, 114
GripperVal, 114
HEA, 114
IOREQUEST, 114
JOYDRIVE, 114
LOADPARAM, 114
LOADWORLD, 114
MOVE, 113
OPEN, 113
PLAYLIST, 115
POLLING, 113
PTUPOS, 114
PULSE, 113
RESETSIMTOORIGIN, 115
ROTATE, 114
RVE, 114
SAY, 114
SETO, 113
SETA, 113
SETRA, 114
SETRV, 114
SETSIMORIGINTH, 115
SETSIMORIGINX, 115
SETSIMORIGINY, 115
SETV, 113
SONAR, 114
SOUND, 115
SOUNDTOG, 115
STEP, 114
STOP, 114
TCM2, 114
TTY2, 114
<table>
<thead>
<tr>
<th>ArDeviceConnection</th>
<th>ArDPPTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ArDeviceConnection, 118</td>
<td>~ArDPPTU, 124</td>
</tr>
<tr>
<td>ArDeviceConnection, 118</td>
<td>ArDPPTU, 124</td>
</tr>
<tr>
<td>openSimple, 119</td>
<td>awaitExec, 126</td>
</tr>
<tr>
<td>STATUS_CLOSED_ERROR, 119</td>
<td>basePanSlew, 127</td>
</tr>
<tr>
<td>STATUS_CLOSED, 119</td>
<td>baseTiltSlew, 127</td>
</tr>
<tr>
<td>NORMALLY, 119</td>
<td>blank, 124</td>
</tr>
<tr>
<td>STATUS_NEVER_OPENED, 119</td>
<td>canZoom, 124</td>
</tr>
<tr>
<td>STATUS_OPEN, 119</td>
<td>disableReset, 124</td>
</tr>
<tr>
<td>STATUS_OPEN_FAILED, 119</td>
<td>disMon, 126</td>
</tr>
<tr>
<td>close, 120</td>
<td>enMon, 126</td>
</tr>
<tr>
<td>getOpenMessage, 120</td>
<td>factorySet, 125</td>
</tr>
<tr>
<td>getStatus, 120</td>
<td>getBasePanSlew, 128</td>
</tr>
<tr>
<td>getMessage, 120</td>
<td>getBaseTiltSlew, 128</td>
</tr>
<tr>
<td>getTimeRead, 121</td>
<td>getMaxNegPan, 126</td>
</tr>
<tr>
<td>isTimeStamping, 121</td>
<td>getMaxNegTilt, 126</td>
</tr>
<tr>
<td>read, 121</td>
<td>getMaxPosPan, 126</td>
</tr>
<tr>
<td>Status, 119</td>
<td>getMaxPosTilt, 126</td>
</tr>
<tr>
<td>write, 122</td>
<td>getPan, 128</td>
</tr>
<tr>
<td>writePacket, 122</td>
<td>getPanAccel, 128</td>
</tr>
<tr>
<td>MAX_PAN ACCEL, 129</td>
<td>getPanSlew, 128</td>
</tr>
<tr>
<td>MAX_PAN SLEW, 129</td>
<td>getTilt, 128</td>
</tr>
<tr>
<td>MAX_TILT, 129</td>
<td>getTiltAccel, 128</td>
</tr>
<tr>
<td>MAX_TILT ACCEL, 129</td>
<td>getTiltSlew, 128</td>
</tr>
<tr>
<td>MAX_TILT SLEW, 129</td>
<td>highMotPower, 127</td>
</tr>
<tr>
<td>MIN_PAN, 129</td>
<td>immedExec, 125</td>
</tr>
<tr>
<td>MIN_PAN ACCEL, 129</td>
<td>independMove, 127</td>
</tr>
<tr>
<td>Status, 119</td>
<td>init, 124</td>
</tr>
<tr>
<td>MAX_PAN ACCEL, 129</td>
<td>initMon, 126</td>
</tr>
<tr>
<td>MAX_PAN SLEW, 129</td>
<td>limitEnforce, 125</td>
</tr>
<tr>
<td>MAX_TILT, 129</td>
<td>lowerPanSlew, 127</td>
</tr>
<tr>
<td>MAX_TILT ACCEL, 129</td>
<td>lowerTiltSlew, 127</td>
</tr>
<tr>
<td>MIN_PAN, 129</td>
<td>lowMotPower, 127</td>
</tr>
<tr>
<td>MIN_PAN ACCEL, 129</td>
<td>lowStatPower, 127</td>
</tr>
</tbody>
</table>

INDEX

VEL, 114
VEL2, 114
ArCommands, 113
Commands, 113
ArCondition
~ArCondition, 116
ArCondition, 116
broadcast, 116
getError, 116
signal, 116
STATUS FAILED, 117
STATUS FAILED - DESTROY, 117
STATUS FAILED INIT, 117
STATUS MUTEX FAILED,
117
STATUS MUTEX FAILED -
INIT, 117
STATUS WAIT INTR, 117
STATUS_WAIT_TIMEDOUT,
117
timedWait, 116
wait, 116
ArCondition, 116
typedef, 117
INDEX

MIN_PAN_SLEW, 129
MIN_TILT, 129
MIN_TILT_ACCEL, 129
MIN_TILT_SLEW, 129
offStatPower, 126
pan, 125
panAccel, 127
panRel, 125
panSlew, 128
panSlewRel, 128
panTilt, 125
panTiltRel, 125
regMotPower, 127
regStatPower, 126
resetAll, 125
resetCalib, 124
resetPan, 125
resetTilt, 125
restoreSet, 125
saveSet, 125
slaveExec, 126
tilt, 125
tiltAccel, 127
tiltRel, 125
tiltSlew, 128
tiltSlewRel, 128
upperPanSlew, 127
upperTiltSlew, 127
velMove, 127
ArDPPTU, 124
ArDPPTUCmds, 130
ACCEL, 130
BASE, 130
CONTROL, 130
DELIM, 130
DISABLE, 130
ENABLE, 130
FACTORY, 130
HALT, 130
IMMED, 131
INIT, 130
LIMIT, 131
MONITOR, 131
OFFSET, 131
PAN, 131
RESET, 131
SPEED, 131
TILT, 131
UPPER, 131
VELOCITY, 131
ArDPPTU, 130
ArDPPTUPacket~ArDPPTU, 132
ArDPPTUPacket, finalize, 132
ArDPPTUPacket, areMotorsEnabled
ArRobot, 294
areSonarsEnabled
ArRobot, 294
ArFunctor
ArFunctor~ArFunctor1, 133
invoke, 133
operator(), 133
ArFunctor1
ArFunctor1~ArFunctor1, 135
invoke, 135
operator(), 135
ArFunctor1C
ArFunctor1C~ArFunctor1C, 137
ArFunctor1C, 137–139
invoke, 137
operator(), 137
ArFunctor2
ArFunctor2~ArFunctor2, 141
invoke, 141
operator(), 141
ArFunctor2C
ArFunctor2C~ArFunctor2C, 144
INDEX

ArFunctor2C, 144, 146, 147
  invoke, 144
  operator(), 145
ArFunctor2C, 144
  ArFunctor2C, 146, 147
    invoke, 147
    operator(), 148
    setP1, 148
    setP2, 148
    setThis, 149
ArFunctorC
  ~ArFunctorC, 150
  ArFunctorC, 150, 151
    invoke, 150
    operator(), 150
ArFunctorC, 150
  ArFunctorC, 151
    setThis, 151
ArGlobalFunctor
  ~ArGlobalFunctor, 153
  ArGlobalFunctor, 153, 154
    invoke, 153
    operator(), 153
ArGlobalFunctor, 153
  ArGlobalFunctor, 154
ArGlobalFunctor1
  ~ArGlobalFunctor1, 155
  ArGlobalFunctor1, 155, 156
    invoke, 155
    operator(), 155
ArGlobalFunctor1, 155
  ArGlobalFunctor1, 156
    invoke, 156
    operator(), 157
    setP1, 157
ArGlobalFunctor2
  ~ArGlobalFunctor2, 158
  ArGlobalFunctor2, 158–160
    invoke, 158
    operator(), 159
ArGlobalFunctor2, 158
  ArGlobalFunctor2, 159, 160
    invoke, 160
    operator(), 161
    setP1, 161
    setP2, 161
ArGlobalRetFunctor
  ~ArGlobalRetFunctor, 163
  ArGlobalRetFunctor, 163, 164
  invokeR, 163
ArGlobalRetFunctor, 163
  ArGlobalRetFunctor, 164
ArGlobalRetFunctor1
  ~ArGlobalRetFunctor1, 165
  ArGlobalRetFunctor1, 165, 166
  invokeR, 165
ArGlobalRetFunctor1, 165
  ArGlobalRetFunctor1, 166
    setP1, 167
ArGlobalRetFunctor2
  ~ArGlobalRetFunctor2, 168
  ArGlobalRetFunctor2, 168, 169
  invokeR, 168
ArGlobalRetFunctor2, 168
  ArGlobalRetFunctor2, 169
    setP1, 170
    setP2, 170
ArGripper
  ~ArGripper, 172
  ArGripper, 174
    connectHandler, 174
    GENIO, 174
    GRIPPAC, 174
    NOGRIPPER, 174
    packetHandler, 174
    printState, 174
    QUERYTYPE, 174
    USERIO, 174
ArGripper, 172
  ArGripper, 174
    getBreakBeamState, 175
    getGraspTime, 175
    getGripState, 175
    getMSecSinceLastPacket, 175
    getPaddleState, 175
    getType, 176
    gripClose, 176
    gripOpen, 176
gripperDeploy, 176
gripperHalt, 176
gripperStore, 176
gripPressure, 177
gripStop, 177
isGripMoving, 177
isLiftMaxed, 177
isLiftMoving, 177
liftCarry, 177
liftDown, 178
liftStop, 178
liftUp, 178
setType, 178
Type, 174
ArGripperCommands
  GRIP_CLOSE, 179
  GRIP_OPEN, 179
  GRIP_PRESSURE, 179
  GRIP_STOP, 179
  GRIPPER_DEPLOY, 179
  GRIPPER_HALT, 179
  GRIPPER_STORE, 179
  LIFT_CARRY, 180
  LIFT_DOWN, 179
  LIFT_STOP, 179
  LIFT_UP, 179
ArGripperCommands, 179
  Commands, 179
Aria, 181
  addInitCallback, 182
  addRobot, 181
  addUninitCallback, 182
  delRobot, 181
  exit, 183
  findRobot, 183
  getDirectory, 183
  getKeyHandler, 182
  getRobotList, 182
  init, 183
  setDirectory, 184
  setKeyHandler, 182
  shutdown, 184
  SIGHANDLE_NONE, 182
  SIGHANDLE_SINGLE, 182
  SIGHANDLE_THREAD, 182
 _SIGHandleMethod, 182
  signalHandlerCB, 182
  uninit, 185
ArInterpolation
  ~ArInterpolation, 186
  addReading, 186
  ArInterpolation, 186
  getNumberOfReadings, 186
  reset, 186
  setNumberOfReadings, 186
ArInterpolation, 186
  getPose, 187
ArIrrfDevice
  ~ArIrrfDevice, 188
  ArIrrfDevice, 188
  setCumulativeMaxRange, 188
  setRobot, 188
ArIrrfDevice, 188
  packetHandler, 189
ArJoyHandler
  ~ArJoyHandler, 190
  ArJoyHandler, 191
  getSpeeds, 191
  getStats, 191
  haveJoystick, 190
  init, 190
  setSpeeds, 190
  setStats, 191
ArJoyHandler, 190
  endCal, 192
  getAdjusted, 192
  getButton, 192
  getDoubles, 192
  getUnfiltered, 193
  getUseOSCal, 193
  setUseOSCal, 193
  startCal, 194
ArKeyHandler
  ~ArKeyHandler, 195
  ArKeyHandler, 191
  endCal, 192
  getAdjusted, 192
  getButton, 192
  getDoubles, 192
  getUnfiltered, 193
  getUseOSCal, 193
  setUseOSCal, 193
  startCal, 194
ArKeyHandler
  ~ArKeyHandler, 195
  ArKeyHandler, 196
  BACKSPACE, 196
  checkKeys, 195
  DOWN, 196
  ENTER, 196
  ESCAPE, 196
  F1, 196
<table>
<thead>
<tr>
<th>ArKeyHandler, 195</th>
</tr>
</thead>
<tbody>
<tr>
<td>addKeyHandler, 197</td>
</tr>
<tr>
<td>ArKeyHandler, 196</td>
</tr>
<tr>
<td>KEY, 196</td>
</tr>
<tr>
<td>remKeyHandler, 197</td>
</tr>
<tr>
<td>ArListPos</td>
</tr>
<tr>
<td>FIRST, 199</td>
</tr>
<tr>
<td>LAST, 199</td>
</tr>
<tr>
<td>ArListPos, 199</td>
</tr>
<tr>
<td>Pos, 199</td>
</tr>
<tr>
<td>ArLog</td>
</tr>
<tr>
<td>close, 200</td>
</tr>
<tr>
<td>Colbert, 201</td>
</tr>
<tr>
<td>File, 201</td>
</tr>
<tr>
<td>None, 201</td>
</tr>
<tr>
<td>Normal, 200</td>
</tr>
<tr>
<td>StdErr, 201</td>
</tr>
<tr>
<td>StdOut, 201</td>
</tr>
<tr>
<td>Tense, 200</td>
</tr>
<tr>
<td>Verbose, 200</td>
</tr>
<tr>
<td>ArLog, 200</td>
</tr>
<tr>
<td>init, 201</td>
</tr>
<tr>
<td>log, 201</td>
</tr>
<tr>
<td>LogLevel, 200</td>
</tr>
<tr>
<td>LogType, 200</td>
</tr>
<tr>
<td>ArMath</td>
</tr>
<tr>
<td>angleBetween, 202</td>
</tr>
<tr>
<td>pointRotate, 202</td>
</tr>
<tr>
<td>ArMath, 202</td>
</tr>
<tr>
<td>addAngle, 203</td>
</tr>
<tr>
<td>atan2, 203</td>
</tr>
<tr>
<td>cos, 203</td>
</tr>
<tr>
<td>degToRad, 204</td>
</tr>
<tr>
<td>fabs, 204</td>
</tr>
<tr>
<td>fixAngle, 204</td>
</tr>
<tr>
<td>radToDeg, 204</td>
</tr>
<tr>
<td>roundInt, 205</td>
</tr>
<tr>
<td>sin, 205</td>
</tr>
<tr>
<td>subAngle, 205</td>
</tr>
<tr>
<td>ArMode</td>
</tr>
<tr>
<td>~ArMode, 207</td>
</tr>
<tr>
<td>activate, 208</td>
</tr>
<tr>
<td>ArMode, 209</td>
</tr>
<tr>
<td>baseHelp, 208</td>
</tr>
<tr>
<td>deactivate, 208</td>
</tr>
<tr>
<td>getKey, 208</td>
</tr>
<tr>
<td>ArMode, 207</td>
</tr>
<tr>
<td>getKey2, 208</td>
</tr>
<tr>
<td>getName, 207</td>
</tr>
<tr>
<td>userTask, 208</td>
</tr>
<tr>
<td>ArMode, 209</td>
</tr>
<tr>
<td>baseActivate, 209</td>
</tr>
<tr>
<td>baseDeactivate, 209</td>
</tr>
<tr>
<td>help, 209</td>
</tr>
<tr>
<td>ArModeCamera</td>
</tr>
<tr>
<td>~ArModeCamera, 211</td>
</tr>
<tr>
<td>activate, 211</td>
</tr>
<tr>
<td>ArModeCamera, 211</td>
</tr>
<tr>
<td>deactivate, 211</td>
</tr>
<tr>
<td>userTask, 211</td>
</tr>
<tr>
<td>ArModeCamera, 211</td>
</tr>
<tr>
<td>help, 212</td>
</tr>
<tr>
<td>ArModeGripper</td>
</tr>
<tr>
<td>~ArModeGripper, 213</td>
</tr>
<tr>
<td>activate, 213</td>
</tr>
<tr>
<td>ArModeGripper, 213</td>
</tr>
<tr>
<td>deactivate, 213</td>
</tr>
<tr>
<td>userTask, 213</td>
</tr>
<tr>
<td>ArModeGripper, 213</td>
</tr>
<tr>
<td>help, 214</td>
</tr>
<tr>
<td>ArModeSonar</td>
</tr>
<tr>
<td>~ArModeSonar, 215</td>
</tr>
<tr>
<td>activate, 215</td>
</tr>
<tr>
<td>ArModeSonar, 215</td>
</tr>
<tr>
<td>deactivate, 215</td>
</tr>
<tr>
<td>userTask, 215</td>
</tr>
<tr>
<td>ArModeSonar, 215</td>
</tr>
<tr>
<td>help, 216</td>
</tr>
<tr>
<td>ArModeTeleop</td>
</tr>
<tr>
<td>~ArModeTeleop, 217</td>
</tr>
<tr>
<td>activate, 217</td>
</tr>
<tr>
<td>ArModeTeleop, 217</td>
</tr>
</tbody>
</table>
deactivate, 217
ArModeTeleop, 217
help, 217
ArModeWander
~ArModeWander, 219
activate, 219
ArModeWander, 219
deactivate, 219
ArModeWander, 219
help, 219
ArModule
~ArModule, 221
ArModule, 221
exit, 221
getRobot, 221
myRobot, 221
setRobot, 221
ArModule, 221
init, 222
ArModuleLoader
closeAll, 224
STATUS_ALREADY_LOADED, 225
STATUS_EXIT_FAILED, 225
STATUS_FAILED_OPEN, 225
STATUS_INIT_FAILED, 225
STATUS_INVALID, 225
STATUS_NOT_FOUND, 225
STATUS_SUCCESS, 225
ArModuleLoader, 224
close, 225
load, 225
reload, 226
Status, 225
ArMutex
~ArMutex, 227
ArMutex, 227
gerError, 227
getMutex, 227
STATUS_ALREADY_LOCKED, 228
STATUS_FAILED, 228
STATUS_FAILED_INIT, 228
unlock, 227
ArMutex, 227
lock, 228
Status, 228
tryLock, 228
ArPeriodicTask
~ArPeriodicTask, 229
ArPeriodicTask, 229
invoke, 229
isRunning, 229
runTask, 229
setRunning, 229
ArPeriodicTask, 229
ArPose
~ArPose, 231
ArPose, 231, 232
getTh, 232
getThRad, 232
gGetX, 232
gGetY, 232
print, 232
setTh, 231
setThRad, 231
setX, 231
setY, 231
ArPose, 231
ArPose, 232
findAngleTo, 233
findDistanceTo, 233
gPose, 233
setPose, 233, 234
ArPoseWithTime, 235
ArPref
~ArPref, 236
ArPref, 236
Bool, 238
Double, 238
Integer, 238
String, 238
ArPref, 236
gerBool, 238
gerBoolSet, 238
gerDouble, 238
gerDoubleSet, 239
gerInt, 239
gerIntSet, 239
getSetCount, 239
getString, 240
getStringSet, 240
INDEX

setBool, 240
setBoolSet, 240
setDouble, 241
setDoubleSet, 241
setInt, 241
setIntSet, 242
setString, 242
ValType, 238

ArPriorityResolver
~ ArPriorityResolver, 243
ArPriorityResolver, 243

ArPTZ
~ ArPTZ, 244
ArPTZ, 247
canGetRealPanTilt, 245
canGetRealZoom, 245
canZoom, 244
connectHandler, 246
getDeviceConnection, 246
getMaxNegPan, 245
getMaxNegTilt, 245
getMaxPosPan, 245
getMaxPosTilt, 245
getMaxZoom, 246
getMinZoom, 246
getPan, 245
getRealPan, 245
getRealTilt, 245
getRealZoom, 245
getTilt, 245
getZoom, 245
init, 244
pan, 244
panRel, 244
panTilt, 244
panTiltRel, 244
sensorInterpHandler, 246
tilt, 244
tiltRel, 244
zoom, 245
zoomRel, 245

ArPTZ, 244
ArPTZ, 247
packetHandler, 247
readPacket, 248
robotPacketHandler, 248
sendPacket, 248
setDeviceConnection, 249

ArRangeBuffer
~ ArRangeBuffer, 250
ArRangeBuffer, 250
getPosTaken, 250
setSize, 250
reset, 251
setPosTaken, 250

ArRangeBuffer, 250
addReading, 251
applyTransform, 252
ArRangeBuffer, 251
beginInvalidationSweep, 252
beginRedoBuffer, 252
endInvalidationSweep, 252
endRedoBuffer, 253
getBuffer, 253
closestBox, 253
closestPolar, 254
invalidateReading, 254
redoReading, 255
setSize, 255

ArRangeDevice
~ ArRangeDevice, 255
addReading, 256
ArRangeDevice, 257
cumulativeBuffer, 257
cumulativeRangeBuffer, 257
currentBuffer, 257
currentRangeBuffer, 257
defmaxRange, 257
defname, 256
defRobot, 256
defSize, 255
defRobot, 256

cumulativeReadingBox, 259
cumulativeReadingPolar, 259
currentReadingBox, 260
currentReadingPolar, 260
getRawReadings, 261
lockDevice, 261
setCumulativeBufferSize, 261
setCurrentBufferSize, 262
tryLockDevice, 262
unlockDevice, 262
ArRangeDeviceThreaded
~ArRangeDeviceThreaded, 264
ArRangeDeviceThreaded, 264
lockDevice, 265
run, 265
runAsync, 265
runThread, 265
tryLockDevice, 266
unlockDevice, 266
ArRecurrentTask
~ArRecurrentTask, 267
ArRecurrentTask, 267
go, 267
reset, 267
ArRecurrentTask, 267
done, 268
runThread, 268
task, 268
ArResolver
~ArResolver, 269
ActionMap, 269
description, 269
getName, 269
resolve, 269
ArResolver, 269
ArRetFunctor
~ArRetFunctor, 271
invoke, 271
invokeR, 271
operator(), 271
ArRetFunctor, 271
ArRetFunctor1
~ArRetFunctor1, 273
invokeR, 273
ArRetFunctor1, 273
invokeR, 274
ArRetFunctor1C
~ArRetFunctor1C, 275
ArRetFunctor1C, 275–277
invokeR, 275
ArRetFunctor1C, 276, 277
invokeR, 277
setP1, 277
setThis, 278
ArRetFunctor2
~ArRetFunctor2, 279
invokeR, 279
ArRetFunctor2, 279
invokeR, 280
ArRetFunctor2C
~ArRetFunctor2C, 282
ArRetFunctor2C, 281, 283, 284
invokeR, 282
ArRetFunctor2C, 281
ArRetFunctor2C, 283, 284
invokeR, 284
setP1, 285
setP2, 285
setThis, 285
ArRetFunctorC
~ArRetFunctorC, 287
ArRetFunctorC, 287, 288
invokeR, 287
ArRetFunctorC, 287
ArRetFunctorC, 288
setThis, 288
ArRobot
~ArRobot, 290
addRangeDevice, 297
areMotorsEnabled, 294
areSonarsEnabled, 294
ArRobot, 305
dropConnection, 304
failedConnect, 304
get, 292
getAnalog, 295
getAnalogPortSelected, 295
getBatteryVoltage, 294
getClosestSonarNumber, 297
getClosestSonarRange, 296
getCompass, 295
getCounter, 302
getDigIn, 295
getDigOut, 295
INDEX

getEncoderPose, 296
getFlags, 294
getIOAnalog, 295
getIOAnalogSize, 295
getIODigIn, 295
getIODigInSize, 295
getIODigOut, 295
getIODigOutSize, 295
getIPacketTime, 296
getKeyHandler, 302
getLeftVel, 294
getMaxRotVel, 293
getMaxTransVel, 293
getMoveDoneDist, 292
getName, 297
getNumSonar, 296
getParams, 296
getPose, 293
getPoseInterpNumReadings, 302
getResolver, 301
getRightVel, 294
getRobotDiagonal, 294
getRobotName, 293
getRobotRadius, 294
getRobotSubType, 293
getRobotType, 293
getRotVel, 294
getRunExitListCopy, 304
getSonarPacCount, 296
getStallValue, 294
getTh, 293
getVel, 294
getX, 293
getY, 293
handlePacket, 304
hasFrontBumpers, 296
hasRearBumpers, 296
hasTableSensingIR, 295
incCounter, 303
isCycleChained, 302
isLeftBreakBeamTriggered, 296
isLeftMotorStalled, 294
isLeftTableSensingIRTriggered, 295
isRightBreakBeamTriggered, 296
isRightMotorStalled, 294
isRightTableSensingIRTriggered, 295
keyHandlerExit, 303
lock, 302
madeConnection, 304
printActions, 301
processEncoderPacket, 304
processIPacket, 304
processMotorPacket, 303
processNewSonar, 303
processParamFile, 304
setCycleChained, 302
setUpPacketHandlers, 304
setUpSyncList, 304
tryLock, 303
unlock, 303
WAIT CONNECTED, 305
WAIT FAIL, 305
WAIT FAILED_CONN, 305
WAIT_INTR, 305
WAIT_RUN_EXIT, 305
WAIT_TIMEDOUT, 305

ArRobot, 290
actionHandler, 306
addAction, 306
addConnectCB, 306
addDisconnectNormallyCB, 306
addDisconnectOnErrorCB, 307
addFailedConnectCB, 307
addPacketHandler, 308
addRunExitCB, 308
addSensorInterpTask, 309
addUserTask, 309

applyTransform, 310
ArRobot, 305
asyncConnect, 310
asyncConnectHandler, 311
attachKeyHandler, 311
blockingConnect, 311
checkRangeDevicesCumulativeBox, 312
checkRangeDevicesCumulativePolar, 313
checkRangeDevicesCurrentBox, 313
checkRangeDevicesCurrentPolar, 314
clearDirectMotion, 314
com, 314
com2Bytes, 315
comInt, 315
comStr, 315
comStrN, 316
disableMotors, 316
disconnect, 316
enableMotors, 316
findAction, 316
findRangeDevice, 317
findTask, 317
findUserTask, 317, 318
getActionMap, 318
getConnectionTimeoutTime, 318
getControl, 318
cycleTime, 319
getDeviceConnection, 319
getDirectMotionPrecedenceTime, 319
getEncoderCorrectionCallback, 320
ggetLastPacketTime, 320
getPoseInterpPosition, 320
gRangeDeviceList, 320
getRobotParams, 321
gSonarRange, 321
gSonarReading, 321
getStateReflectionRefreshTime, 321
gSyncTaskRoot, 322
gToLocalTransform, 322
gToGlobalTransform, 322
hasRangeDevice, 322
init, 323
isConnected, 323
isDirectMotion, 323
isHeadingDone, 323
isMoveDone, 324
isRunning, 324
isSonarNew, 324
loadParamFile, 324
loopOnce, 325
move, 325
moveTo, 325
packetHandler, 326
printAllTasks, 326
printUserTasks, 326
remAction, 326, 327
remConnectCB, 327
remDisconnectNormallyCB, 327
remDisconnectOnErrorCB, 327
remFailedConnectCB, 328
remPacketHandler, 328
remRangeDevice, 328
remRunExitCB, 328
remSensorInterpTask, 329
remUserTask, 329
robotLocker, 329
robotUnlocker, 330
run, 330
runAsync, 330
setConnectionTimeoutTime, 330
setCycleTime, 331
setDeadReconPose, 331
setDeltaHeading, 331
setDeviceConnection, 331
setDirectMotionPrecedenceTime, 332
setEncoderCorrectionCallback, 332
setEncoderTransform, 333
setHeading, 333
setMaxRotVel, 333
INDEX

setMaxTransVel, 334
setRotVel, 334
setStateReflectionRefresh-Time, 334
setVel, 335
setVel2, 335
stateReflector, 335
stop, 336
stopRunning, 336
waitForConnect, 336
waitForConnectOrConnFail, 337
waitForRunExit, 337
WaitState, 305
wakeAllConnOrFailWaitingThreads, 337
wakeAllConnWaitingThreads, 338
wakeAllRunExitWaitingThreads, 338
wakeAllWaitingThreads, 338
ArRobotPacket
~ArRobotPacket, 340
ArRobotPacket, 341
calcCheckSum, 340
finalize, 340
getID, 340
getimeReceived, 340
setTimeReceived, 340
verifyCheckSum, 340
ArRobotPacket, 340
ArRobotPacket, 341
resetRead, 341
ArRobotPacketReceiver
~ArRobotPacketReceiver, 342
ArRobotPacketReceiver, 343
getDeviceConnection, 342
isAllocatingPackets, 342
setDeviceConnection, 342
ArRobotPacketReceiver, 342
ArRobotPacketReceiver, 343
receivePacket, 343
ArRobotPacketSender
~ArRobotPacketSender, 345
ArRobotPacketSender, 346
getDeviceConnection, 345
ArRobotParams
~ArRobotParams, 349
ArRobotParams, 349
getAngleConvFactor, 350
getClass, 349
getDiffConvFactor, 350
getDistConvFactor, 350
getLaserFlipped, 351
getLaserPort, 351
getLaserPossessed, 351
getLaserPowerControlled, 351
getLaserX, 351
getLaserY, 351
getMaxRotVelocity, 349
getMaxVelocity, 349
getNumSonar, 350
getRangeConvFactor, 350
getRequestIOPackets, 349
getRobotDiagonal, 349
getRobotRadius, 349
getSonarTh, 351
getSonarX, 351
getSonarY, 351
getSubClass, 349
getVel2Divisor, 350
getVelConvFactor, 350
hasMoveCommand, 349
haveFrontBumpers, 350
haveNewTableSensingIR, 350
haveRearBumpers, 350
haveSonar, 351
haveTableSensingIR, 350
init, 349
isHolonomic, 349
numFrontBumpers, 350
numRearBumpers, 350
numSonar, 351

ArSectors, 352
ArSectors, 352
clear, 352
didAll, 352
update, 352
ArSectors, 352
ArSensorReading
ArSensorReading, 355
ArSensorReading::newData, 354
getPose, 353
getPosTaken, 354
getSensorDX, 353
getSensorDY, 354
getSensorTh, 353
getSensorX, 353
getSensorY, 353
getThTaken, 354
getX, 353
getXTaken, 354
getY, 353
getYTaken, 354
ArSensorReading, 353
applyTransform, 355
ArSensorReading, 355
getCounterTaken, 355
getRange, 355
getSensorPosition, 355
isNew, 356
newData, 356
resetSensorPosition, 356
ArSensorReading::newData
ArSensorReading, 354
ArSerialConnection
~ArSerialConnection, 358
ArSerialConnection, 358
OPEN_ALREADY_OPEN, 360
OPEN_COULD_NOT_OPEN_-PORT, 359
OPEN_COULD_NOT_SET_-BAUD, 359
OPEN_COULD_NOT_SET_-UP_PORT, 359
OPEN_INVALID_BAUD_-RATE, 359
openSimple, 358
ArSerialConnection, 358
close, 360
getBaud, 360
getHardwareControl, 360
getOpenMessage, 360
getStatus, 360
g getTimeRead, 361
isTimeStamping, 361
Open, 359
open, 361
read, 362
setBaud, 362
setHardwareControl, 362
setPort, 363
write, 363
ArSick
~ArSick, 364
ArSick, 364
BAUD19200, 368
BAUD38400, 369
BAUD9600, 368
DEGREES100, 369
DEGREES180, 369
dropConnection, 368
failedConnect, 368
getDegrees, 367
deviceConnection, 365
getFilterNearDist, 367
getIncrement, 367
ggetLastReadingTime, 366
getSensorPosition, 365
getSensorPositionTh, 365
getSensorPositionX, 365
getSensorPositionY, 365
getSickPacCount, 366
INCREMENT_HALF, 369
INCREMENT_ONE, 369
isConnected, 365
isControllingPower, 367
isLaserFlipped, 367
isUsingSim, 367
madeConnection, 368
processPacket, 367
robotConnectCallback, 368
runOnce, 367
runThread, 367
sensorInterpCallback, 367
setDeviceConnection, 365
setFilterNearDist, 366
setRobot, 367
setSensorPosition, 365
simPacketHandler, 367
STATE_CHANGE_BAUD, 369
STATE_CONFIGURE, 369
STATE_CONNECTED, 369
STATE_INIT, 369
STATE_INSTALL_MODE, 369
STATE_NONE, 369
STATE_SET_MODE, 369
STATE_START_READINGS, 369
STATE_WAIT_FOR_-CONFIGURE_ACK, 369
STATE_WAIT_FOR_-INSTALL_MODE_ACK, 369
STATE_WAIT_FOR_-POWER_ON, 369
STATE_WAIT_FOR_SET_-MODE_ACK, 369
STATE_WAIT_FOR_START_-ACK, 369
switchState, 368
tryingToConnect, 365
ArSick, 364
addConnectCB, 370
addDisconnectNormallyCB, 370
addDisconnectOnErrorCB, 370
addFailedConnectCB, 371
asyncConnect, 371
BaudRate, 368
blockingConnect, 372
calculate, 372
calculateShort, 372
Degrees, 369
disconnect, 372
filterReadings, 373
getConnectionTimeoutTime, 373
Increment, 369
internalConnectHandler, 373
internalConnectSim, 373
remConnectCB, 374
remDisconnectNormallyCB, 374
remDisconnectOnErrorCB, 374
remFailedConnectCB, 374
runOnRobot, 375
setConnectionTimeoutTime, 375
State, 369
ArSickLogger
~ArSickLogger, 376
ArSickLogger, 377
getDegDiff, 376
getDistDiff, 376
robotTask, 376
setDegDiff, 376
setDistDiff, 376
ArSickLogger, 376
ArSickLogger, 377
ArSickPacket
~ArSickPacket, 378
ArSickPacket, 378
calcCRC, 378
finalize, 378
getID, 378
g getTimeReceived, 379
setTimeReceived, 379
verifyCRC, 378
ArSickPacket, 378
duplicatePacket, 379
getReceivedAddress, 379
getSendingAddress, 380
resetRead, 380
setSendingAddress, 380
ArSickPacketReceiver
~ArSickPacketReceiver, 381
ArSickPacketReceiver, 382
getDeviceConnection, 381
isAllocatingPackets, 381
setDeviceConnection, 381
<table>
<thead>
<tr>
<th>Function/Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArSickPacketReceiver</td>
<td>381</td>
</tr>
<tr>
<td>ArSickPacketReceiver</td>
<td>382</td>
</tr>
<tr>
<td>receivePacket</td>
<td>382</td>
</tr>
<tr>
<td>ArSignalHandler</td>
<td>384</td>
</tr>
<tr>
<td>~ArSignalHandler</td>
<td>384</td>
</tr>
<tr>
<td>nameSignal</td>
<td>385</td>
</tr>
<tr>
<td>ArSignalHandler</td>
<td>384</td>
</tr>
<tr>
<td>addHandlerCB</td>
<td>386</td>
</tr>
<tr>
<td>block</td>
<td>386</td>
</tr>
<tr>
<td>blockCommon</td>
<td>386</td>
</tr>
<tr>
<td>blockCommonThisThread</td>
<td>387</td>
</tr>
<tr>
<td>createHandlerNonThreaded</td>
<td>387</td>
</tr>
<tr>
<td>createHandlerThreaded</td>
<td>387</td>
</tr>
<tr>
<td>delHandlerCB</td>
<td>387</td>
</tr>
<tr>
<td>getHandler</td>
<td>388</td>
</tr>
<tr>
<td>handle</td>
<td>388</td>
</tr>
<tr>
<td>runThread</td>
<td>388</td>
</tr>
<tr>
<td>unblock</td>
<td>388</td>
</tr>
<tr>
<td>unblockAll</td>
<td>389</td>
</tr>
<tr>
<td>unhandle</td>
<td>389</td>
</tr>
<tr>
<td>ArSocket</td>
<td>390</td>
</tr>
<tr>
<td>~ArSocket</td>
<td>390</td>
</tr>
<tr>
<td>accept</td>
<td>391</td>
</tr>
<tr>
<td>addHost</td>
<td>392</td>
</tr>
<tr>
<td>ArSocket</td>
<td>390, 393</td>
</tr>
<tr>
<td>close</td>
<td>391</td>
</tr>
<tr>
<td>connect</td>
<td>390</td>
</tr>
<tr>
<td>connectTo</td>
<td>390, 391</td>
</tr>
<tr>
<td>copy</td>
<td>390</td>
</tr>
<tr>
<td>create</td>
<td>390</td>
</tr>
<tr>
<td>findValidPort</td>
<td>390</td>
</tr>
<tr>
<td>getError</td>
<td>392</td>
</tr>
<tr>
<td>getErrorStr</td>
<td>392</td>
</tr>
<tr>
<td>getFD</td>
<td>392</td>
</tr>
<tr>
<td>getHostName</td>
<td>392</td>
</tr>
<tr>
<td>getSockName</td>
<td>391</td>
</tr>
<tr>
<td>getType</td>
<td>392</td>
</tr>
<tr>
<td>hostAddr</td>
<td>392</td>
</tr>
<tr>
<td>hostToNetOrder</td>
<td>393</td>
</tr>
<tr>
<td>inAddr</td>
<td>391</td>
</tr>
<tr>
<td>inPort</td>
<td>391</td>
</tr>
<tr>
<td>inToA</td>
<td>392</td>
</tr>
<tr>
<td>maxHostNameLen</td>
<td>393</td>
</tr>
<tr>
<td>netToHostOrder</td>
<td>393</td>
</tr>
<tr>
<td>open</td>
<td>390</td>
</tr>
<tr>
<td>recvFrom</td>
<td>391</td>
</tr>
<tr>
<td>sendTo</td>
<td>391</td>
</tr>
<tr>
<td>setBroadcast</td>
<td>391</td>
</tr>
<tr>
<td>setDoClose</td>
<td>392</td>
</tr>
<tr>
<td>setLinger</td>
<td>391</td>
</tr>
<tr>
<td>setNonBlock</td>
<td>392</td>
</tr>
<tr>
<td>setReuseAddress</td>
<td>392</td>
</tr>
<tr>
<td>sockAddrIn</td>
<td>391</td>
</tr>
<tr>
<td>sockAddrLen</td>
<td>393</td>
</tr>
<tr>
<td>ArSocket</td>
<td>390</td>
</tr>
<tr>
<td>ArSonarDevice</td>
<td>396</td>
</tr>
<tr>
<td>~ArSonarDevice</td>
<td>396</td>
</tr>
<tr>
<td>processReadings</td>
<td>396</td>
</tr>
<tr>
<td>setCumulativeMaxRange</td>
<td>396</td>
</tr>
<tr>
<td>setRobot</td>
<td>396</td>
</tr>
<tr>
<td>ArSonarDevice</td>
<td>396</td>
</tr>
<tr>
<td>addReading</td>
<td>397</td>
</tr>
<tr>
<td>ArSonyPacket</td>
<td>398</td>
</tr>
<tr>
<td>ArSonyPacket</td>
<td>398</td>
</tr>
<tr>
<td>byte2ToBufAtPos</td>
<td>398</td>
</tr>
<tr>
<td>ArSonyPTZ</td>
<td>400</td>
</tr>
<tr>
<td>canZoom</td>
<td>400</td>
</tr>
<tr>
<td>getMaxNegPan</td>
<td>401</td>
</tr>
<tr>
<td>getMaxNegTilt</td>
<td>401</td>
</tr>
<tr>
<td>getMaxPosPan</td>
<td>401</td>
</tr>
<tr>
<td>getMaxPosTilt</td>
<td>401</td>
</tr>
<tr>
<td>getMaxZoom</td>
<td>401</td>
</tr>
<tr>
<td>getMinZoom</td>
<td>401</td>
</tr>
<tr>
<td>getPan</td>
<td>401</td>
</tr>
<tr>
<td>getTilt</td>
<td>401</td>
</tr>
<tr>
<td>getZoom</td>
<td>401</td>
</tr>
<tr>
<td>init</td>
<td>400</td>
</tr>
<tr>
<td>MAX_PAN</td>
<td>402</td>
</tr>
<tr>
<td>MAX_TILT</td>
<td>402</td>
</tr>
<tr>
<td>MAX_ZOOM</td>
<td>402</td>
</tr>
</tbody>
</table>
INDEX 465

MIN_ZOOM, 402
pan, 400
panRel, 400
panTilt, 400
panTiltRel, 400
tilt, 400
tiltRel, 400
zoom, 401
zoomRel, 401
ArSonyPTZ, 400

ArSyncTask
ArSyncTask, 405
defFunctor, 404
defName, 404
defState, 403
defState, 403
ArSyncTask, 403
ArSyncTask, 405
addNewBranch, 405
addNewLeaf, 405
ArSyncTask, 405
find, 406
findNonRecursive, 406, 407
print, 407
run, 407

ArTaskPool
ArTaskPool, 408
forceUninit, 408
getPool, 408
init, 408
runTask, 408
setPool, 408
uninit, 408
ArTaskPool, 408
ourPool, 409

ArTaskState
ACTIVE, 410
FAILURE, 410
INIT, 410
RESUME, 410
SUCCESS, 410
SUSPEND, 410
USER_START, 410
ArTaskState, 410
State, 410

ArTcpConnection
∼ArTcpConnection, 411
ArTcpConnection, 411
getSocket, 412
internalOpen, 412
OPEN_BAD_HOST, 412
OPEN_CON_REFUSED, 413
OPEN_NET_FAIL, 412
OPEN_NO_ROUTE, 413
openSimple, 411
setStatus, 412
ArSyncTask, 403
close, 413
defHost, 413
defOpenMessage, 413
defPort, 413
defStatus, 414
getTimeRead, 414
isTimeStamping, 414
Open, 412
open, 414
read, 415
setSocket, 415
write, 415

ArThread
∼ArThread, 417
ArThread, 417
cancel, 418
cancelAll, 419
create, 417
detach, 418
defBlockAllSignals, 418
defError, 418
defFunc, 418
defJoinable, 418
defRunning, 418
defRunningWithLock, 418
defThread, 418
join, 418
joinAll, 419
lock, 418
myRunning, 419
setRunning, 418
STATUS_ALREADY_DETACHED, 420
STATUS_FAILED, 420

STATUS_INVALID, 420
STATUS_JOIN_SELF, 420
STATUS_NO_SUCH_THREAD, 420
STATUS_NORESOURCE, 420
stopAll, 419
stopRunning, 417
tryLock, 418
unlock, 418
yield, 419
ArThread, 417
init, 420
self, 420
Status, 420
ArTime
~ArTime, 421
addMSec, 422
ArTime, 421
getMSec, 422
getSec, 422
isAfter, 421
isAt, 421
isBefore, 421
mSecSince, 421
mSecTo, 421
print, 422
secSince, 421
secTo, 421
setMSec, 422
setSec, 422
setToNow, 421
ArTime, 421
ArTransform
~ArTransform, 423
ArTransform, 423
doTransform, 423
getTh, 423
ArTransform, 423
doInvTransform, 424
doTransform, 424
setTransform, 424
ArTypes
Byte, 426
Byte2, 426
Byte4, 426
UByte, 426
UByte2, 426
UByte4, 426
ArTypes, 426
ArUtil
BIT0, 429
BIT1, 429
BIT10, 429
BIT11, 429
BIT12, 429
BIT13, 429
BIT14, 429
BIT15, 429
BIT2, 429
BIT3, 429
BIT4, 429
BIT5, 429
BIT6, 429
BIT7, 429
BIT8, 429
BIT9, 429
REGKEY_CLASSES_ROOT, 429
REGKEY_CURRENT_CONFIG, 429
REGKEY_CURRENT_USER, 429
REGKEY_LOCAL_MACHINE, 429
REGKEY_USERS, 429
ArUtil, 427
appendSlash, 430
BITS, 429
deleteSet, 430
deleteSetPairs, 430
escapeSpaces, 430
findFile, 430
fixSlashes, 431
fixSlashesBackward, 431
fixSlashesForward, 431
getStringFromFile, 431
getStringFromRegistry, 431
getTime, 432
REGKEY_CLASSES_ROOT, 429
REGKEY_CURRENT_CONFIG, 429
REGKEY_CURRENT_USER, 429
REGKEY_LOCAL_MACHINE, 429
REGKEY_USERS, 429
ArUtil, 427
INDEX

strcmp, 433, 434
stripDir, 435
stripFile, 435
ArVCC4
ArVCC4, 438
canZoom, 437
connectHandler, 436
getMaxNegPan, 437
getMaxNegTilt, 437
getMaxPanSlew, 438
getMaxPosPan, 436
getMaxPosTilt, 437
getMaxTiltSlew, 438
getMaxZoom, 438
getMinPanSlew, 438
getMinTiltSlew, 438
getMinZoom, 438
getPan, 437
getPanSlew, 438
getTilt, 437
getTiltSlew, 438
getZoom, 437
haltPanTilt, 437
haltZoom, 437
init, 436
pan, 436
panRel, 436
panSlew, 437
panTilt, 437
panTiltRel, 436
power, 436
preparePacket, 437
tilt, 436
tiltRel, 436
tiltSlew, 437
zoom, 437
ArVCC4, 436
ArVCC4, 438
ArVCC4Commands
CONTROL, 441
DELIM, 440
DEVICEID, 440
FOOTER, 441
HEADER, 441
INIT, 440
PANSLEW, 440
PANTILT, 440
POWER, 441
SETRANGE, 441
STOP, 440
TILTSLEW, 440
ZOOM, 441
ZOOMSTOP, 441
ArVCC4Packet
ArVCC4Packet, 442
ArVCC4Packet, 442
ArVCC4Packet, 442
asyncConnect
ArRobot, 310
ArSick, 371
asyncConnectHandler
ArRobot, 311
atan2
ArMath, 203
attachKeyHandler
ArRobot, 311
awaitExec
ArDPPTU, 126
BACKSPACE
ArKeyHandler, 196
BASE
ArDPPTUCommands, 130
baseActivate
ArMode, 209
baseDeactivate
ArMode, 209
baseHelp
ArMode, 208
basePanSlew
ArDPPTU, 127
baseTiltSlew
ArDPPTU, 127
BAUD19200
ArSick, 368
BAUD38400
ArSick, 369
BAUD9600
ArSick, 368
BaudRate
ArSick, 368  
beginInvalidationSweep  
ArRangeBuffer, 252  
beginRedoBuffer  
ArRangeBuffer, 252  
BIT0  
ArUtil, 429  
BIT1  
ArUtil, 429  
BIT10  
ArUtil, 429  
BIT11  
ArUtil, 429  
BIT12  
ArUtil, 429  
BIT13  
ArUtil, 429  
BIT14  
ArUtil, 429  
BIT15  
ArUtil, 429  
BIT2  
ArUtil, 429  
BIT3  
ArUtil, 429  
BIT4  
ArUtil, 429  
BIT5  
ArUtil, 429  
BIT6  
ArUtil, 429  
BIT7  
ArUtil, 429  
BIT8  
ArUtil, 429  
BIT9  
ArUtil, 429  
BITS  
ArUtil, 429  
blank  
ArDPPTU, 124  
BLOB_DATA_SIZE  
ArACTS, 1, 2, 86  
block  
ArSignalHandler, 386  
blockCommon  
ArSignalHandler, 386  
blockCommonThisThread  
ArSignalHandler, 387  
blockingConnect  
ArRobot, 311  
ArSick, 372  
BOOL  
ArArg, 101  
Boolean  
ArPref, 238  
broadcast  
ArCondition, 116  
bufToByte  
ArBasePacket, 108  
bufToByte2  
ArBasePacket, 108  
bufToByte4  
ArBasePacket, 108  
bufToData  
ArBasePacket, 110  
bufToStr  
ArBasePacket, 110  
bufToUByte  
ArBasePacket, 108  
bufToUByte2  
ArBasePacket, 108  
bufToUByte4  
ArBasePacket, 108  
BUMPSTALL  
ArCommands, 114  
Byte  
ArTypes, 426  
Byte2  
ArTypes, 426  
byte2ToBuf  
ArAMPTUPacket, 97  
ArBasePacket, 107  
ArSonyPacket, 398  
byte2ToBufAtPos  
ArSonyPacket, 398  
Byte4  
ArTypes, 426  
byte4ToBuf  
ArBasePacket, 107  
byteToBuf  
ArAMPTUPacket, 97
ArAMPTUCommands, 95
CONTROL
  ArDPPTUCommands, 130
  ArVCC4Commands, 441
copy
  ArSocket, 390, 394
cos
  ArMath, 203
create
  ArASyncTask, 105
  ArSocket, 390
  ArThread, 417
createHandlerNonThreaded
  ArSignalHandler, 387
createHandlerThreaded
  ArSignalHandler, 387
cumulativeReadingBox
  ArRangeDevice, 259
cumulativeReadingPolar
  ArRangeDevice, 259
currentReadingBox
  ArRangeDevice, 260
currentReadingPolar
  ArRangeDevice, 260

DATA_HEADER
  ArACTS, 1, 2, 86
dataToBuf
  ArBasePacket, 111
DCHEAD
  ArCommands, 114
deactivate
  ArAction, 39
  ArActionGroup, 58
  ArActionKeydrive, 70
  ArMode, 208
  ArModeCamera, 211
  ArModeGripper, 213
  ArModeSonar, 215
  ArModeTeleop, 217
  ArModeWander, 219
Degrees
  ArSick, 369
DEGREES100
  ArSick, 369
DEGREES180
  ArSick, 369
degToRad
  ArMath, 204
deleteSet
  ArUtil, 430
deleteSetPairs
  ArUtil, 430
delHandlerCB
  ArSignalHandler, 387
DELIM
  ArDPPTUCommands, 130
  ArVCC4Commands, 440
delRobot
  Aria, 181
deltaHeading
  ArActionInput, 65
deltaVel
  ArActionInput, 65
detach
  ArThread, 418
DEVICEID
  ArVCC4Commands, 440
DHEAD
  ArCommands, 114
didAll
  ArSectors, 352
DIGOUT
  ArCommands, 114
DISABLE
  ArDPPTUCommands, 130
disableMotors
  ArRobot, 316
disableReset
  ArDPPTU, 124
disconnect
  ArRobot, 316
  ArSick, 372
disMon
  ArDPPTU, 126
doInvTransform
  ArTransform, 424
done
  ArRecurrentTask, 268
doTransform
  ArTransform, 423, 424
DOUBLE
getIndex 473

getClass
  ArRobotParams, 349
getCloseDist
  ArActionGoto, 56
getClosestBox
  ArRangeBuffer, 253
getClosestPolar
  ArRangeBuffer, 254
getClosestSonarNumber
  ArRobot, 297
getClosestSonarRange
  ArRobot, 296
getCompass
  ArRobot, 295
getConnectionTimeoutTime
  ArRobot, 318
  ArSick, 373
getControl
  ArRobot, 318
getCounter
  ArRobot, 302
getCounterTaken
  ArSensorReading, 355
getCumulativeBuffer
  ArRangeDevice, 257
getCumulativeRangeBuffer
  ArRangeDevice, 257
getCurrentBuffer
  ArRangeDevice, 257
getCurrentRangeBuffer
  ArRangeDevice, 257
gency
cycleTime
  ArRobot, 319
getData
  ArACTS, 86
dataDiff
  ArSickLogger, 376
dataDegrees
  ArSick, 367
dataDeltaHeading
  ArActionDesired, 50
dataDeltaHeadingDesiredChannel
  ArActionDesired, 51
dataDeltaHeadingStrength
  ArActionDesired, 50
dataDescription
  ArAction, 39
  ArArg, 100
  ArResolver, 269
dataDesired
  ArAction, 39
  ArActionAvoidFront, 41
  ArActionAvoidSide, 43
  ArActionBumpers, 45
  ArActionConstantVelocity, 47
  ArActionGoto, 57
  ArActionInput, 65
  ArActionJoydrive, 68
  ArActionKeydrive, 70
  ArActionLimiterBackwards, 73
  ArActionLimiterForwards, 75
  ArActionLimiterTableSensor, 77
  ArActionStallRecover, 79
  ArActionStop, 81
  ArActionTurn, 83
dataDeviceConnection
  ArPTZ, 246
  ArRobot, 319
  ArRobotPacketReceiver, 342
  ArRobotPacketSender, 345
  ArSick, 365
  ArSickPacketReceiver, 381
dataDiffConvFactor
  ArRobotParams, 350
dataDigIn
  ArRobot, 295
dataDigOut
  ArRobot, 295
dataDirectMotionPrecedenceTime
  ArRobot, 319
dataDirectory
  ArAct, 183
  ArActionLimiterTableSensor
  ArActionLimiterTableSensor
  ArArgument
  ArArg, 100
  ArPuff
  ArPref, 238
dataDoubles

ArJoyHandler, 192
getDoubleSet
    ArPref, 239
getEncoderCorrectionCallback
    ArRobot, 320
getEncoderPose
    ArRobot, 296
getAddress
    ArCondition, 116
    ArMutex, 227
    ArSocket, 392
    ArThread, 418
getAddressStr
    ArSocket, 392
getFD
    ArSocket, 392
getFilterNearDist
    ArSick, 367
getFlags
    ArRobot, 294
getFunc
    ArThread, 418
getFunctor
    ArSyncTask, 404
getGoal
    ArActionGoto, 56
getGraspTime
    ArGripper, 175
getGripState
    ArGripper, 175
getHandler
    ArSignalHandler, 388
getHardwareControl
    ArSerialConnection, 360
defHeaderLength
    ArBasePacket, 109
defHeading
    ArActionDesired, 50
defHeadingStrength
    ArActionDesired, 50
defHost
    ArTcpConnection, 413
defHostName
    ArSocket, 392
defID
    ArRobotPacket, 340
    ArSickPacket, 378
defIncrement
    ArSick, 367
defInt
    ArArg, 100
    ArPref, 239
defIntSet
    ArPref, 239
defIOAnalog
    ArRobot, 295
defIOAnalogSize
    ArRobot, 295
defIODigIn
    ArRobot, 295
defIODigInSize
    ArRobot, 295
defIODigOut
    ArRobot, 295
defIODigOutSize
    ArRobot, 295
defIOPacketTime
    ArRobot, 296
defJoinable
    ArThread, 418
defKey
    ArKeyHandler, 195
    ArMode, 208
defKey2
    ArMode, 208
defKeyHandler
    Aria, 182
    ArRobot, 302
defLaserFlipped
    ArRobotParams, 351
defLaserPort
    ArRobotParams, 351
defLaserPossessed
    ArRobotParams, 351
defLaserPowerControlled
    ArRobotParams, 351
defLaserX
    ArRobotParams, 351
defLaserY
    ArRobotParams, 351
defLastPacketTime
    ArRobot, 320
getLastErrorTime
  ArSick, 366
getLeft
  ArACTSBlob, 90
getLeftVel
  ArRobot, 294
getLength
  ArBasePacket, 109
getMaxLength
  ArBasePacket, 109
getMaxNegPan
  ArAMPTU, 93
  ArDPPTU, 126
  ArPTZ, 245
  ArSonyPTZ, 401
  ArVCC4, 437
getMaxNegTilt
  ArAMPTU, 93
  ArDPPTU, 126
  ArPTZ, 245
  ArSonyPTZ, 401
  ArVCC4, 437
getMaxNegVel
  ArActionDesired, 50
getMaxNegVelDesiredChannel
  ArActionDesired, 51
getMaxNegVelStrength
  ArActionDesired, 50
getMaxPosPan
  ArAMPTU, 93
  ArDPPTU, 126
  ArPTZ, 245
  ArSonyPTZ, 401
  ArVCC4, 437
getMaxPosTilt
  ArAMPTU, 93
  ArDPPTU, 126
  ArPTZ, 245
  ArSonyPTZ, 401
  ArVCC4, 437
getMaxPosVel
  ArActionDesired, 50
getMaxPosVelDesiredChannel
  ArActionDesired, 51
getMaxPosVelStrength
  ArActionDesired, 50
getMaxRange
  ArRangeDevice, 257
getMaxRotVel
  ArActionDesired, 50
  ArRobot, 293
getMaxRotVelDesiredChannel
  ArActionDesired, 51
getMaxRotVelocity
  ArRobot, 293
getMaxRotVelStrength
  ArActionDesired, 50
getMaxTransVel
  ArRobot, 293
getMaxVel
  ArRobotParams, 349
getMaxVelDesiredChannel
  ArActionDesired, 51
getMaxVelocity
  ArRobotParams, 349
getMaxZoom
  ArPTZ, 246
  ArSonyPTZ, 401
  ArVCC4, 438
getMinPanSlew
  ArVCC4, 438
getMinTiltSlew
  ArVCC4, 438
getMinZoom
  ArPTZ, 246
  ArSonyPTZ, 401
  ArVCC4, 438
getMotorPacCount
  ArRobot, 296
getMoveDoneDist
  ArRobot, 292
getMSec
  ArTime, 422
getMSecSinceLastPacket
  ArGripper, 175
getMutex
  ArMutex, 227
getName
  ArAction, 39
  ArArg, 99
  ArMode, 207
  ArRangeDevice, 256
ArResolver, 269
ArRobot, 297
ArSyncTask, 404
getNumArgs
ArAction, 39
getNumOfReadings
ArInterpolation, 186
getNumBlobs
ArACTS, 1, 2, 87
getNumSonar
ArRobot, 296
ArRobotParams, 350
getOpenMessage
ArDeviceConnection, 120
ArSerialConnection, 360
ArTcpConnection, 413
getPaddleState
ArGripper, 175
getPan
ArAMPTU, 93
ArDPPTU, 128
ArPTZ, 245
ArSonyPTZ, 401
ArVCC4, 437
getPanAccel
ArDPPTU, 128
getPanSlew
ArDPPTU, 128
ArVCC4, 438
getParams
ArRobot, 296
getPool
ArTaskPool, 408
getPort
ArTcpConnection, 413
getPose
ArArg, 100
ArInterpolation, 187
ArPose, 233
ArRobot, 293
ArSensorReading, 353
getPoseInterpNumReadings
ArRobot, 302
getPoseInterpPosition
ArRobot, 320
getPosTaken
ArRangeBuffer, 250
ArSensorReading, 354
getRange
ArSensorReading, 355
getRangeConvFactor
ArRobotParams, 350
getRangeDeviceList
ArRobot, 320
getRawReadings
ArRangeDevice, 261
getReadLength
ArBasePacket, 109
getRealPan
ArPTZ, 245
getRealTilt
ArPTZ, 245
getRealZoom
ArPTZ, 245
getReceivedAddress
ArSickPacket, 379
getRequestIOPackets
ArRobotParams, 349
getResolver
ArRobot, 301
getRight
ArACTSBlob, 90
getRightVel
ArRobot, 294
getRobot
ArACTS, 1, 2, 85
ArModule, 221
ArRangeDevice, 256
getRobotDiagonal
ArRobot, 294
ArRobotParams, 349
getRobotList
Aria, 182
getRobotName
ArRobot, 293
getRobotParams
ArRobot, 321
getRobotRadius
ArRobot, 294
ArRobotParams, 349
getRobotSubType
ArRobot, 293
getRobotType
    ArRobot, 293
getRotVel
    ArRobot, 294
getRunExitListCopy
    ArRobot, 304
getRunning
    ArThread, 418
getRunningWithLock
    ArThread, 418
getSec
    ArTime, 422
getSendingAddress
    ArSickPacket, 380
getSensorDX
    ArSensorReading, 353
getSensorDY
    ArSensorReading, 354
getSensorPosition
    ArSensorReading, 355
    ArSick, 365
getSensorPositionTh
    ArSick, 365
getSensorPositionX
    ArSick, 365
getSensorPositionY
    ArSick, 365
getSensorTh
    ArSensorReading, 353
getSensorX
    ArSensorReading, 353
getSensorY
    ArSensorReading, 353
getSetCount
    ArPref, 239
getSonarPacCount
    ArRobot, 296
getSonarReading
    ArRobot, 321
getSonarTh
    ArRobotParams, 351
getSonarX
    ArRobotParams, 351
getSonarY
    ArRobotParams, 351
getSpeed
    ArActionGoto, 56
getSpeeds
    ArJoyHandler, 191
getStallValue
    ArRobot, 294
getAddress
    ArSyncTask, 403
getState
    ArSyncTask, 403
getStats
    ArJoyHandler, 191
getStatsMessage
    ArDeviceConnection, 120
    ArSerialConnection, 360
    ArTcpConnection, 414
getString
    ArArg, 100
    ArPref, 240
ggetStringFromFile
    ArUtil, 431
ggetStringFromRegistry
    ArUtil, 431
ggetStringSet
    ArPref, 240
gerSubClass
    ArRobotParams, 349
getSyncTaskRoot
    ArRobot, 322
gerTh
    ArPose, 232
    ArRobot, 293
    ArTransform, 423
getThRad
   ArPose, 232
getThread
   ArThread, 418
getThTaken
   ArSensorReading, 354
getTilt
   ArAMPTU, 93
   ArDPPTU, 128
   ArPTZ, 245
   ArSonyPTZ, 401
   ArVCC4, 437
getTiltAccel
   ArDPPTU, 128
getTiltSlew
   ArDPPTU, 128
   ArVCC4, 438
getTime
   ArUtil, 432
ggetTimeRead
   ArDeviceConnection, 121
   ArSerialConnection, 361
   ArTcpConnection, 414
ggetTimeReceived
   ArRobotPacket, 340
   ArSickPacket, 379
goToGlobalTransform
   ArRobot, 322
goToLocalTransform
   ArRobot, 322
goToTop
   ArACTSBlob, 90
getType
   ArArg, 101
   ArGripper, 176
   ArSocket, 392
ggetUnfiltered
   ArJoyHandler, 193
ggetUnitNumber
   ArAMPTUPacket, 98
ggetUseOSCal
   ArActionJoydrive, 69
   ArJoyHandler, 193
ggetVel
   ArActionDesired, 49
   ArRobot, 294
ggetVel2Divisor
   ArRobotParams, 350
ggetVelConvFactor
   ArRobotParams, 350
ggetVelDesiredChannel
   ArActionDesired, 51
ggetVelStrength
   ArActionDesired, 50
ggetX
   ArPose, 232
   ArRobot, 293
   ArSensorReading, 353
ggetXCG
   ArACTSBlob, 90
ggetXTaken
   ArSensorReading, 354
ggetY
   ArPose, 232
   ArRobot, 293
   ArSensorReading, 353
ggetYCG
   ArACTSBlob, 90
ggetYTaken
   ArSensorReading, 354
ggetZoom
   ArPTZ, 245
   ArSonyPTZ, 401
   ArVCC4, 437
ggiveUpKeys
   ArActionKeydrive, 71
   ArRecurrentTask, 267
GRIP_CLOSE
   ArGripperCommands, 179
GRIP_OPEN
   ArGripperCommands, 179
GRIP_PRESSURE
   ArGripperCommands, 179
GRIP_STOP
   ArGripperCommands, 179
gripClose
   ArGripper, 176
gripOpen
   ArGripper, 176
GRIPPAC
   ArGripper, 174
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRIPPER</td>
<td>ArCommands, 114</td>
<td>114</td>
</tr>
<tr>
<td>GRIPPER_DEPLOY</td>
<td>ArGripperCommands, 179</td>
<td>179</td>
</tr>
<tr>
<td>GRIPPER_HALT</td>
<td>ArGripperCommands, 179</td>
<td>179</td>
</tr>
<tr>
<td>GRIPPER_STORE</td>
<td>ArGripperCommands, 179</td>
<td>179</td>
</tr>
<tr>
<td>gripperDeploy</td>
<td>ArGripper, 176</td>
<td></td>
</tr>
<tr>
<td>gripperHalt</td>
<td>ArGripper, 176</td>
<td></td>
</tr>
<tr>
<td>GRIPPERPACREQUEST</td>
<td>ArCommands, 114</td>
<td>114</td>
</tr>
<tr>
<td>gripperStore</td>
<td>ArGripper, 176</td>
<td></td>
</tr>
<tr>
<td>GRIPPERVAL</td>
<td>ArCommands, 114</td>
<td>114</td>
</tr>
<tr>
<td>gripPressure</td>
<td>ArGripper, 177</td>
<td></td>
</tr>
<tr>
<td>gripStop</td>
<td>ArGripper, 177</td>
<td></td>
</tr>
<tr>
<td>HALT</td>
<td>ArDPPTUCommands, 130</td>
<td>130</td>
</tr>
<tr>
<td>haltAll</td>
<td>ArDPPTU, 126</td>
<td>126</td>
</tr>
<tr>
<td>haltPan</td>
<td>ArDPPTU, 126</td>
<td>126</td>
</tr>
<tr>
<td>haltPanTilt</td>
<td>ArVCC4, 437</td>
<td>437</td>
</tr>
<tr>
<td>haltTilt</td>
<td>ArDPPTU, 126</td>
<td>126</td>
</tr>
<tr>
<td>haltZoom</td>
<td>ArVCC4, 437</td>
<td>437</td>
</tr>
<tr>
<td>handle</td>
<td>ArSignalHandler, 388</td>
<td></td>
</tr>
<tr>
<td>handlePacket</td>
<td>ArRobot, 304</td>
<td>304</td>
</tr>
<tr>
<td>hasFrontBumpers</td>
<td>ArRobot, 296</td>
<td>296</td>
</tr>
<tr>
<td>hasMoveCommand</td>
<td>ArRobotParams, 349</td>
<td>349</td>
</tr>
<tr>
<td>hasRangeDevice</td>
<td>ArRobot, 322</td>
<td>322</td>
</tr>
<tr>
<td>hasRearBumpers</td>
<td>ArRobot, 295</td>
<td>295</td>
</tr>
<tr>
<td>hasTableSensingIR</td>
<td>ArRobot, 295</td>
<td>295</td>
</tr>
<tr>
<td>haveAchievedGoal</td>
<td>ArActionGoto, 56</td>
<td>56</td>
</tr>
<tr>
<td>haveFrontBumpers</td>
<td>ArRobotParams, 350</td>
<td>350</td>
</tr>
<tr>
<td>haveJoystick</td>
<td>ArJoyHandler, 190</td>
<td>190</td>
</tr>
<tr>
<td>haveNewTableSensingIR</td>
<td>ArRobotParams, 350</td>
<td>350</td>
</tr>
<tr>
<td>haveRearBumpers</td>
<td>ArRobotParams, 350</td>
<td>350</td>
</tr>
<tr>
<td>haveSonar</td>
<td>ArRobotParams, 351</td>
<td>351</td>
</tr>
<tr>
<td>haveTableSensingIR</td>
<td>ArRobotParams, 350</td>
<td>350</td>
</tr>
<tr>
<td>HEAD</td>
<td>ArCommands, 114</td>
<td>114</td>
</tr>
<tr>
<td>HEADER</td>
<td>ArVCC4Commands, 441</td>
<td>441</td>
</tr>
<tr>
<td>help</td>
<td>ArMode, 209</td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>ArModeCamera, 212</td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>ArModeGripper, 214</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>ArModeSonar, 216</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>ArModeTeleop, 217</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>ArModeWander, 219</td>
<td>219</td>
</tr>
<tr>
<td>highMotPower</td>
<td>ArDPPTU, 127</td>
<td>127</td>
</tr>
<tr>
<td>hostAddr</td>
<td>ArSocket, 392</td>
<td>392</td>
</tr>
<tr>
<td>hostToNetOrder</td>
<td>ArSocket, 393</td>
<td>393</td>
</tr>
<tr>
<td>IMMED</td>
<td>ArDPPTUCommands, 131</td>
<td>131</td>
</tr>
<tr>
<td>immedExec</td>
<td>ArDPPTU, 125</td>
<td>125</td>
</tr>
<tr>
<td>inAddr</td>
<td>ArSocket, 391</td>
<td>391</td>
</tr>
<tr>
<td>incCounter</td>
<td>ArRobot, 303</td>
<td>303</td>
</tr>
<tr>
<td>Increment</td>
<td>ArSick, 369</td>
<td>369</td>
</tr>
<tr>
<td>INCREMENT_HALF</td>
<td>ArRobot, 296</td>
<td>296</td>
</tr>
</tbody>
</table>
INDEX

ArSick, 369
INCREMENT.ONE
  ArSick, 369
indepMove
  ArDPPTU, 127
INIT
  ArAMPTUCommands, 96
  ArDPPTUCommands, 130
  ArTaskState, 410
  ArVCC4Commands, 440
init
  ArAMPTU, 92
  ArDPPTU, 124
  Aria, 183
  ArJoyHandler, 190
  ArLog, 201
  ArModule, 222
  ArPTZ, 244
  ArRobot, 323
  ArRobotParams, 349
  ArSocket, 394
  ArSonyPTZ, 400
  ArTaskPool, 408
  ArThread, 420
  ArVCC4, 436
initMon
  ArDPPTU, 126
inPort
  ArSocket, 391
INT
  ArArg, 101
Integer
  ArPref, 238
internalConnectHandler
    ArSick, 373
internalConnectSim
    ArSick, 373
internalOpen
  ArTcpConnection, 412
inToA
  ArSocket, 392
INVALID
  ArArg, 101
invalidateReading
  ArRangeBuffer, 254
invert

ArACTS.1.2, 87
invoke
  ArFunctor, 133
  ArFunctor1, 135, 136
  ArFunctor1C, 137, 139
  ArFunctor2, 141, 142
  ArFunctor2C, 144, 147
  ArFunctorC, 150
  ArGlobalFunctor, 153
  ArGlobalFunctor1, 155, 156
  ArGlobalFunctor2, 158, 160
  ArPeriodicTask, 229
  ArRetFunctor, 271
invokeR
  ArGlobalRetFunctor, 163
  ArGlobalRetFunctor1, 165, 166
  ArGlobalRetFunctor2, 168, 170
  ArRetFunctor, 271
  ArRetFunctor1, 273, 274
  ArRetFunctor1C, 275, 277
  ArRetFunctor2, 279, 280
  ArRetFunctor2C, 282, 284
  ArRetFunctorC, 287
IOREQUEST
  ArCommands, 114
isActive
  ArAction, 38
isAfter
  ArTime, 421
isAllocatingPackets
  ArRobotPacketReceiver, 342
  ArSickPacketReceiver, 381
isAt
  ArTime, 421
isBefore
  ArTime, 421
isConnected
  ArACTS.1.2, 85
  ArRobot, 323
  ArSick, 365
isControllingPower
  ArSick, 367
isCycleChained
  ArRobot, 302
isDirectMotion
ArRobot, 323
isGripMoving
ArGripper, 177
isHeadingDone
ArRobot, 323
isHolonomic
ArRobotParams, 349
isLaserFlipped
ArSick, 367
isLeftBreakBeamTriggered
ArRobot, 296
isLeftMotorStalled
ArRobot, 294
isLeftTableSensingIRTriggered
ArRobot, 295
isLiftMaxed
ArGripper, 177
isLiftMoving
ArGripper, 177
isMoveDone
ArRobot, 324
isNew
ArSensorReading, 356
isRightBreakBeamTriggered
ArRobot, 296
isRightMotorStalled
ArRobot, 294
isRightTableSensingIRTriggered
ArRobot, 295
isRunning
ArPeriodicTask, 229
ArRobot, 324
isSonarNew
ArRobot, 324
isTimeStamping
ArDeviceConnection, 121
ArSerialConnection, 361
ArTcpConnection, 414
isUsingSim
ArSick, 367

join
ArThread, 418
joinAll
ArThread, 419

JOYDRIVE
ArCommands, 114
joystickInited
ArActionJoydrive, 67

KEY
ArKeyHandler, 196
keyHandlerExit
ArRobot, 303

LAST
ArListPos, 199
LEFT
ArKeyHandler, 196
left
ArActionKeydrive, 71
LIFT_CARRY
ArGripperCommands, 180
LIFT_DOWN
ArGripperCommands, 179
LIFT_STOP
ArGripperCommands, 179
LIFT_UP
ArGripperCommands, 179
lifCarry
ArGripper, 177
liftDown
ArGripper, 178
liftStop
ArGripper, 178
liftUp
ArGripper, 178
LIMIT
ArDPPTUCmds, 131
limitEnforce
ArDPPTU, 125
load
ArModuleLoader, 225
LOADPARAM
ArCommands, 114
loadParamFile
ArRobot, 324
LOADWORLD
ArCommands, 114
lock
ArMutex, 228
ArRobot, 302
ArThread, 418

lockDevice
   ArRangeDevice, 261
   ArRangeDeviceThreaded, 265

log
   ArLog, 201
   LogLevel
   ArLog, 200
   LogType
   ArLog, 200

loopOnce
   ArRobot, 325

lowerPanSlew
   ArDPPTU, 127

lowerTiltSlew
   ArDPPTU, 127

lowMotPower
   ArDPPTU, 127

lowStatPower
   ArDPPTU, 127

madeConnection
   ArRobot, 304
   ArSick, 368

MAX_BLOBS
   ArACTS, 1, 2, 86

MAX_DATA
   ArACTS, 1, 2, 86

MAX_PAN
   ArSonyPTZ, 402

MAX_PAN_ACCEL
   ArDPPTU, 129

MAX_PAN_SLEW
   ArDPPTU, 129

MAX_STRENGTH
   ArActionDesiredChannel, 55

MAX_TILT
   ArDPPTU, 129
   ArSonyPTZ, 402

MAX_TILT_ACCEL
   ArDPPTU, 129

MAX_TILT_SLEW
   ArDPPTU, 129

MAX_ZOOM
   ArSonyPTZ, 402

maxHostNameLen
   ArSocket, 393

merge
   ArActionDesired, 52

MIN_PAN
   ArDPPTU, 129
   MIN_PAN_ACCEL
   ArDPPTU, 129
   MIN_PAN_SLEW
   ArDPPTU, 129

MIN_STRENGTH
   ArActionDesiredChannel, 55

MIN_TILT
   ArDPPTU, 129
   MIN_TILT_ACCEL
   ArDPPTU, 129
   MIN_TILT_SLEW
   ArDPPTU, 129

MIN_ZOOM
   ArSonyPTZ, 402

MONITOR
   ArDPPTUCommands, 131

MOVE
   ArCommands, 113

move
   ArRobot, 325

moveTo
   ArRobot, 325

mSecSince
   ArTime, 421

mSecTo
   ArTime, 421

myRobot
   ArModule, 221

myRunning
   ArThread, 419

nameSignal
   ArSignalHandler, 385

netToHostOrder
   ArSocket, 393

newData
   ArSensorReading, 356

NO_STRENGTH
   ArActionDesiredChannel, 55

NOGRIPPER
ArGripper, 174
None
ArLog, 201
Normal
ArLog, 200
NUM_CHANNELS
ArACTS_1_2, 86
numFrontBumpers
ArRobotParams, 350
numRearBumpers
ArRobotParams, 350
OFFSET
ArDPPTUCommands, 131
offStatPower
ArDPPTU, 126
OPEN
ArCommands, 113
Open
ArSerialConnection, 359
ArTcpConnection, 412
open
ArSerialConnection, 361
ArSocket, 390
ArTcpConnection, 414
OPEN_BAD_ALREADY_OPEN
ArSerialConnection, 360
OPEN_BAD_HOST
ArTcpConnection, 412
OPEN_CON_REFUSED
ArTcpConnection, 413
OPEN_COULD_NOT_OPEN-_PORT
ArSerialConnection, 359
OPEN_COULD_NOT_SET_BAUD
ArSerialConnection, 359
OPEN_COULD_NOT_SET_UP-_PORT
ArSerialConnection, 359
OPEN_INVALID_BAUD_RATE
ArSerialConnection, 359
OPEN_NET_FAIL
ArTcpConnection, 412
OPEN_NO_ROUTE
ArTcpConnection, 413
openPort
ArACTS_1_2, 87
openSimple
ArDeviceConnection, 119
ArSerialConnection, 358
ArTcpConnection, 411
operator()
ArFunctor, 133
ArFunctor1, 135, 136
ArFunctor1C, 137, 139
ArFunctor2, 141, 142
ArFunctor2C, 145, 148
ArFunctorC, 150
ArGlobalFunctor, 153
ArGlobalFunctor1, 155, 157
ArGlobalFunctor2, 159, 161
ArRetFunctor, 271
ourPool
ArTaskPool, 409
packetHandler
ArGripper, 174
ArIrrfDevice, 189
ArPTZ, 247
ArRobot, 326
PAN
ArDPPTUCommands, 131
pan
ArAMPTU, 92
ArDPPTU, 125
ArPTZ, 244
ArSonyPTZ, 400
ArVCC4, 436
panAccel
ArAMPTU, 92
ArDPPTU, 125
ArPTZ, 244
ArSonyPTZ, 400
ArVCC4, 436
PANSLEW
ArAMPTUCommands, 96
ArVCC4Commands, 440
panSlew
ArAMPTU, 93
ArDPPTU, 128
INDEX

ArMath, 204
read
   ArDeviceConnection, 121
   ArSerialConnection, 362
   ArSocket, 394
   ArTcpConnection, 415
readPacket
   ArPTZ, 248
receiveBlobInfo
   ArACTS_1_2, 88
receivePacket
   ArRobotPacketReceiver, 343
   ArSickPacketReceiver, 382
recvFrom
   ArSocket, 391
redoReading
   ArRangeBuffer, 255
REGKEY
   ArUtil, 429
REGKEY_CLASSES_ROOT
   ArUtil, 429
REGKEY_CURRENT_CONFIG
   ArUtil, 429
REGKEY_CURRENT_USER
   ArUtil, 429
REGKEY_LOCAL_MACHINE
   ArUtil, 429
REGKEY_USERS
   ArUtil, 429
regMotPower
   ArDPPTU, 127
regStatPower
   ArDPPTU, 126
reload
   ArModuleLoader, 226
RELPANCCW
   ArAMPTUCommands, 95
RELPANCW
   ArAMPTUCommands, 95
RELTLTD
   ArAMPTUCommands, 95
RELTLTU
   ArAMPTUCommands, 95
remAction
   ArActionGroup, 59
   ArRobot, 326, 327
remConnectCB
   ArRobot, 327
   ArSick, 374
remDisconnectNormallyCB
   ArRobot, 327
   ArSick, 374
remDisconnectOnErrorCB
   ArRobot, 327
   ArSick, 374
remFailedConnectCB
   ArRobot, 328
   ArSick, 374
remKeyHandler
   ArKeyHandler, 197
removeActions
   ArActionGroup, 58
remPacketHandler
   ArRobot, 328
remRangeDevice
   ArRobot, 328
remRunExitCB
   ArRobot, 328
remSensorInterpTask
   ArRobot, 329
remUserTask
   ArRobot, 329
requestPacket
   ArACTS_1_2, 88
requestQuit
   ArACTS_1_2, 88
requestStatus
   ArAMPTU, 93
RESET
   ArDPPTUCommands, 131
reset
   ArActionDesired, 49
   ArInterpolation, 186
   ArRangeBuffer, 251
   ArRecurrentTask, 267
resetAll
   ArDPPTU, 125
resetCalib
   ArDPPTU, 124
resetPan
   ArDPPTU, 125
resetRead
ArBasePacket, 111
ArRobotPacket, 341
ArSickPacket, 380
resetSensorPosition
ArSensorReading, 356
RESETSIMTOORIGIN
ArCommands, 115
resetTilt
ArDPPTU, 125
resolve
ArResolver, 269
RESP
ArAMPTUCommands, 96
restore
ArKeyHandler, 195
restoreSet
ArDPPTU, 125
RESUME
ArTaskState, 410
resume
ArAMPTU, 93
RIGHT
ArKeyHandler, 196
right
ArActionKeydrive, 71
robotConnectCallback
ArSick, 368
robotLocker
ArRobot, 329
robotPacketHandler
ArPTZ, 248
robotTask
ArSickLogger, 376
robotUnlocker
ArRobot, 330
ROTATE
ArCommands, 114
roundInt
ArMath, 205
run
ArRangeDeviceThreaded, 265
ArRobot, 330
ArSyncTask, 407
runAsync
ArRangeDeviceThreaded, 265
ArRobot, 330
runInThisThread
ArASyncTask, 106
runOnce
ArSick, 367
runOnRobot
ArSick, 375
runTask
ArPeriodicTask, 229
ArTaskPool, 408
runThread
ArASyncTask, 106
ArRangeDeviceThreaded, 265
ArRecurrentTask, 268
ArSick, 367
ArSignalHandler, 388
RVEL
ArCommands, 114
saveSet
ArDPPTU, 125
SAY
ArCommands, 114
secSince
ArTime, 421
secTo
ArTime, 421
self
ArThread, 420
sendPacket
ArPTZ, 248
sendTo
ArSocket, 391
sensorInterpCallback
ArSick, 367
sensorInterpHandler
ArPTZ, 246
SETA
ArCommands, 113
setArea
ArACTSBlob, 90
setBaud
ArSerialConnection, 362
setBool
ArArg, 100
ArPref, 240
setBoolSet
INDEX

ArPref, 240
setBottom
ArACTSBlob, 91
setBroadcast
ArSocket, 391
setBuf
ArBasePacket, 109
setCloseDist
ArActionGoto, 56
setConnectionTimeoutTime
ArRobot, 330
ArSick, 375
setCumulativeBufferSize
ArRangeDevice, 261
setCumulativeMaxRange
ArIrrfDevice, 188
ArSonarDevice, 396
setCurrentBufferSize
ArRangeDevice, 262
setCycleChained
ArRobot, 302
setCycleTime
ArRobot, 331
setDeadRecoPose
ArRobot, 331
setDegDiff
ArSickLogger, 376
setDeltaHeading
ArActionDesired, 52
ArRobot, 331
setDeviceConnection
ArPTZ, 249
ArRobot, 331
ArRobotPacketReceiver, 342
ArRobotPacketSender, 345
ArSick, 365
ArSickPacketReceiver, 381
setDirectMotionPrecedenceTime
ArRobot, 332
setDirectory
Aria, 184
setDistDiff
ArSickLogger, 376
setDoClose
ArSocket, 392
setDouble
ArArg, 100
ArPref, 241
setDoubleSet
ArPref, 241
setEncoderCorrectionCallback
ArRobot, 332
setEncoderTransform
ArRobot, 333
setFilterNearDist
ArSick, 366
setGoal
ArActionGoto, 56
setHardwareControl
ArSerialConnection, 362
setHeaderLength
ArBasePacket, 109
setHeading
ArActionDesired, 53
ArRobot, 333
setHeadingDoneDiff
ArRobot, 292
setIncrements
ArActionKeydrive, 70
setInt
ArArg, 100
ArPref, 241
setIntSet
ArPref, 242
setKeyHandler
Aria, 182
setLeft
ArACTSBlob, 91
setLength
ArBasePacket, 109
setLinger
ArSocket, 391
setMaxNegVel
ArActionDesired, 53
setMaxRange
ArSerialDevice, 257
setMaxRotVel
ArActionDesired, 53
ArRobot, 333
setMaxTransVel
ArRobot, 334
setMaxVel

setMoveDoneDist
ArRobot, 292
setMSec
ArTime, 422
setName
ArRobot, 297
setNextArgument
ArAction, 39
setNonBlock
ArSocket, 392
setNumberOfReadings
ArInterpolation, 186

SETO
ArCommands, 113

setP1
ArFunctor1C, 139
ArFunctor2C, 148
ArGlobalFunctor1, 157
ArGlobalFunctor2, 161
ArGlobalRetFunctor1, 167
ArGlobalRetFunctor2, 170
ArRetFunctor1C, 277
ArRetFunctor2C, 285

setP2
ArFunctor2C, 148
ArGlobalFunctor2, 161
ArGlobalRetFunctor2, 170
ArRetFunctor2C, 285

setPool
ArTaskPool, 408

setPort
ArSerialConnection, 363

setPose
ArArg, 100
ArPose, 233, 234

setPoseInterpNumReadings
ArRobot, 302

setState
ArSyncTask, 403

setResolver
ArRobot, 301

setResUseAddress
ArSocket, 392

setRight
ArACTSBlob, 91

setRobot
ArAction, 39
ArActionKeydrive, 70
ArACTS_1, 85
ArIrrfDevice, 188
ArModule, 221
ArRangeDevice, 256
ArSick, 367
ArSonarDevice, 396

setRunnIng
ArPeriodicTask, 229
ArThread, 418

SETRV
ArCommands, 114

setSec
ArTime, 422

setSendingAddress
ArSickPacket, 380

setSensorPosition
ArSick, 365

SETSIMORIGINTH
ArCommands, 115

SETSIMORIGINX
ArCommands, 115

SETSIMORIGINY
ArCommands, 115

setSize
ArRangeBuffer, 255

setSocket
ArTcpConnection, 415

setSpeed
ArActionGoto, 56

setSpeeds
ArActionJoydrive, 67
ArActionKeydrive, 70
ArJoyHandler, 190

setState
ArSyncTask, 403

setStateReflectionRefreshTime
ArRobot, 334
INDEX 489

setStats
    ArJoyHandler, 191
setStats
    ArTcpConnection, 412
setStopIfNoButtonPressed
    ArActionJoydrive, 67
setString
    ArArg, 100
    ArPref, 242
setTh
    ArPose, 231
setThis
    ArFunctor1C, 140
    ArFunctor2C, 149
    ArFunctorC, 151
    ArRetFunctor1C, 278
    ArRetFunctor2C, 285
    ArRetFunctorC, 288
setThRad
    ArPose, 231
setTimeReceived
    ArRobotPacket, 340
    ArSickPacket, 379
setToNow
    ArTime, 421
setTop
    ArACTSBlob, 91
setTransform
    ArTransform, 424
setType
    ArGripper, 178
setUnitNumber
    ArAMPTUPacket, 98
setUpPacketHandlers
    ArRobot, 304
setUpSyncList
    ArRobot, 304
setUseOSCalm
    ArActionJoydrive, 69
    ArJoyHandler, 193
SETV
    ArCommands, 113
set Vel
    ArActionDesired, 54
    ArActionInput, 65
    ArRobot, 335
setVel2
    ArRobot, 335
setX
    ArPose, 231
setXCG
    ArACTSBlob, 90
setY
    ArPose, 231
setYCG
    ArACTSBlob, 90
shutdown
    Aria, 184
    ArSocket, 394
SIGHANDLE_NONE
    Aria, 182
SIGHANDLE_SINGLE
    Aria, 182
SIGHANDLE_THREAD
    Aria, 182
SigHandleMethod
    Aria, 182
signal
    ArCondition, 116
signalHandlerCB
    Aria, 182
simPacketHandler
    ArSick, 367
sin
    ArMath, 205
sizeFile
    ArUtil, 432
slaveExec
    ArDPPTU, 126
sleep
    ArUtil, 432
sockAddrIn
    ArSocket, 391
sockAddrLen
    ArSocket, 393
SONAR
    ArCommands, 114
SOUND
    ArCommands, 115
SOUNDTOG
    ArCommands, 115
SPACE
ArKeyHandler, 196
space
ArActionKeydrive, 71
SPEED
ArDPPTUCmds, 131
splitString
ArUtil, 433
startAverage
ArActionDesired, 54
startCal
ArJoyHandler, 194
State
ArSick, 369
ArTaskState, 410
STATE_CHANGE_BAUD
ArSick, 369
STATE_CONFIGURE
ArSick, 369
STATE_CONNECTED
ArSick, 369
STATE_INIT
ArSick, 369
STATE_INSTALL_MODE
ArSick, 369
STATE_NONE
ArSick, 369
STATE_SET_MODE
ArSick, 369
STATE_START_READINGS
ArSick, 369
STATE_WAIT_FOR-_CONFIGURE_ACK
ArSick, 369
STATE_WAIT_FOR_INSTALL-_MODE_ACK
ArSick, 369
STATE_WAIT_FOR_POWER_ON
ArSick, 369
STATE_WAIT_FOR_SET_MODE-_ACK
ArSick, 369
STATE_WAIT_FOR_START_ACK
ArSick, 369
stateReflector
ArRobot, 335
STATUS
ArAMPTUCmds, 95
Status
ArDeviceConnection, 119
ArModuleLoader, 225
ArMutex, 228
ArThread, 420
STATUS_ALREADY_-_DETACHED
ArThread, 420
STATUS_ALREADY_LOADED
ArModuleLoader, 225
STATUS_ALREADY_LOCKED
ArMutex, 228
STATUS_CLOSED_ERROR
ArDeviceConnection, 119
STATUS_CLOSED_NORMALLY
ArDeviceConnection, 119
STATUS_EXIT_FAILED
ArModuleLoader, 225
STATUS_FAILED
ArCondition, 117
ArMutex, 228
ArThread, 420
STATUS_FAILED_DESTROY
ArCondition, 117
STATUS_FAILED_INIT
ArCondition, 117
ArMutex, 228
STATUS_FAILED_OPEN
ArModuleLoader, 225
STATUS_INIT_FAILED
ArModuleLoader, 225
STATUS_INVALID
ArModuleLoader, 225
ArThread, 420
STATUS_JOIN_SELF
ArThread, 420
STATUS_MUTEX_FAILED
ArCondition, 117
STATUS_MUTEX_FAILED_INIT
ArCondition, 117
STATUS_NEVER_OPENED
ArDeviceConnection, 119
STATUS_NO_SUCH_THREAD
ArThread, 420
STATUS_NORESOURCE
ArThread, 420
STATUS_NOT_FOUND
   ArModuleLoader, 225
STATUS_OPEN
   ArDeviceConnection, 119
STATUS_OPEN_FAILED
   ArDeviceConnection, 119
STATUS_SUCCESS
   ArModuleLoader, 225
STATUS_WAIT_INTR
   ArCondition, 117
STATUS_WAIT_TIMEDOUT
   ArCondition, 117

StdErr
   ArLog, 201

StdOut
   ArLog, 201

STEP
   ArCommands, 114

STOP
   ArCommands, 114
   ArVCC4Commands, 440

stop
   ArRobot, 336
stopAll
   ArThread, 419
stopRunning
   ArRobot, 336
   ArThread, 417

strcmp
   ArUtil, 433, 434

STRING
   ArArg, 101

String
   ArPref, 238
stripDir
   ArUtil, 435
stripFile
   ArUtil, 435
strNToBuf
   ArBasePacket, 111
strToBuf
   ArBasePacket, 112
strToBufPadded
   ArBasePacket, 112
subAngle
   ArMath, 205
   SUCCESS
      ArTaskState, 410
   SUSPEND
      ArTaskState, 410
switchState
   ArSick, 368

TAB
   ArKeyHandler, 196
takeKeys
   ArActionKeydrive, 71
task
   ArRecurrentTask, 268

TCM2
   ArCommands, 114

Terse
   ArLog, 200

TILT
   ArDPPTUCommands, 131
tilt
   ArAMPTU, 92
   ArDPPTU, 125
   ArPTZ, 244
   ArSonyPTZ, 400
   ArVCC4, 436
tiltAccel
   ArDPPTU, 127
tiltRel
   ArAMPTU, 92
   ArDPPTU, 125
   ArPTZ, 244
   ArSonyPTZ, 400
   ArVCC4, 436

TILTSLEW
   ArAMPTUCommands, 96
   ArVCC4Commands, 440
tiltSlew
   ArAMPTU, 93
   ArDPPTU, 128
   ArVCC4, 437
tiltSlewRel
   ArDPPTU, 128
timedWait
   ArCondition, 116
transfer
ArSocket, 395
tryingToConnect
ArSick, 365
tryLock
ArMutex, 228
ArRobot, 303
ArThread, 418
tryLockDevice
ArRangeDevice, 262
ArRangeDeviceThreaded, 266
TTY2
ArCommands, 114
Type
ArArg, 101
ArGripper, 174
typedef
ArCondition, 117
ubyte
ArTypes, 426
ubyte2
ArTypes, 426
ubyte2ToBuf
ArBasePacket, 108
ubyte4
ArTypes, 426
ubyte4ToBuf
ArBasePacket, 108
ubyteToBuf
ArBasePacket, 108
ArSonyPacket, 398
unblock
ArSignalHandler, 388
unblockAll
ArSignalHandler, 389
unhandle
ArSignalHandler, 389
uninit
Aria, 185
ArTaskPool, 408
unlock
ArMutex, 227
ArRobot, 303
ArThread, 418
unlockDevice
ArRangeDevice, 262
ArRangeDeviceThreaded, 266
UP
ArKeyHandler, 196
up
ArActionKeydrive, 71
update
ArSectors, 352
UPPER
ArDPPTUCOMmands, 131
upperPanSlew
ArDPPTU, 127
upperTiltSlew
ArDPPTU, 127
USER_START
ArTaskState, 410
USERIO
ArGripper, 174
userTask
ArMode, 208
ArModeCamera, 211
ArModeGripper, 213
ArModeSonar, 215
ValType
ArPref, 238
VEL
ArCommands, 114
VEL2
ArCommands, 114
velMove
ArDPPTU, 127
VELOCITY
ArDPPTUCOMmands, 131
Verbose
ArLog, 200
verifyCheckSum
ArRobotPacket, 340
verifyCRC
ArSickPacket, 378
wait
ArCondition, 116
WAIT_CONNECTED
ArRobot, 305
WAIT_FAIL
ArRobot, 305
WAIT_FAILED_CONN ArRobot, 305
WAIT_INTR ArRobot, 305
WAIT_RUN_EXIT ArRobot, 305
WAIT_TIMEDOUT ArRobot, 305
waitForConnect ArRobot, 336
waitForConnectOrConnFail ArRobot, 337
waitForRunExit ArRobot, 337
WaitState ArRobot, 305
wakeAllConnOrFailWaitingThreads ArRobot, 337
wakeAllConnWaitingThreads ArRobot, 338
wakeAllRunExitWaitingThreads ArRobot, 338
wakeAllWaitingThreads ArRobot, 338
write ArDeviceConnection, 122
ArSerialConnection, 363
ArSocket, 395
ArTcpConnection, 415
writePacket ArDeviceConnection, 122
yield ArThread, 419

ZOOM
ArAMPTUCommands, 95
ArVCC4Commands, 441

zoom ArPTZ, 245
ArSonyPTZ, 401
ArVCC4, 437
zoomRel ArPTZ, 245
ArSonyPTZ, 401
ZOOMSTOP