# Contents

1 Welcome to PJLIB!  
1.1 What is PJLIB  
1.2 Download  
1.3 About This Documentation  
1.4 Features  
1.5 Configuring Application to use PJLIB  
1.6 Porting PJLIB  
1.7 Enjoy Using PJLIB!  

2 PJLIB Reference Module Index  
2.1 PJLIB Reference Modules  

3 PJLIB Reference Data Structure Index  
3.1 PJLIB Reference Data Structures  

4 PJLIB Reference File Index  
4.1 PJLIB Reference File List  

5 PJLIB Reference Page Index  
5.1 PJLIB Reference Related Pages  

6 PJLIB Reference Module Documentation  
6.1 Network Address Resolution  
6.2 Array helper.  
6.3 Assertion Macro  
6.4 Build Configuration  
6.5 ctype - Character Type  
6.6 Error Codes  
6.7 PJLIB’s Own Error Codes  
6.8 Exception Handling
CONTENTS

6.45 Unicode Support ........................................... 180
6.46 Time Data Type and Manipulation ......................... 182

7 PJLIB Reference Data Structure Documentation 187
  7.1 pj_addr_hdr Struct Reference ............................ 187
  7.2 pj_caching_pool Struct Reference ...................... 188
  7.3 pj_exception_state_t Struct Reference .................. 190
  7.4 pj_fd_set_t Struct Reference ............................ 191
  7.5 pj_file_stat Struct Reference ......................... 192
  7.6 pj_hash_iterator_t Struct Reference .................... 193
  7.7 pj_hostent Struct Reference .............................. 194
  7.8 pj_in6_addr Struct Reference ............................ 195
  7.9 pj_in_addr Struct Reference .............................. 196
  7.10 pj_ioqueue_callback Struct Reference ................. 197
  7.11 pj_ioqueue_op_key_t Struct Reference ................ 198
  7.12 pj_ip_route_entry Union Reference .................... 199
  7.13 pj_list Struct Reference ............................... 200
  7.14 pj_parsed_time Struct Reference ...................... 201
  7.15 pj_pool_block Struct Reference ....................... 203
  7.16 pj_pool_factory Struct Reference ..................... 204
  7.17 pj_pool_factory_policy Struct Reference ............. 206
  7.18 pj_pool_t Struct Reference ............................. 207
  7.19 pj_rbtree Struct Reference ............................. 209
  7.20 pj_rbtree_node Struct Reference ....................... 210
  7.21 pj_sockaddr Union Reference .......................... 211
  7.22 pj_sockaddr_in Struct Reference ...................... 212
  7.23 pj_sockaddr_in6 Struct Reference ..................... 213
  7.24 pj_str_t Struct Reference .............................. 214
  7.25 pj_time_val Struct Reference .......................... 215
  7.26 pj_timer_entry Struct Reference ..................... 216
  7.27 pj_timestamp Union Reference ......................... 217

8 PJLIB Reference File Documentation 219
  8.1 addr_resolv.h File Reference ............................ 219
  8.2 array.h File Reference ................................. 220
  8.3 assert.h File Reference ................................. 221
  8.4 config.h File Reference ................................. 222
8.5 ctype.h File Reference ........................................... 225
8.6 doxygen.h File Reference ........................................ 226
8.7 errno.h File Reference ........................................... 227
8.8 except.h File Reference ......................................... 229
8.9 file_access.h File Reference ................................. 231
8.10 file_io.h File Reference ....................................... 232
8.11 guid.h File Reference .......................................... 233
8.12 hash.h File Reference .......................................... 234
8.13 ioqueue.h File Reference ...................................... 235
8.14 ip_helper.h File Reference ................................. 237
8.15 list.h File Reference .......................................... 238
8.16 lock.h File Reference .......................................... 239
8.17 log.h File Reference .......................................... 240
8.18 os.h File Reference ........................................... 243
8.19 pool.h File Reference .......................................... 246
8.20 rand.h File Reference .......................................... 248
8.21 rbtree.h File Reference ...................................... 249
8.22 sock.h File Reference .......................................... 250
8.23 sock_select.h File Reference .............................. 253
8.24 string.h File Reference ....................................... 254
8.25 timer.h File Reference ....................................... 256
8.26 types.h File Reference ........................................ 257
8.27 unicode.h File Reference ................................... 260

9 PJLIB Reference Page Documentation .......................... 261
  9.1 Coding Convention ........................................... 261
  9.2 Building, and Installing PJLIB ............................. 262
  9.3 Porting PJLIB ................................................. 270
  9.4 Example: Exception Handling ............................... 274
  9.5 Example: List Manipulation ................................ 276
  9.6 Example: Log, Hello World ................................ 278
  9.7 Test: Atomic Variable ....................................... 279
  9.8 Test: Exception Handling ................................ 281
  9.9 Test: I/O Queue Performance ............................... 284
  9.10 Test: I/O Queue (TCP) ...................................... 293
  9.11 Test: I/O Queue (UDP) .................................... 307
  9.12 Test: Linked List ........................................... 321

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.13 Test: Pool</td>
<td>325</td>
</tr>
<tr>
<td>9.14 Test: Socket Select()</td>
<td>329</td>
</tr>
<tr>
<td>9.15 Test: Sleep, Time, and Timestamp</td>
<td>333</td>
</tr>
<tr>
<td>9.16 Test: Socket</td>
<td>337</td>
</tr>
<tr>
<td>9.17 Test: Socket Performance</td>
<td>346</td>
</tr>
<tr>
<td>9.18 Test: String</td>
<td>349</td>
</tr>
<tr>
<td>9.19 Test: Thread Test</td>
<td>356</td>
</tr>
<tr>
<td>9.20 Test: Timer</td>
<td>362</td>
</tr>
<tr>
<td>9.21 Test: Timestamp</td>
<td>365</td>
</tr>
</tbody>
</table>
Chapter 1

Welcome to PJLIB!

1.1 What is PJLIB

PJLIB is an Open Source, small footprint framework library written in C for making scalable applications. Because of its small footprint, it can be used in embedded applications (we hope so!), but yet the library is also aimed for facilitating the creation of high performance protocol stacks.

PJLIB is released under GPL terms.

1.2 Download

PJLIB and all documentation can be downloaded from http://www.pjsip.org.

1.3 About This Documentation

This document is generated directly from PJLIB source file using doxygen (http://www.doxygen.org). Doxygen is a great (and free!) tools for generating such documentation.

1.3.1 Version

This document corresponds to PJLIB version 0.5.10.

1.3.2 How to Read This Document

This documentation is laid out more to be a reference guide instead of tutorial, therefore first time users may find it difficult to grasp PJLIB by reading this document alone.

However, we’ve tried our best to make this document easy to follow. For first time users, we would suggest that you follow these steps when reading this documentation:

- continue reading this introduction chapter. At the end of this chapter, you’ll find section called Principles in Using PJLIB which should guide you to understand basic things about PJLIB.
Welcome to PJLIB!

• find information about specific features that you want to use in PJLIB. Use the Module Index to find
out about all features in PJLIB (if you’re browsing the HTML documentation, click on the Module
link on top of the page, or if you’re reading the PDF documentation, click on Module Documentation
on the navigation pane on the left).

1.3.3 How To’s

Please find below links to specific tasks that you probably want to do:

• How to Build PJLIB
  Please refer to Building, and Installing PJLIB page for more information.

• How to Use PJLIB in My Application
  Please refer to Configuring Application to use PJLIB for more information.

• How to Port PJLIB
  Please refer to Porting PJLIB page.

• Where to Read Samples Documentation
  Most of the modules provide link to the corresponding sample file. Alternatively, to get the list of
all examples, you can click on Related Pages on the top of HTML document or on PJLIB Page
Documentation on navigation pane of your PDF reader.

• How to Submit Code to PJLIB Project
  Please read Coding Convention before submitting your code. Send your code as patch against current
Subversion tree to the appropriate mailing list.

1.4 Features

1.4.1 It’s Open Source!

PJLIB is currently released on GPL license, but other arrangements can be made with the author.

1.4.2 Extreme Portability

PJLIB is designed to be extremely portable. It can run on any kind of processors (16-bit, 32-bit, or 64-bit,
big or little endian, single or multi-processors) and operating systems. Floating point or no floating point.
Multi-threading or not. It can even run in environment where no ANSI LIBC is available.
Currently PJLIB is known to run on these platforms:

• arm, WinCE and Windows Mobile.
• Linux/x86, (user mode and as kernel module(!)).
• Linux/alpha
1.4 Features

- Solaris/ultra.
- MacOS X/powerpc
- RTEMS (x86 and powerpc).

And efforts is under way to port PJLIB on:

- Symbian OS

1.4.3 Small in Size

One of the primary objectives is to have library that is small in size for typical embedded applications. As a rough guidance, we aim to keep the library size below 100KB for it to be considered as small. As the result, most of the functionalities in the library can be tailored to meet the requirements; user can enable/disable specific functionalities to get the desired size/performance/functionality balance.

For more info, please see Build Configuration.

1.4.4 Big in Performance

Almost everything in PJLIB is designed to achieve the highest possible performance out of the target platform.

1.4.5 No Dynamic Memory Allocations

The central idea of PJLIB is that for applications to run as fast as it can, it should not use malloc() at all, but instead should get the memory from a preallocated storage pool. There are few things that can be optimized with this approach:

- alloc() is a O(1) operation.
- no mutex is used inside alloc(). It is assumed that synchronization will be used in higher abstraction by application anyway.
- no free() is required. All chunks will be deleted when the pool is destroyed.

The performance gained on some systems can be as high as 30x speed up against malloc() and free() on certain configurations, but of course your mileage may vary.

For more information, see Fast Memory Pool

1.4.6 Operating System Abstraction

PJLIB has abstractions for features that are normally not portable across operating systems:

- Threads
  Portable thread manipulation.

- Thread Local Storage.
  Storing data in thread’s private data.
Welcome to PJLIB!

- **Mutexes.**
  Mutual exclusion protection.

- **Semaphores.**
  Semaphores.

- **Atomic Variables**
  Atomic variables and their operations.

- **Critical sections.**
  Fast locking of critical sections.

- **Lock Objects**
  High level abstraction for lock objects.

- **Event Object.**
  Event object.

- **Time Data Type and Manipulation.**
  Portable time manipulation.

- **High Resolution Timestamp**
  High resolution time value.

- **etc.**

### 1.4.7 Low-Level Network I/O

PJLIB has very portable abstraction and fairly complete set of API for doing network I/O communications. At the lowest level, PJLIB provides:

- **Socket Abstraction**
  A highly portable socket abstraction, runs on all kind of network APIs such as standard BSD socket, Windows socket, Linux **kernel** socket, PalmOS networking API, etc.

- **Network Address Resolution**
  Portable address resolution, which implements `pj_gethostbyname()`.

- **Socket select() API.**
  A portable `select()` like API (`pj_sock_select()`) which can be implemented with various back-end.

### 1.4.8 Timer Management

A passive framework for managing timer, see [Timer Heap Management](#) for more info. There is also function to retrieve high resolution timestamp from the system (see [High Resolution Timestamp](#)).
1.4 Features

1.4.9 Various Data Structures

Various data structures are provided in the library:

- String Operations
- Array helper.
- Hash Table
- Linked List
- Red/Black Balanced Tree

1.4.10 Exception Construct

A convenient TRY/CATCH like construct to propagate errors, which by default are used by the memory pool and the lexical scanner in pjlib-util. The exception construct can be used to write programs like below:

```c
#define SYNTAX_ERROR 1

PJ_TRY {
    msg = NULL;
    msg = parse_msg(buf, len);
}
PJ_CATCH ( SYNTAX_ERROR ) {
    .. handle error ..
}
PJ_END;
```

Please see Exception Handling for more information.

1.4.11 Logging Facility

PJLIB Logging Facility consists of macros to write logging information to some output device. Some of the features of the logging facility:

- the verbosity can be fine-tuned both at compile time (to control the library size) or run-time (to control the verbosity of the information).
- output device is configurable (e.g. stdout, printk, file, etc.)
- log decoration is configurable.

See Logging Facility for more information.

1.4.12 Random and GUID Generation

PJLIB provides facility to create random string (pj_create_random_string()) or globally unique identifier (see Globally Unique Identifier).
1.5 Configuring Application to use PJLIB

1.5.1 Building PJLIB

Follow the instructions in Building, and Installing PJLIB to build PJLIB.

1.5.2 Building Applications with PJLIB

Use the following settings when building applications with PJLIB.

1.5.2.1 Include Search Path

Add this to your include search path ($PJLIB is PJLIB root directory):

$PJLIB/include

1.5.2.2 Include PJLIB Header

To include all PJLIB headers:

#include <pjlib.h>

Alternatively, you can include individual PJLIB headers like this:

#include <pj/log.h>
#include <pj/os.h>

1.5.2.3 Library Path

Add this to your library search path:

$PJLIB/lib

Then add the appropriate PJLIB library to your link specification. For example, you would add libpj-i386-linux-gcc.a when you're building applications in Linux.

1.5.3 Principles in Using PJLIB

Few things that you MUST do when using PJLIB, to make sure that you create truly portable applications.

1.5.3.1 Call pj_init()

Before you do anything else, call pj_init(). This would make sure that PJLIB system is properly set up.
1.6 Porting PJLIB

1.5.3.2  Do NOT Use ANSI C

Contrary to popular teaching, ANSI C (and LIBC) is not the most portable library in the world, nor it’s the most ubiquitous. For example, LIBC is not available in Linux kernel. Also normally LIBC will be excluded from compilation of RTOSes to reduce size.

So for maximum portability, do NOT use ANSI C. Do not even try to include any other header files outside `<include/pj>`. Stick with the functionalities provided by PJLIB.

1.5.3.3  Use `pj_str_t` instead of C Strings

PJLIB uses `pj_str_t` instead of normal C strings. You SHOULD follow this convention too. Remember, ANSI string-h is not always available. And PJLIB string is faster!

1.5.3.4  Use Pool for Memory Allocations

You MUST NOT use `malloc()` or any other memory allocation functions. Use PJLIB Fast Memory Pool instead! It’s faster and most portable.

1.5.4  Use Logging for Text Display

DO NOT use `<stdio.h>` for text output. Use PJLIB Logging Facility instead.

1.6  Porting PJLIB

Please see Porting PJLIB page on more information to port PJLIB to new target.

1.7  Enjoy Using PJLIB!

We hope that you find PJLIB usefull for your application. If you have any questions, suggestions, critics, bug fixes, or anything else, we would be happy to hear it.

Enjoy using PJLIB!
Benny Prijono < bennylp at pjsip dot org >
# Chapter 2

## PJLIB Reference Module Index

### 2.1 PJLIB Reference Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Category</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>PJ Library</td>
<td>172</td>
</tr>
<tr>
<td>Build Configuration</td>
<td>22</td>
</tr>
<tr>
<td>Error Codes</td>
<td>31</td>
</tr>
<tr>
<td>PJLIB’s Own Error Codes</td>
<td>35</td>
</tr>
<tr>
<td>Data Structure</td>
<td>48</td>
</tr>
<tr>
<td>Array helper</td>
<td>19</td>
</tr>
<tr>
<td>Globally Unique Identifier</td>
<td>49</td>
</tr>
<tr>
<td>Hash Table</td>
<td>51</td>
</tr>
<tr>
<td>Linked List</td>
<td>68</td>
</tr>
<tr>
<td>Red/Black Balanced Tree</td>
<td>126</td>
</tr>
<tr>
<td>String Operations</td>
<td>153</td>
</tr>
<tr>
<td>Basic Data Types and Library Functionality</td>
<td>173</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>76</td>
</tr>
<tr>
<td>Assertion Macro</td>
<td>21</td>
</tr>
<tr>
<td>ctype - Character Type</td>
<td>27</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>38</td>
</tr>
<tr>
<td>Logging Facility</td>
<td>77</td>
</tr>
<tr>
<td>Random Number Generator</td>
<td>125</td>
</tr>
<tr>
<td>Timer Heap Management</td>
<td>167</td>
</tr>
<tr>
<td>Unicode Support</td>
<td>180</td>
</tr>
<tr>
<td>Time Data Type and Manipulation</td>
<td>182</td>
</tr>
<tr>
<td>Operating System Dependent Functionality</td>
<td>81</td>
</tr>
<tr>
<td>Input/Output</td>
<td>55</td>
</tr>
<tr>
<td>Network Address Resolution</td>
<td>17</td>
</tr>
<tr>
<td>File Access</td>
<td>42</td>
</tr>
<tr>
<td>File I/O</td>
<td>44</td>
</tr>
<tr>
<td>IOQueue: I/O Event Dispatching with Proactor Pattern</td>
<td>56</td>
</tr>
<tr>
<td>IP Interface and Routing Helper</td>
<td>67</td>
</tr>
<tr>
<td>Socket Abstraction</td>
<td>130</td>
</tr>
<tr>
<td>Socket select() API</td>
<td>150</td>
</tr>
<tr>
<td>Lock Objects</td>
<td>73</td>
</tr>
<tr>
<td>Threads</td>
<td>82</td>
</tr>
<tr>
<td>Symbian OS Specific</td>
<td>87</td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Thread Local Storage</td>
<td>88</td>
</tr>
<tr>
<td>Atomic Variables</td>
<td>89</td>
</tr>
<tr>
<td>Mutexes</td>
<td>92</td>
</tr>
<tr>
<td>Reader/Writer Mutex</td>
<td>95</td>
</tr>
<tr>
<td>Critical sections</td>
<td>98</td>
</tr>
<tr>
<td>Semaphores</td>
<td>99</td>
</tr>
<tr>
<td>Event Object</td>
<td>101</td>
</tr>
<tr>
<td>High Resolution Timestamp</td>
<td>104</td>
</tr>
<tr>
<td>Time Data Type and Manipulation</td>
<td>182</td>
</tr>
<tr>
<td>Fast Memory Pool</td>
<td>109</td>
</tr>
<tr>
<td>Memory Pool Object</td>
<td>113</td>
</tr>
<tr>
<td>Pool Factory and Policy</td>
<td>118</td>
</tr>
<tr>
<td>Caching Pool Factory</td>
<td>121</td>
</tr>
<tr>
<td>Stack/Buffer Based Memory Pool Allocator</td>
<td>123</td>
</tr>
</tbody>
</table>
## Chapter 3

### PJLIB Reference Data Structure Index

#### 3.1 PJLIB Reference Data Structures

Here are the data structures with brief descriptions:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pj_addr_hdr</td>
<td>187</td>
</tr>
<tr>
<td>pj_caching_pool</td>
<td>188</td>
</tr>
<tr>
<td>pj_exception_state_t</td>
<td>190</td>
</tr>
<tr>
<td>pj_fd_set_t</td>
<td>191</td>
</tr>
<tr>
<td>pj_file_stat</td>
<td>192</td>
</tr>
<tr>
<td>pj_hash_iterator_t</td>
<td>193</td>
</tr>
<tr>
<td>pj_hostent</td>
<td>194</td>
</tr>
<tr>
<td>pj_in6_addr</td>
<td>195</td>
</tr>
<tr>
<td>pj_in_addr</td>
<td>196</td>
</tr>
<tr>
<td>pj_ioqueue_callback</td>
<td>197</td>
</tr>
<tr>
<td>pj_ioqueue_op_key_t</td>
<td>198</td>
</tr>
<tr>
<td>pj_ip_route_entry</td>
<td>199</td>
</tr>
<tr>
<td>pj_list</td>
<td>200</td>
</tr>
<tr>
<td>pj_parsed_time</td>
<td>201</td>
</tr>
<tr>
<td>pj_pool_block</td>
<td>203</td>
</tr>
<tr>
<td>pj_pool_factory</td>
<td>204</td>
</tr>
<tr>
<td>pj_pool_factory_policy</td>
<td>206</td>
</tr>
<tr>
<td>pj_pool_t</td>
<td>207</td>
</tr>
<tr>
<td>pj_rbtree</td>
<td>209</td>
</tr>
<tr>
<td>pj_rbtree_node</td>
<td>210</td>
</tr>
<tr>
<td>pj_sockaddr</td>
<td>211</td>
</tr>
<tr>
<td>pj_sockaddr_in</td>
<td>212</td>
</tr>
<tr>
<td>pj_sockaddr_in6</td>
<td>213</td>
</tr>
<tr>
<td>pj_str_t</td>
<td>214</td>
</tr>
<tr>
<td>pj_time_val</td>
<td>215</td>
</tr>
<tr>
<td>pj_timer_entry</td>
<td>216</td>
</tr>
<tr>
<td>pj_timestamp</td>
<td>217</td>
</tr>
</tbody>
</table>
# Chapter 4

## PJLIB Reference File Index

### 4.1 PJLIB Reference File List

Here is a list of all documented files with brief descriptions:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Brief Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr_resolv.h</td>
<td>(IP address resolution)</td>
<td>219</td>
</tr>
<tr>
<td>array.h</td>
<td>(PJLIB Array helper)</td>
<td>220</td>
</tr>
<tr>
<td>assert.h</td>
<td>(Assertion macro <code>pj_assert()</code>)</td>
<td>221</td>
</tr>
<tr>
<td>config.h</td>
<td>(PJLIB Main configuration settings)</td>
<td>222</td>
</tr>
<tr>
<td>config_site_sample.h</td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>ctype.h</td>
<td>(C type helper macros)</td>
<td>225</td>
</tr>
<tr>
<td>doxygen.h</td>
<td>(Doxygen’s mainpage)</td>
<td>226</td>
</tr>
<tr>
<td>errno.h</td>
<td>(PJLIB Error Codes)</td>
<td>227</td>
</tr>
<tr>
<td>except.h</td>
<td>(Exception Handling in C)</td>
<td>229</td>
</tr>
<tr>
<td>file_access.h</td>
<td>(File manipulation and access)</td>
<td>231</td>
</tr>
<tr>
<td>file_io.h</td>
<td>(Simple file I/O abstraction)</td>
<td>232</td>
</tr>
<tr>
<td>guid.h</td>
<td>(GUID Globally Unique Identifier)</td>
<td>233</td>
</tr>
<tr>
<td>hash.h</td>
<td>(Hash Table)</td>
<td>234</td>
</tr>
<tr>
<td>iqueue.h</td>
<td>(I/O Dispatching Mechanism)</td>
<td>235</td>
</tr>
<tr>
<td>ip_helper.h</td>
<td>(IP helper API)</td>
<td>237</td>
</tr>
<tr>
<td>list.h</td>
<td>(Linked List data structure)</td>
<td>238</td>
</tr>
<tr>
<td>lock.h</td>
<td>(Higher abstraction for locking objects)</td>
<td>239</td>
</tr>
<tr>
<td>log.h</td>
<td>(Logging Utility)</td>
<td>240</td>
</tr>
<tr>
<td>os.h</td>
<td>(OS dependent functions)</td>
<td>243</td>
</tr>
<tr>
<td>pool.h</td>
<td>(Memory Pool)</td>
<td>246</td>
</tr>
<tr>
<td>pool_alt.h</td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>pool_buf.h</td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>rand.h</td>
<td>(Random Number Generator)</td>
<td>248</td>
</tr>
<tr>
<td>rtree.h</td>
<td>(Red/Black Tree)</td>
<td>249</td>
</tr>
<tr>
<td>rtems_network_config.h</td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>sock.h</td>
<td>(Socket Abstraction)</td>
<td>250</td>
</tr>
<tr>
<td>sock_select.h</td>
<td>(Socket select() )</td>
<td>253</td>
</tr>
<tr>
<td>string.h</td>
<td>(PJLIB String Operations)</td>
<td>254</td>
</tr>
<tr>
<td>test.h</td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>timer.h</td>
<td>(Timer Heap)</td>
<td>256</td>
</tr>
<tr>
<td>types.h</td>
<td>(Declaration of basic types and utility)</td>
<td>257</td>
</tr>
<tr>
<td>unicode.h</td>
<td>(Provides Unicode conversion for Unicode OSes)</td>
<td>260</td>
</tr>
</tbody>
</table>
Chapter 5

PJLIB Reference Page Index

5.1 PJLIB Reference Related Pages

Here is a list of all related documentation pages:

- Coding Convention .................................................. 261
- Building, and Installing PJLIB ................................... 262
- Porting PJLIB ............................................................. 270
- Example: Exception Handling .................................... 274
- Example: List Manipulation ....................................... 276
- Example: Log, Hello World ........................................ 278
- Test: Atomic Variable ................................................. 279
- Test: Exception Handling .......................................... 281
- Test: I/O Queue Performance ..................................... 284
- Test: I/O Queue (TCP) ................................................. 293
- Test: I/O Queue (UDP) ................................................ 307
- Test: Linked List ...................................................... 321
- Test: Pool ................................................................. 325
- Test: Socket Select() .................................................. 329
- Test: Sleep, Time, and Timestamp ............................... 333
- Test: Socket ............................................................. 337
- Test: Socket Performance ........................................... 346
- Test: String ............................................................. 349
- Test: Thread Test ....................................................... 356
- Test: Timer ............................................................... 362
- Test: Timestamp ....................................................... 365
Chapter 6

PJLIB Reference Module Documentation

6.1 Network Address Resolution

6.1.1 Detailed Description

This module provides function to resolve Internet address of the specified host name. To resolve a particular host name, application can just call `pj_gethostbyname()`.

Example:

```c
...
pj_hostent he;
pj_status_t rc;
pj_str_t host = pj_str("host.example.com");

rc = pj_gethostbyname( &host, &he);
if (rc != PJ_SUCCESS) {
    char errbuf[80];
    pj_strerror( rc, errbuf, sizeof(errbuf));
    PJ_LOG(2,("sample", "Unable to resolve host, error=%s", errbuf));
    return rc;
}

// process address...
addr.sin_addr.s_addr = *(pj_uint32_t*)he.h_addr;
...
```

It’s pretty simple really...

Data Structures

- struct `pj_hostent`
Defines

- #define h_addr h_addr_list[0]

Functions

- pj_status_t pj_gethostbyname (const pj_str_t *name, pj_hostent *he)
- pj_status_t pj_gethostip (pj_in_addr *ip_addr)

6.1.2 Define Documentation

6.1.2.1 #define h_addr h_addr_list[0]

Shortcut to h_addr_list[0]

6.1.3 Function Documentation

6.1.3.1 pj_status_t pj_gethostbyname (const pj_str_t * name, pj_hostent * he)

This function fills the structure of type pj_hostent for a given host name.

Parameters:

- name  Host name, or IPv4 or IPv6 address in standard dot notation.
- he The pj_hostent structure to be filled.

Returns:

- PJ_SUCCESS, or the appropriate error codes.

6.1.3.2 pj_status_t pj_gethostip (pj_in_addr * ip_addr)

Resolve the primary IP address of local host.

Parameters:

- ip_addr  On successful resolution, this will be filled up with the host IP address, in network byte order.

Returns:

- PJ_SUCCESS on success, or the appropriate error code.
6.2 Array helper.

6.2.1 Detailed Description

This module provides helper to manipulate array of elements of any size. It provides most used array operations such as insert, erase, and search.

Functions

- void pj_array_insert (void *array, unsigned elem_size, unsigned count, unsigned pos, const void *value)
- void pj_array_erase (void *array, unsigned elem_size, unsigned count, unsigned pos)
- pj_status_t pj_array_find (const void *array, unsigned elem_size, unsigned count, pj_status_t (*matching)(const void *value), void **result)

6.2.2 Function Documentation

6.2.2.1 void pj_array_erase (void * array, unsigned elem_size, unsigned count, unsigned pos)

Erase a value from the array at given position, and rearrange the remaining elements post the erased element.

Parameters:

array the array.

elem_size the size of the individual element.

count the current number of elements in the array.

pos the index/position to delete.

6.2.2.2 pj_status_t pj_array_find (const void *array, unsigned elem_size, unsigned count, 
pj_status_t(*)(const void *value) matching, void **result)

Search the first value in the array according to matching function.

Parameters:

array the array.

elem_size the individual size of the element.

count the number of elements.

matching the matching function, which MUST return PJ_SUCCESS if the specified element match.

result the pointer to the value found.

Returns:

PJ_SUCCESS if value is found, otherwise the error code.
6.2.2.3  void pj_array_insert (void * array, unsigned elem_size, unsigned count, unsigned pos, const void * value)

Insert value to the array at the given position, and rearrange the remaining nodes after the position.

Parameters:

array  the array.

elem_size  the size of the individual element.

count  the CURRENT number of elements in the array.

pos  the position where the new element is put.

value  the value to copy to the new element.
6.3 Assertion Macro

6.3.1 Detailed Description

Assertion and other helper macros for sanity checking.

Defines

- `#define pj_assert(expr)`
- `#define PJ_ASSERT_RETURN(expr, retval)`
- `#define PJ_ASSERT_ON_FAIL(expr, exec_on_fail)`

6.3.2 Define Documentation

6.3.2.1 `#define pj_assert(expr)`

Check during debug build that an expression is true. If the expression computes to false during run-time, then the program will stop at the offending statements. For release build, this macro will not do anything.

Parameters:

- `expr` The expression to be evaluated.

6.3.2.2 `#define PJ_ASSERT_ON_FAIL(expr, exec_on_fail)`

If `PJ_ENABLE_EXTRA_CHECK` is declared and non-zero, then `PJ_ASSERT_ON_FAIL` macro will evaluate the expression in `expr` during run-time. If the expression yields false, assertion will be triggered and `exec_on_fail` will be executed.

If `PJ_ENABLE_EXTRA_CHECK` is not declared or is zero, then no run-time checking will be performed. The macro simply evaluates to `pj_assert(expr)`.

6.3.2.3 `#define PJ_ASSERT_RETURN(expr, retval)`

If `PJ_ENABLE_EXTRA_CHECK` is declared and the value is non-zero, then `PJ_ASSERT_RETURN` macro will evaluate the expression in `expr` during run-time. If the expression yields false, assertion will be triggered and the current function will return with the specified return value.

If `PJ_ENABLE_EXTRA_CHECK` is not declared or is zero, then no run-time checking will be performed. The macro simply evaluates to `pj_assert(expr)`. 
6.4 Build Configuration

6.4.1 Detailed Description

This section contains macros that can set during PJLIB build process to control various aspects of the library.

Note: the values in this page does NOT necessarily reflect to the macro values during the build process.

Defines

- `#define PJ_DEBUG 1`
- `#define PJ_FUNCTIONS_ARE_INLINED 0`
- `#define PJ_HAS_FLOATING_POINT 1`
- `#define PJ_LOG_MAX_SIZE 2000`
- `#define PJ_LOG_USE_STACK_BUFFER 1`
- `#define PJ_TERM_HAS_COLOR 1`
- `#define PJ_SAFE_POOL 0`
- `#define PJ_POOL_DEBUG 0`
- `#define PJ_THREAD_DEFAULT_STACK_SIZE 8192`
- `#define PJ_HAS_POOL_ALT_API PJ_POOL_DEBUG`
- `#define PJ_MAX_HOSTNAME (128)`
- `#define PJ_IOQUEUE_MAX_HANDLES (256)`
- `#define PJ_IOQUEUE_HAS_SAFE_UNREG 1`
- `#define PJ_IOQUEUE_KEY_FREE_DELAY 500`
- `#define FD_SETSIZE PJ_IOQUEUE_MAX_HANDLES`
- `#define PJ_MAXPATH 260`
- `#define PJ_ENABLE_EXTRA_CHECK 1`
- `#define PJ_HAS_EXCEPTION_NAMES 1`
- `#define PJ_MAX_EXCEPTION_ID 16`
- `#define PJ_EXCEPTION_USE_WIN32_SEH 0`
- `#define PJ_TIMESTAMP_USE_RDTSC 0`
- `#define PJ_NATIVE_ERR_POSITIVE 1`
- `#define PJ_HAS_ERROR_STRING 1`
- `#define PJ_HAS_STRICMP_ALNUM 0`

6.4.2 Define Documentation

6.4.2.1 `#define FD_SETSIZE PJ_IOQUEUE_MAX_HANDLES`

Overrides `FD_SETSIZE` so it is consistent throughout the library. OS specific configuration header (compat/os_∗) might have declared `FD_SETSIZE`, thus we only set if it hasn’t been declared.

Default: `PJ_IOQUEUE_MAX_HANDLES`

6.4.2.2 `#define PJ_DEBUG 1`

If this macro is set to 1, it will enable some debugging checking in the library.

Default: equal to (NOT NDEBUG).
6.4 Build Configuration

6.4.2.3 #define PJ_ENABLE_EXTRA_CHECK 1

Enable library's extra check. If this macro is enabled, PJ_ASSERT_RETURN macro will expand to runtime checking. If this macro is disabled, PJ_ASSERT_RETURN will simply evaluate to pj_assert().
You can disable this macro to reduce size, at the risk of crashes if invalid value (e.g. NULL) is passed to the library.
Default: 1

6.4.2.4 #define PJ_EXCEPTION_USE_WIN32_SEH 0

Should we use Windows Structured Exception Handling (SEH) for the PJLIB exceptions.
Default: 0

6.4.2.5 #define PJ_FUNCTIONS_ARE_INLINED 0

Expand functions in *.h header files as inline.
Default: 0.

6.4.2.6 #define PJ_HAS_ERROR_STRING 1

Include error message string in the library (pj_strerror()). This is very much desirable!
Default: 1

6.4.2.7 #define PJ_HAS_EXCEPTION_NAMES 1

Enable name registration for exceptions with pj_exception_id_alloc(). If this feature is enabled, then the library will keep track of names associated with each exception ID requested by application via pj_exception_id_alloc().
Disabling this macro will reduce the code and .bss size by a tad bit. See also PJ_MAX_EXCEPTION_ID.
Default: 1

6.4.2.8 #define PJ_HAS_FLOATING_POINT 1

Use floating point computations in the library.
Default: 1.

6.4.2.9 #define PJ_HAS_POOL_ALT_API PJ_POOL_DEBUG

Do we have alternate pool implementation?
Default: 0

6.4.2.10 #define PJ_HAS_STRICMP_ALNUM 0

Include pj_stricmp_alnum() and pj_strnicmp_alnum(), i.e. custom functions to compare alnum strings. On some systems, they're faster then strcmp/strcasecmp, but they can be slower on other systems. When
disabled, pjlib will fallback to strcmp/strnicmp.
Default: 0

6.4.2.11  #define PJ_IOQUEUE_HAS_SAFE_UNREG 1

If PJ_IOQUEUE_HAS_SAFE_UNREG macro is defined, then ioqueue will do more things to ensure thread safety of handle unregistration operation by employing reference counter to each handle.
In addition, the ioqueue will preallocate memory for the handles, according to the maximum number of handles that is specified during ioqueue creation.
All applications would normally want this enabled, but you may disable this if:

• there is no dynamic unregistration to all ioqueues.
• there is no threading, or there is no preemptive multitasking.

Default: 1

6.4.2.12  #define PJ_IOQUEUE_KEY_FREE_DELAY 500

When safe unregistration (PJ_IOQUEUE_HAS_SAFE_UNREG) is configured in ioqueue, the PJ_-IOQUEUE_KEY_FREE_DELAY macro specifies how long the ioqueue key is kept in closing state before it can be reused.
The value is in miliseconds.
Default: 500 msec.

6.4.2.13  #define PJ_IOQUEUE_MAX_HANDLES (256)

Constants for declaring the maximum handles that can be supported by a single IOQ framework. This constant might not be relevant to the underlying I/O queue implelementation, but still, developers should be aware of this constant, to make sure that the program will not break when the underlying implementation changes.
For implementation based on select(), the value here will be used as the maximum number of socket handles passed to select() (i.e. FD_SETSIZE will be set to this value).
Default: if FD_SETSIZE is defined and the value is greater than 256, then it will be used. Otherwise 256 (64 for WinCE).

6.4.2.14  #define PJ_LOG_MAX_SIZE 2000

Maximum message size that can be sent to output device for each call to PJ_LOG(). If the message size is longer than this value, it will be cut. This may affect the stack usage, depending whether PJ_LOG_USE_-STACK_BUFFER flag is set.
Default: 2000

6.4.2.15  #define PJ_LOG_USE_STACK_BUFFER 1

Log buffer. Does the log get the buffer from the stack? (default is yes). If the value is set to NO, then the buffer will be taken from static buffer, which in this case will make the log function non-reentrant.
6.4 Build Configuration

Default: 1

6.4.2.16 #define PJ_MAX_EXCEPTION_ID 16

Maximum number of unique exception IDs that can be requested with `pj_exception_id_alloc()`. For each entry, a small record will be allocated in the `.bss` segment.
Default: 16

6.4.2.17 #define PJ_MAX_HOSTNAME (128)

Maximum hostname length. Libraries sometimes needs to make copy of an address to stack buffer; the value here affects the stack usage.
Default: 128

6.4.2.18 #define PJ_MAXPATH 260

Maximum file name length.

6.4.2.19 #define PJ_NATIVE_ERR_POSITIVE 1

Is native platform error positive number? Default: 1 (yes)

6.4.2.20 #define PJ_POOL_DEBUG 0

If pool debugging is used, then each memory allocation from the pool will call malloc(), and pool will release all memory chunks when it is destroyed. This works better when memory verification programs such as Rational Purify is used.
Default: 0

6.4.2.21 #define PJ_SAFE_POOL 0

Set this flag to non-zero to enable various checking for pool operations. When this flag is set, assertion must be enabled in the application.
This will slow down pool creation and destruction and will add few bytes of overhead, so application would normally want to disable this feature on release build.
Default: 0

6.4.2.22 #define PJ_TERM_HAS_COLOR 1

Colorfull terminal (for logging etc).
Default: 1
6.4.2.23  

#define PJ_THREAD_DEFAULT_STACK_SIZE 8192

Specify this as stack_size argument in pj_thread_create() to specify that thread should use default stack size for the current platform.

Default: 8192

6.4.2.24  

#define PJ_TIMESTAMP_USE_RDTSC 0

Should we attempt to use Pentium’s rdtsc for high resolution timestamp.

Default: 0
6.5 ctype - Character Type

6.5.1 Detailed Description

This module contains several inline functions/macros for testing or manipulating character types. It is provided in PJLIB because PJLIB must not depend to LIBC.

Defines

- `#define pj_hex_digits "0123456789abcdef"

Functions

- `int pj_isalnum (int c)`
- `int pj_isalpha (int c)`
- `int pj_isascii (int c)`
- `int pj_isdigit (int c)`
- `int pj_isspace (int c)`
- `int pj_islower (int c)`
- `int pj_isupper (int c)`
- `int pj_isblank (int c)`
- `int pj_tolower (int c)`
- `int pj_toupper (int c)`
- `int pj_isxdigit (int c)`
- `void pj_val_to_hex_digit (unsigned value, char *p)`
- `unsigned pj_hex_digit_to_val (unsigned c)`

6.5.2 Define Documentation

6.5.2.1 `#define pj_hex_digits "0123456789abcdef"

Array of hex digits, in lowerspace.

6.5.3 Function Documentation

6.5.3.1 `unsigned pj_hex_digit_to_val (unsigned c)`

Convert hex digit `c` to integral value.

Parameters:

- `c` The hex digit character.

Returns:

The integral value between 0 and 15.
6.5.3.2 int pj_isalnum (int c)

Returns a non-zero value if either isalpha or isdigit is true for c.

Parameters:
   c The integer character to test.

Returns:
   Non-zero value if either isalpha or isdigit is true for c.

6.5.3.3 int pj_isalpha (int c)

Returns a non-zero value if c is a particular representation of an alphabetic character.

Parameters:
   c The integer character to test.

Returns:
   Non-zero value if c is a particular representation of an alphabetic character.

6.5.3.4 int pj_isascii (int c)

Returns a non-zero value if c is a particular representation of an ASCII character.

Parameters:
   c The integer character to test.

Returns:
   Non-zero value if c is a particular representation of an ASCII character.

6.5.3.5 int pj_isblank (int c)

Returns a non-zero value if c is a either a space (' ') or horizontal tab ('\t') character.

Parameters:
   c The integer character to test.

Returns:
   Non-zero value if c is a either a space (' ') or horizontal tab ('\t') character.
6.5.3.6  int pj_isdigit (int c)

Returns a non-zero value if c is a particular representation of a decimal-digit character.

Parameters:
   c  The integer character to test.

Returns:
   Non-zero value if c is a particular representation of a decimal-digit character.

6.5.3.7  int pj_islower (int c)

Returns a non-zero value if c is a particular representation of a lowercase character.

Parameters:
   c  The integer character to test.

Returns:
   Non-zero value if c is a particular representation of a lowercase character.

6.5.3.8  int pj_isspace (int c)

Returns a non-zero value if c is a particular representation of a space character (0x09 - 0x0D or 0x20).

Parameters:
   c  The integer character to test.

Returns:
   Non-zero value if c is a particular representation of a space character (0x09 - 0x0D or 0x20).

6.5.3.9  int pj_isupper (int c)

Returns a non-zero value if c is a particular representation of a uppercase character.

Parameters:
   c  The integer character to test.

Returns:
   Non-zero value if c is a particular representation of a uppercase character.
6.5.3.10 int pj_isxdigit (int c)

Returns a non-zero value if c is a particular representation of an hexadecimal digit character.

Parameters:
   c The integer character to test.

Returns:
   Non-zero value if c is a particular representation of an hexadecimal digit character.

6.5.3.11 int pjtolower (int c)

Converts character to lowercase.

Parameters:
   c The integer character to convert.

Returns:
   Lowercase character of c.

6.5.3.12 int pjtoupper (int c)

Converts character to uppercase.

Parameters:
   c The integer character to convert.

Returns:
   Uppercase character of c.

6.5.3.13 void pj_val_to_hex_digit (unsigned value, char * p)

Convert a value to hex representation.

Parameters:
   value Integral value to convert.
   p Buffer to hold the hex representation, which must be at least two bytes length.
6.6 Error Codes

6.6.1 Detailed Description

In PJLIB, error/status codes from operating system are translated into PJLIB error namespace, and stored in \texttt{pj\_status\_t}. All functions that work with \texttt{pj\_status\_t} expect to get PJLIB error code instead of native codes.

6.6.2 Return Values

All functions that returns \texttt{pj\_status\_t} returns \texttt{PJ\_SUCCESS} if the operation was completed successfully, or non-zero value to indicate error. If the error came from operating system, then the native error code is translated/folded into PJLIB’s error namespace by using \texttt{PJ\_STATUS\_FROM\_OS()} macro. The function will do this automatically before returning the error to caller.

6.6.3 Error Message

To get the error message corresponding to a particular code, use function \texttt{pj\__strerror()}. This function expects error code in PJLIB error namespace, not the native error code. Application can pass the value from the following sources to this function:

- \texttt{pj\_get\_os\_error()}
- \texttt{pj\_get\_netos\_error()}
- any return value from function returning \texttt{pj\_status\_t}.

Application MUST NOT pass native error code (such as error code from functions like GetLastError() or errno) to PJLIB functions expecting \texttt{pj\_status\_t}.

Modules

- PJLIB’s Own Error Codes

Defines

- \#define \texttt{PJ\_ERR\_MSG\_SIZE} 80
- \#define \texttt{PJ\_RETURN\_OS\_ERROR(os\_code)}
- \#define \texttt{PJ\_STATUS\_FROM\_OS(e)}
- \#define \texttt{PJ\_STATUS\_TO\_OS(e)}

Typedefs

- typedef \texttt{pj\_str\_t(\*)} pjsip\_error\_callback (\texttt{pj\_status\_t, char \*, pj\_size\_t})
Functions

• `pj_status_t pj_get_os_error (void)`
• `void pj_set_os_error (pj_status_t code)`
• `pj_status_t pj_get_netos_error (void)`
• `void pj_set_netos_error (pj_status_t code)`
• `pj_str_t pj_strerror (pj_status_t statcode, char *buf, pj_size_t bufsize)`
• `pj_status_t pj_register_strerror (pj_status_t start_code, pj_status_t err_space, pjsip_error_callback f)`

6.6.4 Define Documentation

6.6.4.1 `#define PJ_ERR_MSG_SIZE 80`

Guidelines on error message length.

6.6.4.2 `#define PJ_RETURN_OS_ERROR(os_code)`

Return platform os error code folded into `pj_status_t` code. This is the macro that is used throughout the library for all PILIB’s functions that returns error from operating system. Application may override this macro to reduce size (e.g. by defining it to always return `PJ_EUNKNOWN`).

Note: This macro MUST return non-zero value regardless whether zero is passed as the argument. The reason is to protect logic error when the operating system doesn’t report error codes properly.

Parameters:

`os_code` Platform OS error code. This value may be evaluated more than once.

Returns:

The platform os error code folded into `pj_status_t`.

6.6.4.3 `#define PJ_STATUS_FROM_OS(e)`

Fold a platform specific error into an `pj_status_t` code.

Parameters:

`e` The platform os error code.

Returns:

`pj_status_t`

Warning:

Macro implementation; the syserr argument may be evaluated multiple times.
6.6 Error Codes

6.6.4.4 #define PJ_STATUS_TO_OS(e)

Fold an pj_status_t code back to the native platform defined error.

Parameters:
    e  The pj_status_t folded platform os error code.

Returns:
    pj_os_err_type

Warning:
    macro implementation; the statcode argument may be evaluated multiple times. If the statcode was not
    created by pj_get_os_error or PJ_STATUS_FROM_OS, the results are undefined.

6.6.5 Function Documentation

6.6.5.1 pj_status_t pj_get_netos_error (void)

Get the last error from socket operations.

Returns:
    Last socket error, folded into pj_status_t.

6.6.5.2 pj_status_t pj_get_os_error (void)

Get the last platform error/status, folded into pj_status_t.

Returns:
    OS dependent error code, folded into pj_status_t.

Remarks:
    This function gets errno, or calls GetLastError() function and convert the code into pj_status_t with
    PJ_STATUS_FROM_OS. Do not call this for socket functions!

See also:
    pj_get_netos_error()

6.6.5.3 pj_status_t pj_register_strerror (pj_status_t start_code,
                        pj_status_t err_space,
                        pjsip_error_callback f)

Register strerror message handler for the specified error space. Application can register its own handler to
supply the error message for the specified error code range. This handler will be called by pj_strerror().

Parameters:
    start_code  The starting error code where the handler should be called to retrieve the error message.
**err_space** The size of error space. The error code range then will fall in start_code to start_code+err_space-1 range.

**f** The handler to be called when pj_strerror() is supplied with error code that falls into this range.

**Returns:**

PJ_SUCCESS or the specified error code. The registration may fail when the error space has been occupied by other handler, or when there are too many handlers registered to PJLIB.

### 6.6.5.4 void pj_set_netos_error (pj_status_t code)

Set error code.

**Parameters:**

- **code** pj_status_t

### 6.6.5.5 void pj_set_os_error (pj_status_t code)

Set last error.

**Parameters:**

- **code** pj_status_t

### 6.6.5.6 pj_str_t pj_strerror (pj_status_t statcode, char * buf, pj_size_t.bufsize)

Get the error message for the specified error code. The message string will be NULL terminated.

**Parameters:**

- **statcode** The error code.
- **buf** Buffer to hold the error message string.
- **bufsize** Size of the buffer.

**Returns:**

The error message as NULL terminated string, wrapped with pj_str_t.
6.7 PJLIB’s Own Error Codes

Defines

- \#define PJ_BUILD_ERR(code, msg) { code, msg "(" #code ")" }
- \#define PJ_EUNKNOWN
- \#define PJ_EPENDING
- \#define PJ_ETOOMANYCONN
- \#define PJ_EINVAL
- \#define PJ_ENAMETOOLONG
- \#define PJ_ENOTFOUND
- \#define PJ_ENOMEM
- \#define PJ_EBUG
- \#define PJ_ETIMEDOUT
- \#define PJ_ETOOMANY
- \#define PJ_EBUSY
- \#define PJ_ENOTSUP
- \#define PJ_EINVALIDOP
- \#define PJ_ECANCELLED
- \#define PJ_EEXISTS
- \#define PJ_EEOF
- \#define PJ_EOObIG
- \#define PJ_ERESOLVE
- \#define PJ_ETOOSMALL

6.7.1 Define Documentation

6.7.1.1 \#define PJ_BUILD_ERR(code, msg) { code, msg "(" #code ")" }

Use this macro to generate error message text for your error code, so that they look uniformly as the rest of the libraries.

Parameters:

- **code** The error code
- **msg** The error test.

6.7.1.2 \#define PJ_EBUG

Bug detected!

6.7.1.3 \#define PJ_EBUSY

Object is busy.

6.7.1.4 \#define PJ_ECANCELLED

Operation is cancelled.
6.7.1.5  #define PJ_EEOF

End of file.

6.7.1.6  #define PJ_EEXISTS

Object already exists.

6.7.1.7  #define PJ_EINVAL

Invalid argument.

6.7.1.8  #define PJ_EINVALIDOP

Invalid operation.

6.7.1.9  #define PJ_ENAMETOOLONG

Name too long (eg. hostname too long).

6.7.1.10  #define PJ_ENOMEM

Not enough memory.

6.7.1.11  #define PJ_ENOTFOUND

Not found.

6.7.1.12  #define PJ_ENOTSUP

The specified option is not supported.

6.7.1.13  #define PJ_EPENDING

The operation is pending and will be completed later.

6.7.1.14  #define PJ_ERESOLVE

Error in gethostbyname(). This is a generic error returned when gethostbyname() has returned an error.

6.7.1.15  #define PJ_ETIMEDOUT

Operation timed out.
6.7 PJLIB’s Own Error Codes

6.7.1.16 #define PJ_ETOOBIG

Size is too big.

6.7.1.17 #define PJ_ETOOMANY

Too many objects.

6.7.1.18 #define PJ_ETOOMANYCONN

Too many connecting sockets.

6.7.1.19 #define PJ_ETOOSMALL

Size is too small.

6.7.1.20 #define PJ_EUNKNOWN

Unknown error has been reported.
6.8 Exception Handling

6.8.1 Detailed Description

6.8.2 Quick Example

For the impatient, take a look at some examples:

- Example: Exception Handling
- Test: Exception Handling

6.8.3 Exception Handling

This module provides exception handling syntactically similar to C++ in C language. In Win32 systems, it uses Windows Structured Exception Handling (SEH) if macro PJ_EXCEPTION_USE_WIN32_SEH is non-zero. Otherwise it will use setjmp() and longjmp().

On some platforms where setjmp/longjmp is not available, setjmp/longjmp implementation is provided. See <pj/compat/setjmp.h> for compatibility.

The exception handling mechanism is completely thread safe, so the exception thrown by one thread will not interfere with other thread.

CAVEATS:

- unlike C++ exception, the scheme here won’t call destructors of local objects if exception is thrown. Care must be taken when a function hold some resource such as pool or mutex etc.

- You CAN NOT make nested exception in one single function without using a nested PJ_USE_EXCEPTION.

- You can not provide more than PJ_CATCH or PJ_CATCH_ANY nor use PJ_CATCH and PJ_CATCH_ANY for a single PJ_TRY.

- Exceptions will always be caught by the first handler (unlike C++ where exception is only caught if the type matches.

The exception handling constructs are similar to C++. The blocks will be constructed similar to the following sample:

```c
#define NO_MEMORY 1
#define SYNTAX_ERROR 2

int sample1()
{
    PJ_USE_EXCEPTION; // declare local exception stack.

    PJ_TRY {
        ...// do something..
    }
    PJ_CATCH(NO_MEMORY) {
        ... // handle exception 1
    }
    PJ_END;
}

int sample2()
```
6.8 Exception Handling

```cpp
PJ_USE_EXCEPTION; // declare local exception stack.

PJ_TRY {
  ...// do something..
}
PJ_CATCH_ANY {
  if (PJ_GET_EXCEPTION() == NO_MEMORY)
    ...; // handle no memory situation
  else if (PJ_GET_EXCEPTION() == SYNTAX_ERROR)
    ...; // handle syntax error
}
PJ_END;
```

The above sample uses hard coded exception ID. It is strongly recommended that applications request a unique exception ID instead of hard coded value like above.

6.8.4 Exception ID Allocation

To ensure that exception ID (number) are used consistently and to prevent ID collisions in an application, it is strongly suggested that applications allocate an exception ID for each possible exception type. As a bonus of this process, the application can identify the name of the exception when the particular exception is thrown.

Exception ID management are performed with the following APIs:

- `pj_exception_id_alloc()`.
- `pj_exception_id_free()`.
- `pj_exception_id_name()`.

PJLIB itself automatically allocates one exception id, i.e. PJ_NO_MEMORY_EXCEPTION which is declared in `<pj/pool.h>`. This exception ID is raised by default pool policy when it fails to allocate memory.

6.8.5 Keywords

6.8.5.1 PJ_THROW(expression)

Throw an exception. The expression thrown is an integer as the result of the expression. This keyword can be specified anywhere within the program.

6.8.5.2 PJ_USE_EXCEPTION

Specify this in the variable definition section of the function block (or any blocks) to specify that the block has `PJ_TRY/PJ_CATCH` exception block. Actually, this is just a macro to declare local variable which is used to push the exception state to the exception stack. Note: you must specify PJ_USE_EXCEPTION as the last statement in the local variable declarations, since it may evaluate to nothing.

6.8.5.3 PJ_TRY

The `PJ_TRY` keyword is typically followed by a block. If an exception is thrown in this block, then the execution will resume to the `PJ_CATCH` handler.
6.8.5.4 PJ_CATCH(expression)

The PJ_CATCH is normally followed by a block. This block will be executed if the exception being thrown is equal to the expression specified in the PJ_CATCH.

6.8.5.5 PJ_CATCH_ANY

The PJ_CATCH is normally followed by a block. This block will be executed if any exception was raised in the TRY block.

6.8.5.6 PJ_END

Specify this keyword to mark the end of PJ_TRY / PJ_CATCH blocks.

6.8.5.7 PJ_GET_EXCEPTION(void)

Get the last exception thrown. This macro is normally called inside the PJ_CATCH or PJ_CATCH_ANY block, although it can be used anywhere where the PJ_USE_EXCEPTION definition is in scope.

6.8.6 Examples

For some examples on how to use the exception construct, please see:

- Example: Exception Handling
- Test: Exception Handling

Functions

- pj_status_t pj_exception_id_alloc (const char *name, pj_exception_id_t *id)
- pj_status_t pj_exception_id_free (pj_exception_id_t id)
- const char * pj_exception_id_name (pj_exception_id_t id)

6.8.7 Function Documentation

6.8.7.1 pj_status_t pj_exception_id_alloc (const char * name, pj_exception_id_t * id)

Allocate a unique exception id. Applications don’t have to allocate a unique exception ID before using the exception construct. However, by doing so it ensures that there is no collisions of exception ID.

As a bonus, when exception number is acquired through this function, the library can assign name to the exception (only if PJ_HAS_EXCEPTION_NAMES is enabled (default is yes)) and find out the exception name when it catches an exception.

Parameters:

- name Name to be associated with the exception ID.
- id Pointer to receive the ID.
6.8 Exception Handling

Returns:

PJ_SUCCESS on success or PJ_ETOOMANY if the library is running out of IDs.

6.8.7.2 pj_status_t pj_exception_id_free (pj_exception_id_t id)

Free an exception id.

Parameters:

id  The exception ID.

Returns:

PJ_SUCCESS or the appropriate error code.

6.8.7.3 const char *pj_exception_id_name (pj_exception_id_t id)

Retrieve name associated with the exception id.

Parameters:

id  The exception ID.

Returns:

The name associated with the specified ID.
6.9  File Access

Data Structures

• struct pj_file_stat

Functions

• pj_bool_t pj_file_exists (const char *filename)
• pj_off_t pj_file_size (const char *filename)
• pj_status_t pj_file_delete (const char *filename)
• pj_status_t pj_file_move (const char *oldname, const char *newname)
• pj_status_t pj_file_getstat (const char *filename, pj_file_stat *stat)

6.9.1  Function Documentation

6.9.1.1  pj_status_t pj_file_delete (const char * filename)

Delete a file.

Parameters:

  filename  The filename.

Returns:

  PJ_SUCCESS on success or the appropriate error code.

6.9.1.2  pj_bool_t pj_file_exists (const char * filename)

Returns non-zero if the specified file exists.

Parameters:

  filename  The file name.

Returns:

  Non-zero if the file exists.

6.9.1.3  pj_status_t pj_file_getstat (const char * filename, pj_file_stat * stat)

Return information about the specified file. The time information in the stat structure will be in local time.

Parameters:

  filename  The filename.
  stat  Pointer to variable to receive file information.

Returns:

  PJ_SUCCESS on success or the appropriate error code.
6.9 File Access

6.9.1.4  `pj_status_t pj_file_move (const char * oldname, const char * newname)`

Move a `oldname` to `newname`. If `newname` already exists, it will be overwritten.

Parameters:
- `oldname`  The file to rename.
- `newname`  New filename to assign.

Returns:
PJ_SUCCESS on success or the appropriate error code.

6.9.1.5  `pj_off_t pj_file_size (const char * filename)`

Returns the size of the file.

Parameters:
- `filename`  The file name.

Returns:
The file size in bytes or -1 on error.
6.10 File I/O

6.10.1 Detailed Description

This file contains functionalities to perform file I/O. The file I/O can be implemented with various back-end, either using native file API or ANSI stream.

6.10.2 Size Limits

There may be limitation on the size that can be handled by the `pj_file_setpos()` or `pj_file_getpos()` functions. The API itself uses 64-bit integer for the file offset/position (where available); however some backends (such as ANSI) may only support signed 32-bit offset resolution.

Reading and writing operation uses signed 32-bit integer to indicate the size.

Enumerations

- `enum pj_file_access` { PJ_O_RDONLY = 0x1101, PJ_O_WRONLY = 0x1102, PJ_O_RDWR = 0x1103, PJ_O_APPEND = 0x1108 }
- `enum pj_file_seek_type` { PJ_SEEK_SET = 0x1201, PJ_SEEK_CUR = 0x1202, PJ_SEEK_END = 0x1203 }

Functions

- `pj_status_t pj_file_open (pj_pool_t *pool, const char *pathname, unsigned flags, pj_oshandle_t *fd)`
- `pj_status_t pj_file_close (pj_oshandle_t fd)`
- `pj_status_t pj_file_write (pj_oshandle_t fd, const void *data, pj_ssize_t *size)`
- `pj_status_t pj_file_read (pj_oshandle_t fd, void *data, pj_ssize_t *size)`
- `pj_status_t pj_file_setpos (pj_oshandle_t fd, pj_off_t offset, enum pj_file_seek_type whence)`
- `pj_status_t pj_file_getpos (pj_oshandle_t fd, pj_off_t *pos)`
- `pj_status_t pj_file_flush (pj_oshandle_t fd)`

6.10.3 Enumeration Type Documentation

6.10.3.1 `enum pj_file_access`

These enumerations are used when opening file. Values PJ_O_RDONLY, PJ_O_WRONLY, and PJ_O_RDWR are mutually exclusive. Value PJ_O_APPEND can only be used when the file is opened for writing.

Enumerator:

- `PJ_O_RDONLY` Open file for reading.
- `PJ_O_WRONLY` Open file for writing.
- `PJ_O_RDWR` Open file for reading and writing. File will be truncated.
- `PJ_O_APPEND` Append to existing file.
6.10 File I/O

6.10.3.2 enum pj_file_seek_type

The seek directive when setting the file position with `pj_file_setpos`.

**Enumerator:**

- **PJ_SEEK_SET** Offset from beginning of the file.
- **PJ_SEEK_CUR** Offset from current position.
- **PJ_SEEK_END** Size of the file plus offset.

6.10.4 Function Documentation

6.10.4.1 `pj_status_t pj_file_close (pj_oshandle_t fd)`

Close an opened file descriptor.

**Parameters:**

- `fd` The file descriptor.

**Returns:**

PJ_SUCCESS or the appropriate error code on error.

6.10.4.2 `pj_status_t pj_file_flush (pj_oshandle_t fd)`

Flush file buffers.

**Parameters:**

- `fd` The file descriptor.

**Returns:**

PJ_SUCCESS or the appropriate error code on error.

6.10.4.3 `pj_status_t pj_file_getpos (pj_oshandle_t fd, pj_off_t *pos)`

Get current file position.

**Parameters:**

- `fd` The file descriptor.
- `pos` On return contains the file position as measured from the beginning of the file.

**Returns:**

PJ_SUCCESS or the appropriate error code on error.
6.10.4.4  **pj_status_t pj_file_open (pj_pool_t * pool, const char * pathname, unsigned flags, pj_oshandle_t * fd)**

Open the file as specified in `pathname` with the specified mode, and return the handle in `fd`. All files will be opened as binary.

**Parameters:**

- `pool` Pool to allocate memory for the new file descriptor.
- `pathname` File name to open.
- `flags` Open flags, which is bitmask combination of `pj_file_access` enum. The flag must be either PJ_\_O_RDONLY, PJ_O_WRONLY, or PJ_O_RDWR. When file writing is specified, existing file will be truncated unless PJ_O_APPEND is specified.
- `fd` The returned descriptor.

**Returns:**

- PJ_SUCCESS or the appropriate error code on error.

6.10.4.5  **pj_status_t pj_file_read (pj_oshandle_t fd, void * data, pj_ssize_t * size)**

Read data from the specified file. When end-of-file condition is set, this function will return PJ_SUCCESS but the size will contain zero.

**Parameters:**

- `fd` The file descriptor.
- `data` Pointer to buffer to receive the data.
- `size` On input, specifies the maximum number of data to read from the file. On output, it contains the size of data actually read from the file. It will contain zero when EOF occurs.

**Returns:**

- PJ_SUCCESS or the appropriate error code on error. When EOF occurs, the return is PJ_SUCCESS but size will report zero.

6.10.4.6  **pj_status_t pj_file_setpos (pj_oshandle_t fd, pj_off_t offset, enum pj_file_seek_type whence)**

Set file position to new offset according to directive `whence`.

**Parameters:**

- `fd` The file descriptor.
- `offset` The new file position to set.
- `whence` The directive.

**Returns:**

- PJ_SUCCESS or the appropriate error code on error.
6.10.4.7  `pj_status_t pj_file_write (pj_oshandle_t fd, const void * data, pj_ssize_t * size)`

Write data with the specified size to an opened file.

**Parameters:**

- `fd` The file descriptor.
- `data` Data to be written to the file.
- `size` On input, specifies the size of data to be written. On return, it contains the number of data actually written to the file.

**Returns:**

PJ_SUCCESS or the appropriate error code on error.
6.11 Data Structure.

Modules

- Array helper.
- Globally Unique Identifier
- Hash Table
- Linked List
- Red/Black Balanced Tree

Red/Black tree is the variant of balanced tree, where the search, insert, and delete operation is guaranteed to take at most $O(\lg(n))$.

- String Operations
- Basic Data Types and Library Functionality.
6.12 Globally Unique Identifier

6.12.1 Detailed Description

This module provides API to create string that is globally unique. If application doesn’t require that strong requirement, it can just use `pj_create_random_string()` instead.

Defines

- `#define PJ_GUID_MAX_LENGTH 36`

Functions

- ` PJ_DECL_DATA (const unsigned) PJ_GUID_STRING_LENGTH`
- `unsigned pj_GUID_STRING_LENGTH ()`
- `pj_str_t * pj_generate_unique_string (pj_str_t *str)`
- `void pj_create_unique_string (pj_pool_t *pool, pj_str_t *str)`

6.12.2 Define Documentation

6.12.2.1 `#define PJ_GUID_MAX_LENGTH 36`

`PJ_GUID_MAX_LENGTH` specifies the maximum length of GUID string, regardless of which algorithm to use.

6.12.3 Function Documentation

6.12.3.1 `void pj_create_unique_string (pj_pool_t *pool, pj_str_t * str)`

Generate a unique string.

Parameters:

- `pool` Pool to allocate memory from.
- `str` The string.

6.12.3.2 `PJ_DECL_DATA (const unsigned)`

`PJ_GUID_STRING_LENGTH` specifies length of GUID string. The value is dependent on the algorithm used internally to generate the GUID string. If real GUID generator is used, then the length will be between 32 and 36 bytes. If shadow GUID generator is used, then the length will be 20 bytes. Application should not assume which algorithm will be used by GUID generator.

Regardless of the actual length of the GUID, it will not exceed `PJ_GUID_MAX_LENGTH` characters.

See also:

- `pj_GUID_STRING_LENGTH()`
- `PJ_GUID_MAX_LENGTH`
6.12.3.3  `pj_str_t* pj_generate_unique_string (pj_str_t* str)`

Create a globally unique string, which length is PJ_GUID_STRING_LENGTH characters. Caller is re-
sponsible for preallocating the storage used in the string.

Parameters:
   
   `str` The string to store the result.

Returns:
   
   The string.

6.12.3.4  `unsigned pj_GUID_STRING_LENGTH ()`

Get PJ_GUID_STRING_LENGTH constant.
6.13 Hash Table

6.13.1 Detailed Description

A hash table is a dictionary in which keys are mapped to array positions by hash functions. Having the keys of more than one item map to the same position is called a collision. In this library, we will chain the nodes that have the same key in a list.

Defines

- #define PJ_HASH_KEY_STRING ((unsigned)-1)
- #define PJ_HASH_ENTRY_BUF_SIZE (3*sizeof(void*) + 2*sizeof(pj_uint32_t))

Typedefs

- typedef void * pj_hash_entry_buf [((3*sizeof(void*)+2*sizeof(pj_uint32_t))+sizeof(void *)-1)/(sizeof(void *))]

Functions

- pj_uint32_t pj_hash_calc (pj_uint32_t hval, const void *key, unsigned keylen)
- pj_uint32_t pj_hash_calc_tolower (pj_uint32_t hval, char *result, const pj_str_t *key)
- pj_hash_table_t * pj_hash_create (pj_pool_t *pool, unsigned size)
- void * pj_hash_get (pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t *hval)
- void pj_hash_set (pj_pool_t *pool, pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t hval, void *value)
- void pj_hash_set_np (pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t hval, pj_hash_entry_buf entry_buf, void *value)
- unsigned pj_hash_count (pj_hash_table_t *ht)
- pj_hash_iterator_t * pj_hash_first (pj_hash_table_t *ht, pj_hash_iterator_t *it)
- pj_hash_iterator_t * pj_hash_next (pj_hash_table_t *ht, pj_hash_iterator_t *it)
- void * pj_hash_this (pj_hash_table_t *ht, pj_hash_iterator_t *it)

6.13.2 Define Documentation

6.13.2.1 #define PJ_HASH_ENTRY_BUF_SIZE (3*sizeof(void*) + 2*sizeof(pj_uint32_t))

This indicates the size of each hash entry.

6.13.2.2 #define PJ_HASH_KEY_STRING ((unsigned)-1)

If this constant is used as keylen, then the key is interpreted as NULL terminated string.

6.13.3 Typedef Documentation

6.13.3.1 typedef void * pj_hash_entry_buf [((3*sizeof(void*)+2*sizeof(pj_uint32_t))+sizeof(void *)-1)/(sizeof(void *))]

Type declaration for entry buffer, used by pj_hash_set_np()
6.13.4 Function Documentation

6.13.4.1 **pj_uint32_t pj_hash_calc (pj_uint32_t hval, const void * key, unsigned keylen)**

This is the function that is used by the hash table to calculate hash value of the specified key.

**Parameters:**
- *hval* the initial hash value, or zero.
- *key* the key to calculate.
- *keylen* the length of the key, or PJ_HASH_KEY_STRING to treat the key as null terminated string.

**Returns:**
the hash value.

6.13.4.2 **pj_uint32_t pj_hash_calc_tolower (pj_uint32_t hval, char * result, const pj_str_t * key)**

Convert the key to lowercase and calculate the hash value. The resulting string is stored in *result*.

**Parameters:**
- *hval* The initial hash value, normally zero.
- *result* Buffer to store the result, which must be enough to hold the string.
- *key* The input key to be converted and calculated.

**Returns:**
The hash value.

6.13.4.3 **unsigned pj_hash_count (pj_hash_table_t * ht)**

Get the total number of entries in the hash table.

**Parameters:**
- *ht* the hash table.

**Returns:**
the number of entries in the hash table.

6.13.4.4 **pj_hash_table_t * pj_hash_create (pj_pool_t * pool, unsigned size)**

Create a hash table with the specified 'bucket' size.

**Parameters:**
- *pool* the pool from which the hash table will be allocated from.
- *size* the bucket size, which will be round-up to the nearest 2^n-1

**Returns:**
the hash table.
6.13 Hash Table

6.13.4.5  **pj_hash_iterator_t**  \*ptr **pj_hash_first**( **pj_hash_table_t**  \*ht, **pj_hash_iterator_t**  \*it)

Get the iterator to the first element in the hash table.

**Parameters:**

  *ht*  the hash table.
  *it*  the iterator for iterating hash elements.

**Returns:**

  the iterator to the hash element, or NULL if no element presents.

6.13.4.6  **void**  \*ptr **pj_hash_get**( **pj_hash_table_t**  \*ht, **const void**  \*key, **unsigned**  \*keylen, **pj_uint32_t**  \*hval)

Get the value associated with the specified key.

**Parameters:**

  *ht*  the hash table.
  *key*  the key to look for.
  *keylen*  the length of the key, or PJ_HASH_KEY_STRING to use the string length of the key.
  *hval*  if this argument is not NULL and the value is not zero, the value will be used as the computed hash value. If the argument is not NULL and the value is zero, it will be filled with the computed hash upon return.

**Returns:**

  the value associated with the key, or NULL if the key is not found.

6.13.4.7  **pj_hash_iterator_t**  \*ptr **pj_hash_next**( **pj_hash_table_t**  \*ht, **pj_hash_iterator_t**  \*it)

Get the next element from the iterator.

**Parameters:**

  *ht*  the hash table.
  *it*  the hash iterator.

**Returns:**

  the next iterator, or NULL if there’s no more element.

6.13.4.8  **void**  **pj_hash_set**( **pj_pool_t**  \*pool, **pj_hash_table_t**  \*ht, **const void**  \*key, **unsigned**  \*keylen, **pj_uint32_t**  \*hval, **void**  \*value)

Associate/disassociate a value with the specified key. If value is not NULL and entry already exists, the entry’s value will be overwritten. If value is not NULL and entry does not exist, a new one will be created with the specified pool. Otherwise if value is NULL, entry will be deleted if it exists.
Parameters:

- **pool** the pool to allocate the new entry if a new entry has to be created.
- **ht** the hash table.
- **key** the key, which MUST point to buffer that remains valid for the duration of the entry.
- **keylen** the length of the key, or PJ_HASH_KEY_STRING to use the string length of the key.
- **hval** if the value is not zero, then the hash table will use this value to search the entry’s index, otherwise it will compute the key. This value can be obtained when calling `pj_hash_get()`.
- **value** value to be associated, or NULL to delete the entry with the specified key.

6.13.4.9 **void** `pj_hash_set_np (pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t hval, pj_hash_entry_buf entry_buf, void *value)`

Associate/disassociate a value with the specified key. This function works like `pj_hash_set()`, except that it doesn’t use pool (hence the np – no pool suffix). If new entry needs to be allocated, it will use the entry_buf.

Parameters:

- **ht** the hash table.
- **key** the key.
- **keylen** the length of the key, or PJ_HASH_KEY_STRING to use the string length of the key.
- **hval** if the value is not zero, then the hash table will use this value to search the entry’s index, otherwise it will compute the key. This value can be obtained when calling `pj_hash_get()`.
- **entry_buf** Buffer which will be used for the new entry, when one needs to be created.
- **value** value to be associated, or NULL to delete the entry with the specified key.

6.13.4.10 **void** `pj_hash_this (pj_hash_table_t *ht, pj_hash_iterator_t *it)`

Get the value associated with a hash iterator.

Parameters:

- **ht** the hash table.
- **it** the hash iterator.

Returns:

the value associated with the current element in iterator.
6.14 Input/Output

6.14.1 Detailed Description

Input/Output.

This section contains API building blocks to perform network I/O and communications. It provides:

- **Socket Abstraction**
  A highly portable socket abstraction, runs on all kind of network APIs such as standard BSD socket, Windows socket, Linux *kernel* socket, PalmOS networking API, etc.

- **Network Address Resolution**
  Portable address resolution, which implements `pj_gethostbyname()`.

- **Socket select() API.**
  A portable `select()` like API (`pj_sock_select()`) which can be implemented with various back-ends.

- **IOQueue: I/O Event Dispatching with Proactor Pattern**
  Framework for dispatching network events.

For more information see the modules below.

**Modules**

- **Network Address Resolution**
- **File Access**
- **File I/O**
- **IOQueue: I/O Event Dispatching with Proactor Pattern**
- **IP Interface and Routing Helper**
- **Socket Abstraction**
- **Socket select() API**.
6.15 IOQueue: I/O Event Dispatching with Proactor Pattern

6.15.1 Detailed Description

I/O Queue provides API for performing asynchronous I/O operations. It conforms to proactor pattern, which allows application to submit an asynchronous operation and to be notified later when the operation has completed.

The I/O Queue can work on both socket and file descriptors. For asynchronous file operations however, one must make sure that the correct file I/O back-end is used, because not all file I/O back-end can be used with the ioqueue. Please see File I/O for more details.

The framework works natively in platforms where asynchronous operation API exists, such as in Windows NT with IoCompletionPort/IOCP. In other platforms, the I/O queue abstracts the operating system’s event poll API to provide semantics similar to IoCompletionPort with minimal penalties (i.e. per ioqueue and per handle mutex protection).

The I/O queue provides more than just unified abstraction. It also:

- makes sure that the operation uses the most effective way to utilize the underlying mechanism, to achieve the maximum theoretical throughput possible on a given platform.
- choose the most efficient mechanism for event polling on a given platform.

Currently, the I/O Queue is implemented using:

- **select()**, as the common denominator, but the least efficient. Also the number of descriptor is limited to PJ_IOQUEUE_MAX_HANDLES (which by default is 64).

- **/dev/epoll** on Linux (user mode and kernel mode), a much faster replacement for select() on Linux (and more importantly doesn’t have limitation on number of descriptors).

- **I/O Completion ports** on Windows NT/2000/XP, which is the most efficient way to dispatch events in Windows NT based OSes, and most importantly, it doesn’t have the limit on how many handles to monitor. And it works with files (not only sockets) as well.

6.15.2 Concurrency Rules

The items below describe rules that must be obeyed when using the I/O queue, with regard to concurrency:

- simultaneous operations (by different threads) to different key is safe.

- simultaneous operations to the same key is also safe, except **unregistration**, which is described below.

- **care must be taken when un registering a key** from the ioqueue. Application must take care that when one thread is issuing an unregistration, other thread is not simultaneously invoking an operation **to the same key**.

  This happens because the ioqueue functions are working with a pointer to the key, and there is a possible race condition where the pointer has been rendered invalid by other threads before the ioqueue has a chance to acquire mutex on it.
6.15 I/OQueue: I/O Event Dispatching with Proactor Pattern

6.15.3 Examples

For some examples on how to use the I/O Queue, please see:

- Test: I/O Queue (TCP)
- Test: I/O Queue (UDP)
- Test: I/O Queue Performance

Data Structures

- struct pj_ioqueue_op_key_t
- struct pj_ioqueue_callback

Defines

- #define PJ_IOQUEUE_MAX_EVENTS_IN_SINGLE_POLL (16)
- #define PJ_IOQUEUE_ALWAYS_ASYNC ((pj_uint32_t)1 << (pj_uint32_t)31)

Enumerations

- enum pj_ioqueue_operation_e {
  PJ_IOQUEUE_OP_NONE = 0, PJ_IOQUEUE_OP_READ = 1, PJ_IOQUEUE_OP_RECV = 2,
  PJ_IOQUEUE_OP_RECV_FROM = 4,
  PJ_IOQUEUE_OP_WRITE = 8, PJ_IOQUEUE_OP_SEND = 16, PJ_IOQUEUE_OP_SEND_TO =
  32, PJ_IOQUEUE_OP_ACCEPT = 64,
  PJ_IOQUEUE_OP_CONNECT = 128
}

Functions

- const char * pj_ioqueue_name (void)
- pj_status_t pj_ioqueue_create (pj_pool_t *pool, pj_size_t max_fd, pj_ioqueue_t **ioqueue)
- pj_status_t pj_ioqueue_destroy (pj_ioqueue_t *ioqueue)
- pj_status_t pj_ioqueue_set_lock (pj_ioqueue_t *ioqueue, pj_lock_t *lock, pj_bool_t auto_delete)
- pj_status_t pj_ioqueue_register_sock (pj_pool_t *pool, pj_ioqueue_t *ioqueue, pj_sock_t sock, void *
  user_data, const pj_ioqueue_callback *cb, pj_ioqueue_key_t **key)
- pj_status_t pj_ioqueue_unregister (pj_ioqueue_key_t *key)
- void * pj_ioqueue_get_user_data (pj_ioqueue_key_t *key)
- pj_status_t pj_ioqueue_set_user_data (pj_ioqueue_key_t *key, void *user_data, void **old_data)
- void pj_ioqueue_op_key_init (pj_ioqueue_op_key_t *op_key, pj_size_t size)
- pj_bool_t pj_ioqueue_is_pending (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key)
- pj_status_t pj_ioqueue_post_completion (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key,
  pj_ssize_t bytes_status)
- pj_status_t pj_ioqueue_accept (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_sock_t
  *new_sock, pj_sockaddr_t *local, pj_sockaddr_t *remote, int *addrlen)
- pj_status_t pj_ioqueue_connect (pj_ioqueue_key_t *key, const pj_sockaddr_t *addr, int addrlen)
- int pj_ioqueue_poll (pj_ioqueue_t *ioqueue, const pj_time_val *timeout)
pj_status_t pj_ioqueue_recv (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, void *buffer, pj_ssize_t *length, pj_uint32_t *flags)
pj_status_t pj_ioqueue_recvfrom (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, void *buffer, pj_ssize_t *length, pj_uint32_t *flags, pj_sockaddr_t *addr, int *addrlen)
pj_status_t pj_ioqueue_send (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, const void *data, pj_ssize_t *length, pj_uint32_t *flags)
pj_status_t pj_ioqueue_sendto (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, const void *data, pj_ssize_t *length, pj_uint32_t *flags, const pj_sockaddr_t *addr, int addrlen)

Variables
- void *pj_ioqueue_op_key_t::user_data
- void(* pj_ioqueue_callback::on_write_complete ) (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_ssize_t bytes_sent)
- void(* pj_ioqueue_callback::on_accept_complete ) (pj_ioqueue_key_t *key, pj_status_t status)
- void(* pj_ioqueue_callback::on_connect_complete ) (pj_ioqueue_key_t *key, pj_status_t status)

6.15.4 Define Documentation
6.15.4.1 #define PJ_IOQUEUE_ALWAYS_ASYNC ((pj_uint32_t)1 << (pj_uint32_t)31)
When this flag is specified in ioqueue’s recv() or send() operations, the ioqueue will always mark the operation as asynchronous.

6.15.4.2 #define PJ_IOQUEUE_MAX_EVENTS_IN_SINGLE_POLL (16)
This macro specifies the maximum number of events that can be processed by the ioqueue on a single poll cycle, on implementation that supports it. The value is only meaningful when specified during PJLIB build.

6.15.5 Enumeration Type Documentation
6.15.5.1 enum pj_ioqueue_operation_e
Types of pending I/O Queue operation. This enumeration is only used internally within the ioqueue.

 Enumerator:

PJ_IOQUEUE_OP_NONE No operation.
PJ_IOQUEUE_OP_READ read() operation.
PJ_IOQUEUE_OP_RECV recv() operation.
PJ_IOQUEUE_OP_RECV_FROM recvfrom() operation.
PJ_IOQUEUE_OP_WRITE write() operation.
PJ_IOQUEUE_OP_SEND send() operation.
PJ_IOQUEUE_OP_SEND_TO sendto() operation.
PJ_IOQUEUE_OP_ACCEPT accept() operation.
PJ_IOQUEUE_OP_CONNECT connect() operation.
6.15.6 Function Documentation

6.15.6.1 `pj_status_t pj_ioqueue_accept (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_sock_t *new_sock, pj_sockaddr_t *local, pj_sockaddr_t *remote, int *addrlen)`

Instruct I/O Queue to accept incoming connection on the specified listening socket. This function will return immediately (i.e. non-blocking) regardless whether a connection is immediately available. If the function can’t complete immediately, the caller will be notified about the incoming connection when it calls `pj_ioqueue_poll()`. If a new connection is immediately available, the function returns PJ_SUCCESS with the new connection; in this case, the callback WILL NOT be called.

Parameters:

- **key** The key which registered to the server socket.
- **op_key** An operation specific key to be associated with the pending operation, so that application can keep track of which operation has been completed when the callback is called.
- **new_sock** Argument which contain pointer to receive the new socket for the incoming connection.
- **local** Optional argument which contain pointer to variable to receive local address.
- **remote** Optional argument which contain pointer to variable to receive the remote address.
- **addrlen** On input, contains the length of the buffer for the address, and on output, contains the actual length of the address. This argument is optional.

Returns:

- PJ_SUCCESS When connection is available immediately, and the parameters will be updated to contain information about the new connection. In this case, a completion callback WILL NOT be called.
- PJ_EPENDING If no connection is available immediately. When a new connection arrives, the callback will be called.
- non-zero which indicates the appropriate error code.

6.15.6.2 `pj_status_t pj_ioqueue_connect (pj_ioqueue_key_t *key, const pj_sockaddr_t *addr, int addrlen)`

Initiate non-blocking socket connect. If the socket can NOT be connected immediately, asynchronous connect() will be scheduled and caller will be notified via completion callback when it calls `pj_ioqueue_poll()`. If socket is connected immediately, the function returns PJ_SUCCESS and completion callback WILL NOT be called.

Parameters:

- **key** The key associated with TCP socket
- **addr** The remote address.
- **addrlen** The remote address length.

Returns:

- PJ_SUCCESS If socket is connected immediately. In this case, the completion callback WILL NOT be called.
- PJ_EPENDING If operation is queued, or
- non-zero Indicates the error code.
6.15.6.3  

**pj_status_t pj_ioqueue_create (pj_pool_t * pool, pj_size_t max_fd, pj_ioqueue_t ** ioqueue)**

Create a new I/O Queue framework.

**Parameters:**
- *pool*  The pool to allocate the I/O queue structure.
- *max_fd*  The maximum number of handles to be supported, which should not exceed PJ_IOQUEUE_MAX_HANDLES.
- *ioqueue*  Pointer to hold the newly created I/O Queue.

**Returns:**
- PJ_SUCCESS on success.

6.15.6.4  

**pj_status_t pj_ioqueue_destroy (pj_ioqueue_t * ioque)**

Destroy the I/O queue.

**Parameters:**
- *ioque*  The I/O Queue to be destroyed.

**Returns:**
- PJ_SUCCESS if success.

6.15.6.5  

**void* pj_ioqueue_get_user_data (pj_ioqueue_key_t * key)**

Get user data associated with an ioqueue key.

**Parameters:**
- *key*  The key that was previously obtained from registration.

**Returns:**
- The user data associated with the descriptor, or NULL on error or if no data is associated with the key during registration.

6.15.6.6  

**pj_bool_t pj_ioqueue_is_pending (pj_ioqueue_key_t * key, pj_ioqueue_op_key_t * op_key)**

Check if operation is pending on the specified operation key. The *op_key* must have been initialized with `pj_ioqueue_op_key_init()` or submitted as pending operation before, or otherwise the result is undefined.

**Parameters:**
- *key*  The key.
**op_key** The operation key, previously submitted to any of the I/O functions and has returned PJ_EPENDING.

**Returns:**
Non-zero if operation is still pending.

### 6.15.6.7 const char∗ pj_ioqueue_name (void)

Return the name of the ioqueue implementation.

**Returns:**
Implementation name.

### 6.15.6.8 void pj_ioqueue_op_key_init (pj_ioqueue_op_key_t ∗ op_key, pj_size_t size)

Initialize operation key.

**Parameters:**
- **op_key** The operation key to be initialized.
- **size** The size of the operation key.

### 6.15.6.9 int pj_ioqueue_poll (pj_ioqueue_t ∗ ioque, const pj_time_val ∗ timeout)

Poll the I/O Queue for completed events.

**Parameters:**
- **ioque** the I/O Queue.
- **timeout** polling timeout, or NULL if the thread wishes to wait indefinitely for the event.

**Returns:**
- zero if timed out (no event).
- (≤0) if error occurred during polling. Callback will NOT be called.
- (>1) to indicate numbers of events. Callbacks have been called.

### 6.15.6.10 pj_status_t pj_ioqueue_post_completion (pj_ioqueue_key_t ∗ key, pj_ioqueue_op_key_t ∗ op_key, pj_ssize_t bytes_status)

Post completion status to the specified operation key and call the appropriate callback. When the callback is called, the number of bytes received in read/write callback or the status in accept/connect callback will be set from the bytes_status parameter.

**Parameters:**
- **key** The key.
**op_key** Pending operation key.

**bytes_status** Number of bytes or status to be set. A good value to put here is -PJ_ECANCELLED.

Returns:

PJ_SUCCESS if completion status has been successfully sent.

### 6.15.6.11 pj_status_t pj_ioqueue_recv (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, void *buffer, pj_ssize_t *length, pj_uint32_t flags)

Instruct the I/O Queue to read from the specified handle. This function returns immediately (i.e., non-blocking) regardless whether some data has been transferred. If the operation can’t complete immediately, caller will be notified about the completion when it calls pj_ioqueue_poll(). If data is immediately available, the function will return PJ_SUCCESS and the callback WILL NOT be called.

Parameters:

- **key** The key that uniquely identifies the handle.
- **op_key** An operation specific key to be associated with the pending operation, so that application can keep track of which operation has been completed when the callback is called. Caller must make sure that this key remains valid until the function completes.
- **buffer** The buffer to hold the read data. The caller MUST make sure that this buffer remain valid until the framework completes reading the handle.
- **length** On input, it specifies the size of the buffer. If data is available to be read immediately, the function returns PJ_SUCCESS and this argument will be filled with the amount of data read. If the function is pending, caller will be notified about the amount of data read in the callback. This parameter can point to local variable in caller’s stack and doesn’t have to remain valid for the duration of pending operation.
- **flags** Recv flag. If flags has PJ_IOQUEUE_ALWAYS_ASYNC then the function will never return PJ_SUCCESS.

Returns:

- PJ_SUCCESS If immediate data has been received in the buffer. In this case, the callback WILL NOT be called.
- PJ_EPENDING If the operation has been queued, and the callback will be called when data has been received.
- non-zero The return value indicates the error code.

### 6.15.6.12 pj_status_t pj_ioqueue_recvfrom (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, void *buffer, pj_ssize_t *length, pj_uint32_t flags, pj_sockaddr_t *addr, int *addrlen)

This function behaves similarly as pj_ioqueue_recv(), except that it is normally called for socket, and the remote address will also be returned along with the data. Caller MUST make sure that both buffer and addr remain valid until the framework completes reading the data.

Parameters:

- **key** The key that uniquely identifies the handle.
op_key An operation specific key to be associated with the pending operation, so that application can keep track of which operation has been completed when the callback is called.

buffer The buffer to hold the read data. The caller MUST make sure that this buffer remain valid until the framework completes reading the handle.

length On input, it specifies the size of the buffer. If data is available to be read immediately, the function returns PJ_SUCCESS and this argument will be filled with the amount of data read. If the function is pending, caller will be notified about the amount of data read in the callback. This parameter can point to local variable in caller’s stack and doesn’t have to remain valid for the duration of pending operation.

flags Recv flag. If flags has PJ_IOQUEUE_ALWAYS_ASYNC then the function will never return PJ_SUCCESS.

addr Optional Pointer to buffer to receive the address.

addrlen On input, specifies the length of the address buffer. On output, it will be filled with the actual length of the address. This argument can be NULL if addr is not specified.

Returns:

• PJ_SUCCESS If immediate data has been received. In this case, the callback must have been called before this function returns, and no pending operation is scheduled.
• PJ_EPENDING If the operation has been queued.
• non-zero The return value indicates the error code.

6.15.6.13 pj_status_t pj_ioqueue_register_sock (pj_pool_t * pool, pj_ioqueue_t * ioue, pj_sock_t * sock, void * user_data, const pj_ioqueue_callback * cb, pj_ioqueue_key_t ** key)

Register a socket to the I/O queue framework. When a socket is registered to the IOQueue, it may be modified to use non-blocking IO. If it is modified, there is no guarantee that this modification will be restored after the socket is unregistered.

Parameters:

pool To allocate the resource for the specified handle, which must be valid until the handle/key is unregistered from I/O Queue.

ioue The I/O Queue.

sock The socket.

user_data User data to be associated with the key, which can be retrieved later.

cb Callback to be called when I/O operation completes.

key Pointer to receive the key to be associated with this socket. Subsequent I/O queue operation will need this key.

Returns:

PJ_SUCCESS on success, or the error code.

6.15.6.14 pj_status_t pj_ioqueue_send (pj_ioqueue_key_t * key, pj_ioqueue_op_key_t * op_key, const void * data, pj_ssize_t * length, pj_uint32_t flags)

Instruct the I/O Queue to write to the handle. This function will return immediately (i.e. non-blocking) regardless whether some data has been transferred. If the function can’t complete immediately, the caller will be notified about the completion when it calls pj_ioqueue_poll(). If operation completes immediately and data has been transferred, the function returns PJ_SUCCESS and the callback will NOT be called.
Parameters:

key The key that identifies the handle.

op_key An operation specific key to be associated with the pending operation, so that application can keep track of which operation has been completed when the callback is called.

data The data to send. Caller MUST make sure that this buffer remains valid until the write operation completes.

length On input, it specifies the length of data to send. When data was sent immediately, this function returns PJ_SUCCESS and this parameter contains the length of data sent. If data can not be sent immediately, an asynchronous operation is scheduled and caller will be notified via callback the number of bytes sent. This parameter can point to local variable on caller’s stack and doesn’t have to remain valid until the operation has completed.

flags Send flags. If flags has PJ_IOQUEUE_ALWAYS_ASYNC then the function will never return PJ_SUCCESS.

Returns:

- PJ_SUCCESS If data was immediately transfered. In this case, no pending operation has been scheduled and the callback WILL NOT be called.
- PJ_EPENDING If the operation has been queued. Once data base been transfered, the callback will be called.
- non-zero The return value indicates the error code.

6.15.6.15 pj_status_t pj_ioqueue_sendto (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key,
const void *data, pj_ssize_t *length, pj_uint32_t flags, const pj_sockaddr_t *addr, int addrlen)

Instruct the I/O Queue to write to the handle. This function will return immediately (i.e. non-blocking) regardless whether some data has been transferred. If the function can’t complete immediately, the caller will be notified about the completion when it calls pj_ioqueue_poll(). If operation completes immediately and data has been transfered, the function returns PJ_SUCCESS and the callback will NOT be called.

Parameters:

key the key that identifies the handle.

op_key An operation specific key to be associated with the pending operation, so that application can keep track of which operation has been completed when the callback is called.

data the data to send. Caller MUST make sure that this buffer remains valid until the write operation completes.

length On input, it specifies the length of data to send. When data was sent immediately, this function returns PJ_SUCCESS and this parameter contains the length of data sent. If data can not be sent immediately, an asynchronous operation is scheduled and caller will be notified via callback the number of bytes sent. This parameter can point to local variable on caller’s stack and doesn’t have to remain valid until the operation has completed.

flags send flags. If flags has PJ_IOQUEUE_ALWAYS_ASYNC then the function will never return PJ_SUCCESS.

addr Optional remote address.

addrlen Remote address length, addr is specified.

Returns:

- PJ_SUCCESS If data was immediately written.
• PJ_EPENDING If the operation has been queued.
• non-zero The return value indicates the error code.

6.15.6.16  **pj_status_t pj_ioqueue_set_lock (pj_ioqueue_t *ioque, pj_lock_t *lock, pj_bool_t auto_delete)**

Set the lock object to be used by the I/O Queue. This function can only be called right after the I/O queue
is created, before any handle is registered to the I/O queue.

Initially the I/O queue is created with non-recursive mutex protection. Applications can supply alternative
lock to be used by calling this function.

Parameters:

  - *ioque* The ioqueue instance.
  - *lock* The lock to be used by the ioqueue.
  - *auto_delete* In non-zero, the lock will be deleted by the ioqueue.

Returns:  
PJ_SUCCESS or the appropriate error code.

6.15.6.17  **pj_status_t pj_ioqueue_set_user_data (pj_ioqueue_key_t *key, void *user_data, void **old_data)**

Set or change the user data to be associated with the file descriptor or handle or socket descriptor.

Parameters:

  - *key* The key that was previously obtained from registration.
  - *user_data* User data to be associated with the descriptor.
  - *old_data* Optional parameter to retrieve the old user data.

Returns:  
PJ_SUCCESS on success or the error code.

6.15.6.18  **pj_status_t pj_ioqueue_unregister (pj_ioqueue_key_t *key)**

Unregister from the I/O Queue framework. Caller must make sure that the key doesn’t have any pending
operations before calling this function, by calling **pj_ioqueue_is_pending()** for all previously submitted
operations except asynchronous connect, and if necessary call **pj_ioqueue_post_completion()** to cancel the
pending operations.

Note that asynchronous connect operation will automatically be cancelled during the unregistration.
Also note that when I/O Completion Port backend is used, application MUST close the handle immediately
after unregistering the key. This is because there is no unregistering API for IOCP. The only way to
unregister the handle from IOCP is to close the handle.
Parameters:

- **key** The key that was previously obtained from registration.

Returns:

- PJ_SUCCESS on success or the error code.

See also:

- `pj_ioqueue_is_pending`

### 6.15.7 Variable Documentation

#### 6.15.7.1 void (* pj_ioqueue_callback::on_accept_complete)(pj_ioqueue_key_t *key,
                                                pj_ioqueue_op_key_t *op_key, pj_sock_t sock, pj_status_t status) [inherited]

This callback is called when `pj_ioqueue_accept` completes.

Parameters:

- **key** The key.
- **op_key** Operation key.
- **sock** Newly connected socket.
- **status** Zero if the operation completes successfully.

#### 6.15.7.2 void (* pj_ioqueue_callback::on_connect_complete)(pj_ioqueue_key_t *key, pj_status_t status) [inherited]

This callback is called when `pj_ioqueue_connect` completes.

Parameters:

- **key** The key.
- **status** PJ_SUCCESS if the operation completes successfully.

#### 6.15.7.3 void (* pj_ioqueue_callback::on_write_complete)(pj_ioqueue_key_t *key,
                                                            pj_ioqueue_op_key_t *op_key, pj_ssize_t bytes_sent) [inherited]

This callback is called when `pj_ioqueue_write` or `pj_ioqueue_sendto` completes.

Parameters:

- **key** The key.
- **op_key** Operation key.
- **bytes_sent** \( \geq 0 \) to indicate the amount of data written, otherwise negative value containing the error code. To obtain the pj_status_t error code, use (pj_status_t code = -bytes_sent).

#### 6.15.7.4 void* pj_ioqueue_op_key_t::user_data [inherited]

Application data.
6.16 IP Interface and Routing Helper

6.16.1 Detailed Description

This module provides functions to query local host’s IP interface and routing table.

Data Structures

- union pj_ip_route_entry

Functions

- pj_status_t pj_enum_ip_interface (unsigned *count, pj_in_addr ifs[ ])
- pj_status_t pj_enum_ip_route (unsigned *count, pj_ip_route_entry routes[ ])

6.16.2 Function Documentation

6.16.2.1 pj_status_t pj_enum_ip_interface (unsigned *count, pj_in_addr ifs[ ])

Enumerate the local IP interface currently active in the host.

Parameters:

- count On input, specify the number of entries. On output, it will be filled with the actual number of entries.
- ifs Array of IP addresses.

Returns:

PJ_SUCCESS on success, or the appropriate error code.

6.16.2.2 pj_status_t pj_enum_ip_route (unsigned *count, pj_ip_route_entry routes[ ])

Enumerate the IP routing table for this host.

Parameters:

- count On input, specify the number of routes entries. On output, it will be filled with the actual number of route entries.
- routes Array of IP routing entries.

Returns:

PJ_SUCCESS on success, or the appropriate error code.
6.17 Linked List

6.17.1 Detailed Description

List in PJLIB is implemented as doubly-linked list, and it won’t require dynamic memory allocation (just as all PJLIB data structures). The list here should be viewed more like a low level C list instead of high level C++ list (which normally are easier to use but require dynamic memory allocations), therefore all caveats with C list apply here too (such as you can NOT put a node in more than one lists).

6.17.2 Examples

See below for examples on how to manipulate linked list:

- Example: List Manipulation
- Test: Linked List

Data Structures

- struct pj_list

Defines

- #define PJ_DECL_LIST_MEMBER(type)

Functions

- void pj_list_init (pj_list_type *node)
- int pj_list_empty (const pj_list_type *node)
- void pj_list_insert_before (pj_list_type *pos, pj_list_type *node)
- void pj_list_push_back (pj_list_type *list, pj_list_type *node)
- void pj_list_insert_nodes_before (pj_list_type *list, pj_list_type *nodes)
- void pj_list_insert_after (pj_list_type *pos, pj_list_type *node)
- void pj_list_push_front (pj_list_type *list, pj_list_type *node)
- void pj_list_insert_nodes_after (pj_list_type *list, pj_list_type *nodes)
- void pj_list_merge_first (pj_list_type *list1, pj_list_type *list2)
- void pj_list_merge_last (pj_list_type *list1, pj_list_type *list2)
- void pj_list_erase (pj_list_type *node)
- pj_list_type * pj_list_find_node (pj_list_type *list, pj_list_type *node)
- pj_list_type * pj_list_search (pj_list_type *list, void *value, int(*comp)(void *value, const pj_list_type *node))
- pj_size_t pj_list_size (pj_list_type *list)

6.17.3 Define Documentation

6.17.3.1 #define PJ_DECL_LIST_MEMBER(type)

Use this macro in the start of the structure declaration to declare that the structure can be used in the linked list operation. This macro simply declares additional member prev and next to the structure.
6.17.4 Function Documentation

6.17.4.1 int pj_list_empty (const pj_list_type * node)

Check that the list is empty.

Parameters:
   
   node  The list head.

Returns:

   Non-zero if the list is not-empty, or zero if it is empty.

6.17.4.2 void pj_list_erase (pj_list_type * node)

Erase the node from the list it currently belongs.

Parameters:

   node  The element to be erased.

6.17.4.3 pj_list_type* pj_list_find_node (pj_list_type * list, pj_list_type * node)

Find node in the list.

Parameters:

   list  The list head.
   node  The node element to be searched.

Returns:

   The node itself if it is found in the list, or NULL if it is not found in the list.

6.17.4.4 void pj_list_init (pj_list_type * node)

Initialize the list. Initially, the list will have no member, and function pj_list_empty() will always return nonzero (which indicates TRUE) for the newly initialized list.

Parameters:

   node  The list head.

6.17.4.5 void pj_list_insert_after (pj_list_type * pos, pj_list_type * node)

Insert a node to the list after the specified element position.

Parameters:

   pos   The element in the list which will precede the inserted element.
The element to be inserted after the position element.

Returns:
void.

6.17.4.6 void pj_list_insert_before (pj_list_type * pos, pj_list_type * node)
Insert the node to the list before the specified element position.

Parameters:
    pos  The element to which the node will be inserted before.
    node The element to be inserted.

Returns:
void.

6.17.4.7 void pj_list_insert_nodes_after (pj_list_type * lst, pj_list_type * nodes)
Insert all nodes in nodes to the target list.

Parameters:
    lst  The target list.
    nodes Nodes list.

6.17.4.8 void pj_list_insert_nodes_before (pj_list_type * lst, pj_list_type * nodes)
Inserts all nodes in nodes to the target list.

Parameters:
    lst  The target list.
    nodes Nodes list.

6.17.4.9 void pj_list_merge_first (pj_list_type * list1, pj_list_type * list2)
Remove elements from the source list, and insert them to the destination list. The elements of the source list will occupy the front elements of the target list. Note that the node pointed by list2 itself is not considered as a node, but rather as the list descriptor, so it will not be inserted to the list1. The elements to be inserted starts at list2->next. If list2 is to be included in the operation, use pj_list_insert_nodes_before.

Parameters:
    list1  The destination list.
    list2  The source list.

Returns:
void.
6.17 Linked List

6.17.4.10 **void pj_list_merge_last** *(pj_list_type *list1, pj_list_type *list2)*

Remove elements from the second list argument, and insert them to the list in the first argument. The elements from the second list will be appended to the first list. Note that the node pointed by list2 itself is not considered as a node, but rather as the list descriptor, so it will not be inserted to the list1. The elements to be inserted starts at list2->next. If list2 is to be included in the operation, use pj_list_insert_nodes_-before.

**Parameters:**
- *list1* The element in the list which will precede the inserted element.
- *list2* The element in the list to be inserted.

**Returns:**
- void.

6.17.4.11 **void pj_list_push_back** *(pj_list_type *list, pj_list_type *node)*

Insert the node to the back of the list. This is just an alias for pj_list_insert_before().

**Parameters:**
- *list* The list.
- *node* The element to be inserted.

6.17.4.12 **void pj_list_push_front** *(pj_list_type *list, pj_list_type *node)*

Insert the node to the front of the list. This is just an alias for pj_list_insert_after().

**Parameters:**
- *list* The list.
- *node* The element to be inserted.

6.17.4.13 **pj_list_type* pj_list_search** *(pj_list_type *list, void *value, int(*)(void *value, const pj_list_type *node) comp)*

Search the list for the specified value, using the specified comparison function. This function iterates on nodes in the list, started with the first node, and call the user supplied comparison function until the comparison function returns ZERO.

**Parameters:**
- *list* The list head.
- *value* The user defined value to be passed in the comparison function
- *comp* The comparison function, which should return ZERO to indicate that the searched value is found.

**Returns:**
- The first node that matched, or NULL if it is not found.
6.17.4.14 \texttt{pj\_size\_t pj\_list\_size (pj\_list\_type * list)}

Traverse the list to get the number of elements in the list.

\textbf{Parameters:}

- \texttt{list} The list head.

\textbf{Returns:}

Number of elements.
6.18 Lock Objects

6.18.1 Detailed Description

Lock Objects are higher abstraction for different lock mechanisms. It offers the same API for manipulating different lock types (e.g. mutex, semaphores, or null locks). Because Lock Objects have the same API for different types of lock implementation, it can be passed around in function arguments. As the result, it can be used to control locking policy for a particular feature.

Functions

- \texttt{pj\_status\_t pj\_lock\_create\_simple\_mutex (pj\_pool\_t *pool, const char *name, pj\_lock\_t **lock)}
- \texttt{pj\_status\_t pj\_lock\_create\_recursive\_mutex (pj\_pool\_t *pool, const char *name, pj\_lock\_t **lock)}
- \texttt{pj\_status\_t pj\_lock\_create\_null\_mutex (pj\_pool\_t *pool, const char *name, pj\_lock\_t **lock)}
- \texttt{pj\_status\_t pj\_lock\_create\_semaphore (pj\_pool\_t *pool, const char *name, unsigned\ initialized, unsigned max, pj\_lock\_t **lock)}
- \texttt{pj\_status\_t pj\_lock\_acquire (pj\_lock\_t *lock)}
- \texttt{pj\_status\_t pj\_lock\_tryacquire (pj\_lock\_t *lock)}
- \texttt{pj\_status\_t pj\_lock\_release (pj\_lock\_t *lock)}
- \texttt{pj\_status\_t pj\_lock\_destroy (pj\_lock\_t *lock)}

6.18.2 Function Documentation

6.18.2.1 \texttt{pj\_status\_t pj\_lock\_acquire (pj\_lock\_t * lock)}

Acquire lock on the specified lock object.

Parameters:

- \textit{lock} The lock object.

Returns:

PJ_SUCCESS or the appropriate error code.

6.18.2.2 \texttt{pj\_status\_t pj\_lock\_create\_null\_mutex (pj\_pool\_t * pool, const char * name, pj\_lock\_t ** lock)}

Create NULL mutex. A NULL mutex doesn’t actually have any synchronization object attached to it.

Parameters:

- \textit{pool} Memory pool.
- \textit{name} Lock object’s name.
- \textit{lock} Pointer to store the returned handle.

Returns:

PJ_SUCCESS or the appropriate error code.
6.18.2.3  \texttt{pj\_status\_t pj\_lock\_create\_recursive\_mutex (pj\_pool\_t * pool, const char * name, pj\_lock\_t ** lock)}

Create recursive mutex lock object.

**Parameters:**

- \textit{pool}  Memory pool.
- \textit{name}  Lock object’s name.
- \textit{lock}  Pointer to store the returned handle.

**Returns:**

PJ\_SUCCESS or the appropriate error code.

6.18.2.4  \texttt{pj\_status\_t pj\_lock\_create\_semaphore (pj\_pool\_t * pool, const char * name, unsigned initial, unsigned max, pj\_lock\_t ** lock)}

Create semaphore lock object.

**Parameters:**

- \textit{pool}  Memory pool.
- \textit{name}  Lock object’s name.
- \textit{initial}  Initial value of the semaphore.
- \textit{max}  Maximum value of the semaphore.
- \textit{lock}  Pointer to store the returned handle.

**Returns:**

PJ\_SUCCESS or the appropriate error code.

6.18.2.5  \texttt{pj\_status\_t pj\_lock\_create\_simple\_mutex (pj\_pool\_t * pool, const char * name, pj\_lock\_t ** lock)}

Create simple, non recursive mutex lock object.

**Parameters:**

- \textit{pool}  Memory pool.
- \textit{name}  Lock object’s name.
- \textit{lock}  Pointer to store the returned handle.

**Returns:**

PJ\_SUCCESS or the appropriate error code.
6.18.2.6  \texttt{pj\_status\_t pj\_lock\_destroy (pj\_lock\_t * lock)}

Destroy the lock object.

\textbf{Parameters}:

\begin{itemize}
  \item \textit{lock}  The lock object.
\end{itemize}

\textbf{Returns}:

\begin{itemize}
  \item PJ\_SUCCESS or the appropriate error code.
\end{itemize}

6.18.2.7  \texttt{pj\_status\_t pj\_lock\_release (pj\_lock\_t * lock)}

Release lock on the specified lock object.

\textbf{Parameters}:

\begin{itemize}
  \item \textit{lock}  The lock object.
\end{itemize}

\textbf{Returns}:

\begin{itemize}
  \item PJ\_SUCCESS or the appropriate error code.
\end{itemize}

6.18.2.8  \texttt{pj\_status\_t pj\_lock\_tryacquire (pj\_lock\_t * lock)}

Try to acquire lock on the specified lock object.

\textbf{Parameters}:

\begin{itemize}
  \item \textit{lock}  The lock object.
\end{itemize}

\textbf{Returns}:

\begin{itemize}
  \item PJ\_SUCCESS or the appropriate error code.
\end{itemize}
6.19 Miscellaneous

Modules

- Assertion Macro
- ctype - Character Type
- Exception Handling
- Logging Facility
- Random Number Generator
- Timer Heap Management.

*The timer scheduling implementation here is based on ACE library’s ACE_Timer_Heap, with only little modification to suit our library’s style (I even left most of the comments in the original source).*

- Unicode Support
- Time Data Type and Manipulation.
6.20 Logging Facility

6.20.1 Detailed Description

The PJLIB logging facility is a configurable, flexible, and convenient way to write logging or trace information.

To write to the log, one uses constructs like below:

```c
... PJ_LOG(3, ("main.c", "Starting hello...");
... PJ_LOG(3, ("main.c", "Hello world from process %d", pj_getpid());
...```

In the above example, the number 3 controls the verbosity level of the information (which means "information", by convention). The string "main.c" specifies the source or sender of the message.

6.20.2 Examples

For examples, see:

- Example: Log, Hello World.

Defines

- `#define PJ_LOG(level, arg)`

Typedefs

- `typedef void pj_log_func (int level, const char *data, int len)`

Enumerations

- `enum pj_log_decoration {
    PJ_LOG_HAS_DAY_NAME = 1, PJ_LOG_HAS_YEAR = 2, PJ_LOG_HAS_MONTH = 4, PJ_LOG_HAS_DAY_OF_MON = 8,
    PJ_LOG_HAS_TIME = 16, PJ_LOG_HAS_MICRO_SEC = 32, PJ_LOG_HAS_SENDER = 64,
    PJ_LOG_HAS_NEWLINE = 128,
    PJ_LOG_HAS_CR = 256
  }`

Functions

- `void pj_log_write (int level, const char *buffer, int len)`
- `void pj_log (const char *sender, int level, const char *format, va_list marker)`
- `void pj_log_set_log_func (pj_log_func *func)`
- `pj_log_func * pj_log_get_log_func (void)`

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
• void pj_log_set_level (int level)
• int pj_log_get_level (void)
• void pj_log_set_decor (unsigned decor)
• unsigned pj_log_get_decor (void)

6.20.3 Define Documentation

6.20.3.1 #define PJ_LOG(level, arg)

Write log message. This is the main macro used to write text to the logging backend.

Parameters:

level The logging verbosity level. Lower number indicates higher importance, with level zero indicates fatal error. Only numeral argument is permitted (e.g. not variable).

arg Enclosed 'printf' like arguments, with the first argument is the sender, the second argument is format string and the following arguments are variable number of arguments suitable for the format string.

Sample:

PJ_LOG(2, (__FILE__, "current value is %d", value));

6.20.4 Typedef Documentation

6.20.4.1 typedef void pj_log_func (int level, const char ∗data, int len)

Signature for function to be registered to the logging subsystem to write the actual log message to some output device.

Parameters:

level Log level.

data Log message.

len Message length.

6.20.5 Enumeration Type Documentation

6.20.5.1 enum pj_log-decoration

Log decoration flag, to be specified with pj_log_set_decor().

Enumerator:

PJ_LOG_HAS_DAY_NAME Include day name [default: no],

PJ_LOG_HAS_YEAR Include year digit [default: no]

PJ_LOG_HAS_MONTH Include month [default: no]

PJ_LOG_HAS_DAY_OF_MON Include day of month [default: no]

PJ_LOG_HAS_TIME Include time [default: yes].
6.20 Logging Facility

PJ_LOG_HAS_MICRO_SEC  Include microseconds [yes]
PJ_LOG_HAS_SENDER     Include sender in the log [yes].
PJ_LOG_HAS_NEWLINE    Terminate each call with newline [yes].
PJ_LOG_HAS_CR        Include carriage return [no].

6.20.6 Function Documentation

6.20.6.1 void pj_log (const char *sender, int level, const char *format, va_list marker)

Write to log.

Parameters:

sender  Source of the message.
level   Verbosity level.
format  Format.
marker  Marker.

6.20.6.2 unsigned pj_log_get_decor (void)

Get current log decoration flag.

Returns:

Log decoration flag.

6.20.6.3 int pj_log_get_level (void)

Get current maximum log verbosity level.

Returns:

Current log maximum level.

6.20.6.4 pj_log_func* pj_log_get_log_func (void)

Get the current log output function that is used to write log messages.

Returns:

Current log output function.

6.20.6.5 void pj_log_set_decor (unsigned decor)

Set log decoration. The log decoration flag controls what are printed to output device alongside the actual message. For example, application can specify that date/time information should be displayed with each log message.

Parameters:

decor  Bitmask combination of pj_log_decoration to control the layout of the log message.
6.20.6.6 void pj_log_set_level (int level)

Set maximum log level. Application can call this function to set the desired level of verbosity of the logging messages. The bigger the value, the more verbose the logging messages will be printed. However, the maximum level of verbosity can not exceed compile time value of PJ_LOG_MAX_LEVEL.

Parameters:

- **level** The maximum level of verbosity of the logging messages (6=very detailed, 1=error only, 0=disabled)

6.20.6.7 void pj_log_set_log_func (pj_log_func ∗ func)

Change log output function. The front-end logging functions will call this function to write the actual message to the desired device. By default, the front-end functions use pj_log_write() to write the messages, unless it’s changed by calling this function.

Parameters:

- **func** The function that will be called to write the log messages to the desired device.

6.20.6.8 void pj_log_write (int level, const char ∗ buffer, int len)

Default logging writer function used by front end logger function. This function will print the log message to stdout only. Application normally should NOT need to call this function, but rather use the PJ_LOG macro.

Parameters:

- **level** Log level.
- **buffer** Log message.
- **len** Message length.
6.21 Operating System Dependent Functionality.

Modules

- Input/Output

- Lock Objects
- Symbian OS Specific
- Thread Local Storage.
- Atomic Variables
- Mutexes.
- Reader/Writer Mutex
- Critical sections.
- Semaphores.
- Event Object.
- High Resolution Timestamp
- Time Data Type and Manipulation.
6.22 Threads

6.22.1 Detailed Description

This module provides multithreading API.

6.22.2 Examples

For examples, please see:

• Test: Thread Test
• Test: Sleep, Time, and Timestamp

Defines

• #define PJ_THREAD_DESC_SIZE (64)
• #define PJ_CHECK_STACK()
• #define pj_thread_get_stack_max_usage(thread) 0
• #define pj_thread_get_stack_info(thread, f, l) (∗(f)=”,∗(l)=0)

Typedefs

• typedef long pj_thread_desc [64]

Enumerations

• enum pj_thread_create_flags { PJ_THREAD_SUSPENDED = 1 }

Functions

• typedef int (PJ_THREAD_FUNC pj_thread_proc)(void *)
• pj_uint32_t pj_getpid (void)
• pj_status_t pj_thread_create (pj_pool_t *pool, const char *thread_name, pj_thread_proc *proc, void *arg, pj_size_t stack_size, unsigned flags, pj_thread_t **thread)
• pj_status_t pj_thread_register (const char *thread_name, pj_thread_desc desc, pj_thread_t **thread)
• pj_bool_t pj_thread_is_registered (void)
• const char * pj_thread_get_name (pj_thread_t *thread)
• pj_status_t pj_thread_resume (pj_thread_t *thread)
• pj_thread_t * pj_thread_this (void)
• pj_status_t pj_thread_join (pj_thread_t *thread)
• pj_status_t pj_thread_destroy (pj_thread_t *thread)
• pj_status_t pj_thread_sleep (unsigned msec)
6.22 Threads

6.22.3 Define Documentation

6.22.3.1 #define PJ_CHECK_STACK()

PJ_CHECK_STACK() macro is used to check the sanity of the stack. The OS implementation may check that no stack overflow occurs, and it also may collect statistic about stack usage.

6.22.3.2 #define PJ_THREAD_DESC_SIZE (64)

Size of thread struct.

6.22.3.3 #define pj_thread_get_stack_info(thread, f, l) (∗(f)="",∗(l)=0)

pj_thread_get_stack_info() for the thread

6.22.3.4 #define pj_thread_get_stack_max_usage(thread) 0

pj_thread_get_stack_max_usage() for the thread

6.22.4 Typedef Documentation

6.22.4.1 typedef long pj_thread_desc[(64)]

Thread structure, to thread’s state when the thread is created by external or native API.

6.22.5 Enumeration Type Documentation

6.22.5.1 enum pj_thread_create_flags

Thread creation flags:

- PJ_THREAD_SUSPENDED: specify that the thread should be created suspended.

6.22.6 Function Documentation

6.22.6.1 typedef int (PJ THREAD FUNC pj_thread_proc)

Type of thread entry function.

6.22.6.2 pj_uint32_t pj_getpid (void)

Get process ID.

Returns:

process ID.
6.22.6.3  `pj_status_t pj_thread_create (pj_pool_t *pool, const char *thread_name, pj_thread_proc *proc, void *arg, pj_size_t stack_size, unsigned flags, pj_thread_t **thread)`

Create a new thread.

**Parameters:**

- `pool`  The memory pool from which the thread record will be allocated from.
- `thread_name`  The optional name to be assigned to the thread.
- `proc`  Thread entry function.
- `arg`  Argument to be passed to the thread entry function.
- `stack_size`  The size of the stack for the new thread, or ZERO or PJ_THREAD_DEFAULT_STACK_-SIZE to let the library choose the reasonable size for the stack. For some systems, the stack will be allocated from the pool, so the pool must have suitable capacity.
- `flags`  Flags for thread creation, which is bitmask combination from enum pj_thread_create_flags.
- `thread`  Pointer to hold the newly created thread.

**Returns:**

- PJ_SUCCESS on success, or the error code.

6.22.6.4  `pj_status_t pj_thread_destroy (pj_thread_t *thread)`

Destroy thread and release resources allocated for the thread. However, the memory allocated for the `pj_thread_t` itself will only be released when the pool used to create the thread is destroyed.

**Parameters:**

- `thread`  The thread handle.

**Returns:**

- zero on success.

6.22.6.5  `const char* pj_thread_get_name (pj_thread_t *thread)`

Get thread name.

**Parameters:**

- `thread`  The thread handle.

**Returns:**

- Thread name as null terminated string.

6.22.6.6  `pj_bool_t pj_thread_is_registered (void)`

Check if this thread has been registered to PJLIB.

**Returns:**

- Non-zero if it is registered.
6.22 Threads

6.22.6.7  

\texttt{pj\_status\_t pj\_thread\_join (pj\_thread\_t * thread)}

Join thread, and block the caller thread until the specified thread exits. If the specified thread has already
been dead, or it does not exist, the function will return immediately with successful status.

**Parameters:**
- \texttt{thread} The thread handle.

**Returns:**
- PJ_SUCCESS on success.

6.22.6.8  

\texttt{pj\_status\_t pj\_thread\_register (const char * thread\_name, pj\_thread\_desc desc, pj\_thread\_t ** thread)}

Register a thread that was created by external or native API to PJLIB. This function must be called in
the context of the thread being registered. When the thread is created by external function or API call, it
must be ‘registered’ to PJLIB using \texttt{pj\_thread\_register()}, so that it can cooperate with PJLIB’s framework.
During registration, some data needs to be maintained, and this data must remain available during the
thread’s lifetime.

**Parameters:**
- \texttt{thread\_name} The optional name to be assigned to the thread.
- \texttt{desc} Thread descriptor, which must be available throughout the lifetime of the thread.
- \texttt{thread} Pointer to hold the created thread handle.

**Returns:**
- PJ_SUCCESS on success, or the error code.

6.22.6.9  

\texttt{pj\_status\_t pj\_thread\_resume (pj\_thread\_t * thread)}

Resume a suspended thread.

**Parameters:**
- \texttt{thread} The thread handle.

**Returns:**
- zero on success.

6.22.6.10  

\texttt{pj\_status\_t pj\_thread\_sleep (unsigned msec)}

Put the current thread to sleep for the specified milliseconds.

**Parameters:**
- \texttt{msec} Miliseconds delay.

**Returns:**
- zero if successful.
6.22.6.11  `pj_thread_t* pj_thread_this (void)`

Get the current thread.

**Returns:**

Thread handle of current thread.
6.23 Symbian OS Specific

6.23.1 Detailed Description

Functionalities specific to Symbian OS.

Symbian OS strongly discourages the use of polling since this wastes CPU power, and instead provides Active Object and Active Scheduler pattern to allow application (in this case, PJLIB) to register asynchronous tasks. PJLIB port for Symbian complies to this recommended behavior. As the result, few things have been changed in PJLIB for Symbian:

- the timer heap (see Timer Heap Management) is implemented with active object framework, and each timer entry registered to the timer heap will register an Active Object to the Active Scheduler. Because of this, polling the timer heap with \texttt{pj_timer_heap_poll()} is no longer necessary, and this function will just evaluate to nothing.

- the iioqueue (see IOQueue: I/O Event Dispatching with Proactor Pattern) is also implemented with active object framework, with each asynchronous operation will register an Active Object to the Active Scheduler. Because of this, polling the iioqueue with \texttt{pj_iioqueue_poll()} is no longer necessary, and this function will just evaluate to nothing.

Since timer heap and iioqueue polling are no longer necessary, Symbian application can now poll for all events by calling \texttt{User::WaitForAnyRequest()} and \texttt{ CActiveScheduler::RunIfReady()}. PJLIB provides a thin wrapper which calls these two functions, called \texttt{pj_symbianos_poll()}.

Functions

- \texttt{pj_bool_t \texttt{pj_symbianos_poll} (int priority, int ms_timeout)}

6.23.2 Function Documentation

6.23.2.1 \texttt{pj_bool_t \texttt{pj_symbianos_poll} (int priority, int ms_timeout)}

Wait the completion of any Symbian active objects. When the timeout value is not specified (the \texttt{ms_timeout} argument is -1), this function is a thin wrapper which calls \texttt{User::WaitForAnyRequest()} and \texttt{ CActiveScheduler::RunIfReady()}. If the timeout value is specified, this function will schedule a timer entry to the timer heap (which is an Active Object), to limit the wait time for event occurrences. Scheduling a timer entry is an expensive operation, therefore application should only specify a timeout value when it’s really necessary (for example, when it’s not sure there are other Active Objects currently running in the application).

Parameters:

- \texttt{priority} The minimum priority of the Active Objects to poll, which values are from CActive::TPriority constants. If -1 is given, CActive::EPriorityStandard. priority will be used.

- \texttt{ms_timeout} Optional timeout to wait. Application should specify -1 to let the function wait indefinitely for any events.

Returns:

- PJ_TRUE if there have been any events executed during the polling. This function will only return PJ_FALSE if \texttt{ms_timeout} argument is specified (i.e. the value is not -1) and there was no event executed when the timeout timer elapsed.
6.24  Thread Local Storage.

Functions

• `pj_status_t pj_thread_local_alloc (long *index)`
• `void pj_thread_local_free (long index)`
• `pj_status_t pj_thread_local_set (long index, void *value)`
• `void * pj_thread_local_get (long index)`

6.24.1  Function Documentation

6.24.1.1  `pj_status_t pj_thread_local_alloc (long * index)`

Allocate thread local storage index. The initial value of the variable at the index is zero.

Parameters:

  `index`  Pointer to hold the return value.

Returns:

  PJ_SUCCESS on success, or the error code.

6.24.1.2  `void pj_thread_local_free (long index)`

Deallocate thread local variable.

Parameters:

  `index`  The variable index.

6.24.1.3  `void * pj_thread_local_get (long index)`

Get the value of thread local variable.

Parameters:

  `index`  The index of the variable.

Returns:

  The value.

6.24.1.4  `pj_status_t pj_thread_local_set (long index, void * value)`

Set the value of thread local variable.

Parameters:

  `index`  The index of the variable.

  `value`  The value.
6.25 Atomic Variables

6.25.1 Detailed Description

This module provides API to manipulate atomic variables.

6.25.2 Examples

For some example codes, please see:

- Test: Atomic Variable

Functions

- `pj_status_t pj_atomic_create (pj_pool_t *pool, pj_atomic_value_t initial, pj_atomic_t **atomic)`
- `pj_status_t pj_atomic_destroy (pj_atomic_t *atomic_var)`
- `void pj_atomic_set (pj_atomic_t *atomic_var, pj_atomic_value_t value)`
- `pj_atomic_value_t pj_atomic_get (pj_atomic_t *atomic_var)`
- `void pj_atomic_inc (pj_atomic_t *atomic_var)`
- `pj_atomic_value_t pj_atomic_inc_and_get (pj_atomic_t *atomic_var)`
- `void pj_atomic_dec (pj_atomic_t *atomic_var)`
- `pj_atomic_value_t pj_atomic_dec_and_get (pj_atomic_t *atomic_var)`
- `void pj_atomic_add (pj_atomic_t *atomic_var, pj_atomic_value_t value)`
- `pj_atomic_value_t pj_atomic_add_and_get (pj_atomic_t *atomic_var, pj_atomic_value_t value)`

6.25.3 Function Documentation

6.25.3.1 `void pj_atomic_add (pj_atomic_t *atomic_var, pj_atomic_value_t value)`

Add a value to an atomic type.

Parameters:

- `atomic_var` The atomic variable.
- `value` Value to be added.

6.25.3.2 `pj_atomic_value_t pj_atomic_add_and_get (pj_atomic_t *atomic_var, pj_atomic_value_t value)`

Add a value to an atomic type and get the result.

Parameters:

- `atomic_var` The atomic variable.
- `value` Value to be added.

Returns:

The result after the addition.
6.25.3.3  `pj_status_t pj_atomic_create (pj_pool_t * pool, pj_atomic_value_t initial, pj_atomic_t ** atomic)`  

Create atomic variable.  

**Parameters:**  
- pool The pool.  
- initial The initial value of the atomic variable.  
- atomic Pointer to hold the atomic variable upon return.  

**Returns:**  
- PJ_SUCCESS on success, or the error code.

6.25.3.4  `void pj_atomic_dec (pj_atomic_t * atomic_var)`  

Decrement the value of an atomic type.  

**Parameters:**  
- atomic_var the atomic variable.  

6.25.3.5  `pj_atomic_value_t pj_atomic_dec_and_get (pj_atomic_t * atomic_var)`  

Decrement the value of an atomic type and get the result.  

**Parameters:**  
- atomic_var the atomic variable.  

**Returns:**  
- The decremented value.

6.25.3.6  `pj_status_t pj_atomic_destroy (pj_atomic_t * atomic_var)`  

Destroy atomic variable.  

**Parameters:**  
- atomic_var the atomic variable.  

**Returns:**  
- PJ_SUCCESS if success.
6.25.3.7  `pj_atomic_value_t pj_atomic_get (pj_atomic_t * atomic_var)`

Get the value of an atomic type.

**Parameters:**

- `atomic_var` the atomic variable.

**Returns:**

the value of the atomic variable.

6.25.3.8  `void pj_atomic_inc (pj_atomic_t * atomic_var)`

Increment the value of an atomic type.

**Parameters:**

- `atomic_var` the atomic variable.

6.25.3.9  `pj_atomic_value_t pj_atomic_inc_and_get (pj_atomic_t * atomic_var)`

Increment the value of an atomic type and get the result.

**Parameters:**

- `atomic_var` the atomic variable.

**Returns:**

The incremented value.

6.25.3.10  `void pj_atomic_set (pj_atomic_t * atomic_var, pj_atomic_value_t value)`

Set the value of an atomic type, and return the previous value.

**Parameters:**

- `atomic_var` the atomic variable.
- `value` value to be set to the variable.
6.26 Mutexes.

6.26.1 Detailed Description

Mutex manipulation. Alternatively, application can use higher abstraction for lock objects, which provides uniform API for all kinds of lock mechanisms, including mutex. See Lock Objects for more information.

Defines

- #define pj_mutex_is_locked(mutex) 1

Enumerations

- enum pj_mutex_type_e { PJ_MUTEX_DEFAULT, PJ_MUTEX_SIMPLE, PJ_MUTEX_RECURSE }

Functions

- pj_status_t pj_mutex_create (pj_pool_t *pool, const char *name, int type, pj_mutex_t **mutex)
- pj_status_t pj_mutex_create_simple (pj_pool_t *pool, const char *name, pj_mutex_t **mutex)
- pj_status_t pj_mutex_create_recursive (pj_pool_t *pool, const char *name, pj_mutex_t **mutex)
- pj_status_t pj_mutex_lock (pj_mutex_t *mutex)
- pj_status_t pj_mutex_unlock (pj_mutex_t *mutex)
- pj_status_t pj_mutex_trylock (pj_mutex_t *mutex)
- pj_status_t pj_mutex_destroy (pj_mutex_t *mutex)

6.26.2 Define Documentation

6.26.2.1 #define pj_mutex_is_locked(mutex) 1

Determine whether calling thread is owning the mutex (only available when PJ_DEBUG is set).

Parameters:

- mutex The mutex.

Returns:

- Non-zero if yes.

6.26.3 Enumeration Type Documentation

6.26.3.1 enum pj_mutex_type_e

Mutex types:

- PJ_MUTEX_DEFAULT: default mutex type, which is system dependent.
- PJ_MUTEX_SIMPLE: non-recursive mutex.
- PJ_MUTEX_RECURSE: recursive mutex.
6.26 Mutexes.

6.26.4 Function Documentation

6.26.4.1  

\texttt{pj\_status\_t pj\_mutex\_create (pj\_pool\_t * pool, const char * name, int type, pj\_mutex\_t ** mutex)}

Create mutex of the specified type.

\textbf{Parameters:}

- \textit{pool}  The pool.
- \textit{name}  Name to be associated with the mutex (for debugging).
- \textit{type}  The type of the mutex, of type \texttt{pj\_mutex\_type\_e}.
- \textit{mutex}  Pointer to hold the returned mutex instance.

\textbf{Returns:}

PJ\_SUCCESS on success, or the error code.

6.26.4.2  

\texttt{pj\_status\_t pj\_mutex\_create\_recursive (pj\_pool\_t * pool, const char * name, pj\_mutex\_t ** mutex)}

Create recursive mutex. This function is a simple wrapper for \texttt{pj\_mutex\_create} to create recursive mutex.

\textbf{Parameters:}

- \textit{pool}  The pool.
- \textit{name}  Mutex name.
- \textit{mutex}  Pointer to hold the returned mutex instance.

\textbf{Returns:}

PJ\_SUCCESS on success, or the error code.

6.26.4.3  

\texttt{pj\_status\_t pj\_mutex\_create\_simple (pj\_pool\_t * pool, const char * name, pj\_mutex\_t ** mutex)}

Create simple, non-recursive mutex. This function is a simple wrapper for \texttt{pj\_mutex\_create} to create non-recursive mutex.

\textbf{Parameters:}

- \textit{pool}  The pool.
- \textit{name}  Mutex name.
- \textit{mutex}  Pointer to hold the returned mutex instance.

\textbf{Returns:}

PJ\_SUCCESS on success, or the error code.
6.26.4.4  pj_status_t pj_mutex_destroy (pj_mutex_t *mutex)

Destroy mutex.

Parameters:
  
  mutex  The mutex.

Returns:
  
  PJ_SUCCESS on success, or the error code.

6.26.4.5  pj_status_t pj_mutex_lock (pj_mutex_t *mutex)

Acquire mutex lock.

Parameters:
  
  mutex  The mutex.

Returns:
  
  PJ_SUCCESS on success, or the error code.

6.26.4.6  pj_status_t pj_mutex_trylock (pj_mutex_t *mutex)

Try to acquire mutex lock.

Parameters:
  
  mutex  The mutex.

Returns:
  
  PJ_SUCCESS on success, or the error code if the lock couldn’t be acquired.

6.26.4.7  pj_status_t pj_mutex_unlock (pj_mutex_t *mutex)

Release mutex lock.

Parameters:
  
  mutex  The mutex.

Returns:
  
  PJ_SUCCESS on success, or the error code.
6.27 Reader/Writer Mutex

6.27.1 Detailed Description

Reader/writer mutex is a classic synchronization object where multiple readers can acquire the mutex, but only a single writer can acquire the mutex.

Typedefs

- typedef pj_rwlock_t pj_rwlock_t

Functions

- pj_status_t pj_rwlock_create (pj_pool_t *pool, const char *name, pj_rwlock_t **mutex)
- pj_status_t pj_rwlock_lock_read (pj_rwlock_t *mutex)
- pj_status_t pj_rwlock_lock_write (pj_rwlock_t *mutex)
- pj_status_t pj_rwlock_unlock_read (pj_rwlock_t *mutex)
- pj_status_t pj_rwlock_unlock_write (pj_rwlock_t *mutex)
- pj_status_t pj_rwlock_destroy (pj_rwlock_t *mutex)

6.27.2 Typedef Documentation

6.27.2.1 typedef struct pj_rwlock_t pj_rwlock_t

Opaque declaration for reader/writer mutex. Reader/writer mutex is a classic synchronization object where multiple readers can acquire the mutex, but only a single writer can acquire the mutex.

6.27.3 Function Documentation

6.27.3.1 pj_status_t pj_rwlock_create (pj_pool_t *pool, const char * name, pj_rwlock_t **mutex)

Create reader/writer mutex.

Parameters:

- pool Pool to allocate memory for the mutex.
- name Name to be assigned to the mutex.
- mutex Pointer to receive the newly created mutex.

Returns:

- PJ_SUCCESS on success, or the error code.
6.27.3.2 pj_status_t pj_rwlockmutex_destroy (pj_rwlockmutex_t * mutex)

Destroy reader/writer mutex.

Parameters:

 mutex The mutex.

Returns:

 PJ_SUCCESS on success, or the error code.

6.27.3.3 pj_status_t pj_rwlockmutex_lock_read (pj_rwlockmutex_t * mutex)

Lock the mutex for reading.

Parameters:

 mutex The mutex.

Returns:

 PJ_SUCCESS on success, or the error code.

6.27.3.4 pj_status_t pj_rwlockmutex_lock_write (pj_rwlockmutex_t * mutex)

Lock the mutex for writing.

Parameters:

 mutex The mutex.

Returns:

 PJ_SUCCESS on success, or the error code.

6.27.3.5 pj_status_t pj_rwlockmutex_unlock_read (pj_rwlockmutex_t * mutex)

Release read lock.

Parameters:

 mutex The mutex.

Returns:

 PJ_SUCCESS on success, or the error code.
6.27 Reader/Writer Mutex

6.27.3.6  

**pj_status_t pj_rwlockmutex_unlock_write (pj_rwlockmutex_t * mutex)**

Release write lock.

**Parameters:**

*mutex*  The mutex.

**Returns:**

PJ_SUCCESS on success, or the error code.
6.28 Critical sections.

6.28.1 Detailed Description

Critical section protection can be used to protect regions where:

- mutual exclusion protection is needed.
- it’s rather too expensive to create a mutex.
- the time spent in the region is very very brief.

Critical section is a global object, and it prevents any threads from entering any regions that are protected by critical section once a thread is already in the section.

Critical section is not recursive!

Application MUST NOT call any functions that may cause current thread to block (such as allocating memory, performing I/O, locking mutex, etc.) while holding the critical section.

Functions

- void pj_enter_critical_section (void)
- void pj_leave_critical_section (void)

6.28.2 Function Documentation

6.28.2.1 void pj_enter_critical_section (void)

Enter critical section.

6.28.2.2 void pj_leave_critical_section (void)

Leave critical section.
6.29 Semaphores.

6.29.1 Detailed Description

This module provides abstraction for semaphores, where available.

Functions

- \texttt{pj_status_t pj\_sem\_create (pj\_pool\_t \ast pool, \texttt{const char \ast name, unsigned initial, unsigned max, pj\_sem\_t \ast\ast sem})}
- \texttt{pj_status_t pj\_sem\_wait (pj\_sem\_t \ast sem)}
- \texttt{pj_status_t pj\_sem\_trywait (pj\_sem\_t \ast sem)}
- \texttt{pj_status_t pj\_sem\_post (pj\_sem\_t \ast sem)}
- \texttt{pj_status_t pj\_sem\_destroy (pj\_sem\_t \ast sem)}

6.29.2 Function Documentation

6.29.2.1 \texttt{pj_status_t pj\_sem\_create (pj\_pool\_t \ast pool, \texttt{const char \ast name, unsigned initial, unsigned max, pj\_sem\_t \ast\ast sem})}

Create semaphore.

Parameters:

\begin{itemize}
  \item \texttt{pool} The pool.
  \item \texttt{name} Name to be assigned to the semaphore (for logging purpose)
  \item \texttt{initial} The initial count of the semaphore.
  \item \texttt{max} The maximum count of the semaphore.
  \item \texttt{sem} Pointer to hold the semaphore created.
\end{itemize}

Returns:

- PJ\_SUCCESS on success, or the error code.

6.29.2.2 \texttt{pj_status_t pj\_sem\_destroy (pj\_sem\_t \ast sem)}

Destroy semaphore.

Parameters:

\begin{itemize}
  \item \texttt{sem} The semaphore.
\end{itemize}

Returns:

- PJ\_SUCCESS on success, or the error code.
6.29.2.3  
\texttt{pj\_status\_t pj\_sem\_post (pj\_sem\_t * sem)}

Release semaphore.

**Parameters:**

sem\textsuperscript{1} The semaphore.

**Returns:**

PJ\_SUCCESS on success, or the error code.

6.29.2.4  
\texttt{pj\_status\_t pj\_sem\_trywait (pj\_sem\_t * sem)}

Try wait for semaphore.

**Parameters:**

sem\textsuperscript{1} The semaphore.

**Returns:**

PJ\_SUCCESS on success, or the error code.

6.29.2.5  
\texttt{pj\_status\_t pj\_sem\_wait (pj\_sem\_t * sem)}

Wait for semaphore.

**Parameters:**

sem\textsuperscript{1} The semaphore.

**Returns:**

PJ\_SUCCESS on success, or the error code.
6.30 Event Object.

6.30.1 Detailed Description

This module provides abstraction to event object (e.g. Win32 Event) where available. Event objects can be used for synchronization among threads.

Functions

- **pj_status_t pj_event_create (pj_pool_t *pool, const char *name, pj_bool_t manual_reset, pj_bool_t initial, pj_event_t **event)**
- **pj_status_t pj_event_wait (pj_event_t *event)**
- **pj_status_t pj_event_trywait (pj_event_t *event)**
- **pj_status_t pj_event_set (pj_event_t *event)**
- **pj_status_t pj_event_pulse (pj_event_t *event)**
- **pj_status_t pj_event_reset (pj_event_t *event)**
- **pj_status_t pj_event_destroy (pj_event_t *event)**

6.30.2 Function Documentation

6.30.2.1 **pj_status_t pj_event_create (pj_pool_t *pool, const char *name, pj_bool_t manual_reset, pj_bool_t initial, pj_event_t **event)**

Create event object.

**Parameters:**

- **pool** The pool.
- **name** The name of the event object (for logging purpose).
- **manual_reset** Specify whether the event is manual-reset
- **initial** Specify the initial state of the event object.
- **event** Pointer to hold the returned event object.

**Returns:**

- event handle, or NULL if failed.

6.30.2.2 **pj_status_t pj_event_destroy (pj_event_t *event)**

Destroy the event object.

**Parameters:**

- **event** The event object.

**Returns:**

- zero if successfull.
6.30.2.3 \texttt{pj\_status\_t pj\_event\_pulse (pj\_event\_t * event)}

Set the event object to signaled state to release appropriate number of waiting threads and then reset the event object to non-signaled. For manual-reset event, this function will release all waiting threads. For auto-reset event, this function will only release one waiting thread.

\textbf{Parameters:}

- \emph{event} The event object.

\textbf{Returns:}

- zero if successful.

6.30.2.4 \texttt{pj\_status\_t pj\_event\_reset (pj\_event\_t * event)}

Set the event object state to non-signaled.

\textbf{Parameters:}

- \emph{event} The event object.

\textbf{Returns:}

- zero if successful.

6.30.2.5 \texttt{pj\_status\_t pj\_event\_set (pj\_event\_t * event)}

Set the event object state to signaled. For auto-reset event, this will only release the first thread that are waiting on the event. For manual reset event, the state remains signaled until the event is reset. If there is no thread waiting on the event, the event object state remains signaled.

\textbf{Parameters:}

- \emph{event} The event object.

\textbf{Returns:}

- zero if successful.

6.30.2.6 \texttt{pj\_status\_t pj\_event\_trywait (pj\_event\_t * event)}

Try wait for event object to be signalled.

\textbf{Parameters:}

- \emph{event} The event object.

\textbf{Returns:}

- zero if successful.
6.30 Event Object.

6.30.2.7  `pj_status_t pj_event_wait (pj_event_t * event)`

Wait for event to be signaled.

**Parameters:**

- `event`  The event object.

**Returns:**

- zero if successful.
6.31 High Resolution Timestamp

6.31.1 Detailed Description

PJLIB provides **High Resolution Timestamp** API to access highest resolution timestamp value provided by the platform. The API is useful to measure precise elapsed time, and can be used in applications such as profiling.

The timestamp value is represented in cycles, and can be related to normal time (in seconds or sub-seconds) using various functions provided.

6.31.2 Examples

For examples, please see:

- Test: Sleep, Time, and Timestamp
- Test: Timestamp

Functions

- `pj_status_t pj_get_timestamp (pj_timestamp *ts)`
- `pj_status_t pj_get_timestamp_freq (pj_timestamp *freq)`
- `void pj_set_timestamp32 (pj_timestamp *t, pj_uint32_t hi, pj_uint32_t lo)`
- `int pj_cmp_timestamp (const pj_timestamp *t1, const pj_timestamp *t2)`
- `void pj_add_timestamp (pj_timestamp *t1, const pj_timestamp *t2)`
- `void pj_add_timestamp32 (pj_timestamp *t1, pj_uint32_t t2)`
- `void pj_sub_timestamp (pj_timestamp *t1, const pj_timestamp *t2)`
- `void pj_sub_timestamp32 (pj_timestamp *t1, pj_uint32_t t2)`
- `pj_int32_t pj_timestamp_diff32 (const pj_timestamp *t1, const pj_timestamp *t2)`
- `pj_time_val pj_elapsed_time (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_msec (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_usec (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_nanosec (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_cycle (const pj_timestamp *start, const pj_timestamp *stop)`

6.31.3 Function Documentation

6.31.3.1 void pj_add_timestamp (pj_timestamp *t1, const pj_timestamp *t2)

Add timestamp t2 to t1.

Parameters:

- `t1` t1.
- `t2` t2.
6.31 High Resolution Timestamp

6.31.3.2 void pj_add_timestamp32 (pj_timestamp ∗ t1, pj_uint32_t t2)
Add timestamp t2 to t1.

Parameters:
  t1 t1.
  t2 t2.

6.31.3.3 int pj_cmp_timestamp (const pj_timestamp ∗ t1, const pj_timestamp ∗ t2)
Compare timestamp t1 and t2.

Parameters:
  t1 t1.
  t2 t2.

Returns:
  -1 if (t1 < t2), 1 if (t1 > t2), or 0 if (t1 == t2)

6.31.3.4 pj_uint32_t pj_elapsed_cycle (const pj_timestamp ∗ start, const pj_timestamp ∗ stop)
Calculate the elapsed time in 32-bit cycles. This function calculates the elapsed time using highest precision calculation that is available for current platform, considering whether floating point or 64-bit precision arithmetic is available. For maximum portability, application should prefer to use this function rather than calculating the elapsed time by itself.

Parameters:
  start The starting timestamp.
  stop The end timestamp.

Returns:
  Elapsed time in cycles.

See also:
  pj_elapsed_usec(), pj_elapsed_time(), pj_elapsed_nanosec()

6.31.3.5 pj_uint32_t pj_elapsed_msec (const pj_timestamp ∗ start, const pj_timestamp ∗ stop)
Calculate the elapsed time as 32-bit milliseconds. This function calculates the elapsed time using highest precision calculation that is available for current platform, considering whether floating point or 64-bit precision arithmetic is available. For maximum portability, application should prefer to use this function rather than calculating the elapsed time by itself.

Parameters:
  start The starting timestamp.
stop The end timestamp.

Returns:
Elapsed time in milisecond.

See also:
pj_elapsed_time(), pj_elapsed_cycle(), pj_elapsed_nanosec()

6.31.3.6 \texttt{pj\_uint32\_t pj\_elapsed\_nanosec (const pj\_timestamp \* start, const pj\_timestamp \* stop)}

Calculate the elapsed time in 32-bit nanoseconds. This function calculates the elapsed time using highest precision calculation that is available for current platform, considering whether floating point or 64-bit precision arithmetic is available. For maximum portability, application should prefer to use this function rather than calculating the elapsed time by itself.

Parameters:
\begin{itemize}
\item \textit{start} The starting timestamp.
\item \textit{stop} The end timestamp.
\end{itemize}

Returns:
Elapsed time in nanoseconds.

See also:
pj_elapsed_time(), pj_elapsed_cycle(), pj_elapsed_usec()

6.31.3.7 \texttt{pj\_time\_val pj\_elapsed\_time (const pj\_timestamp \* start, const pj\_timestamp \* stop)}

Calculate the elapsed time, and store it in \texttt{pj\_time\_val}. This function calculates the elapsed time using highest precision calculation that is available for current platform, considering whether floating point or 64-bit precision arithmetic is available. For maximum portability, application should prefer to use this function rather than calculating the elapsed time by itself.

Parameters:
\begin{itemize}
\item \textit{start} The starting timestamp.
\item \textit{stop} The end timestamp.
\end{itemize}

Returns:
Elapsed time as \texttt{pj\_time\_val}.

See also:
pj_elapsed_usec(), pj_elapsed_cycle(), pj_elapsed_nanosec()}
6.31 High Resolution Timestamp

6.31.3.8 `pj_uint32_t pj_elapsed_usec (const pj_timestamp *start, const pj_timestamp *stop)`

Calculate the elapsed time in 32-bit microseconds. This function calculates the elapsed time using highest precision calculation that is available for current platform, considering whether floating point or 64-bit precision arithmetic is available. For maximum portability, application should prefer to use this function rather than calculating the elapsed time by itself.

**Parameters:**

- `start` The starting timestamp.
- `stop` The end timestamp.

**Returns:**

Elapsed time in microsecond.

**See also:**

`pj_elapsed_time()`, `pj_elapsed_cycle()`, `pj_elapsed_nanosec()`

6.31.3.9 `pj_status_t pj_get_timestamp (pj_timestamp *ts)`

Acquire high resolution timer value. The time value are stored in cycles.

**Parameters:**

- `ts` High resolution timer value.

**Returns:**

PJ_SUCCESS or the appropriate error code.

**See also:**

`pj_get_timestamp_freq()`

6.31.3.10 `pj_status_t pj_get_timestamp_freq (pj_timestamp *freq)`

Get high resolution timer frequency, in cycles per second.

**Parameters:**

- `freq` Timer frequency, in cycles per second.

**Returns:**

PJ_SUCCESS or the appropriate error code.
6.31.3.11  void pj_set_timestamp32 (pj_timestamp * t, pj_uint32_t hi, pj_uint32_t lo)
Set timestamp from 32bit values.

Parameters:
   t  The timestamp to be set.
   hi The high 32bit part.
   lo The low 32bit part.

6.31.3.12  void pj_sub_timestamp (pj_timestamp * t1, const pj_timestamp * t2)
Subtract timestamp t2 from t1.

Parameters:
   t1  t1.
   t2  t2.

6.31.3.13  void pj_sub_timestamp32 (pj_timestamp * t1, pj_uint32_t t2)
Subtract timestamp t2 from t1.

Parameters:
   t1  t1.
   t2  t2.

6.31.3.14  pj_int32_t pj_timestamp_diff32 (const pj_timestamp * t1, const pj_timestamp * t2)
Get the timestamp difference between t2 and t1 (that is t2 minus t1), and return a 32bit signed integer difference.
6.32 Fast Memory Pool

6.32.1 Detailed Description

Memory pools allow dynamic memory allocation comparable to malloc or the new in operator C++. Those implementations are not desirable for very high performance applications or real-time systems, because of the performance bottlenecks and it suffers from fragmentation issue.

6.32.2 PJLIB’s Memory Pool

6.32.2.1 Advantages

PJLIB’s pool has many advantages over traditional malloc/new operator and over other memory pool implementations, because:

- unlike other memory pool implementation, it allows allocation of memory chunks of different sizes,
- it’s very very fast.
  Memory chunk allocation is not only an O(1) operation, but it’s also very simple (just few pointer arithmetic operations) and it doesn’t require locking any mutex,
- it’s memory efficient.
  Pool doesn’t keep track individual memory chunks allocated by applications, so there is no additional overhead needed for each memory allocation (other than possible additional of few bytes, up to PJ_POOL_ALIGNMENT-1, for aligning the memory). But see the Caveats below.
- it prevents memory leaks.
  Memory pool inherently has garbage collection functionality. In fact, there is no need to free the chunks allocated from the memory pool. All chunks previously allocated from the pool will be freed once the pool itself is destroyed. This would prevent memory leaks that haunt programmers for decades, and it provides additional performance advantage over traditional malloc/new operator.

Even more, PJLIB’s memory pool provides some additional usability and flexibility for applications:

- memory leaks are easily traceable, since memory pool is assigned name, and application can inspect what pools currently active in the system.
- by design, memory allocation from a pool is not thread safe. We assumed that a pool will be owned by a higher level object, and thread safety should be handled by that object. This enables very fast pool operations and prevents unnecessary locking operations,
- by default, the memory pool API behaves more like C++ new operator, in that it will throw PJ_NO_MEMORY_EXCEPTION exception (see Exception Handling) when memory chunk allocation fails. This enables failure handling to be done on more high level function (instead of checking the result of pj_pool_alloc() everytime). If application doesn’t like this, the default behavior can be changed on global basis by supplying different policy to the pool factory.
- any memory allocation backend allocator/deallocator may be used. By default, the policy uses malloc() and free() to manage the pool’s block, but application may use different strategy, for example to allocate memory blocks from a globally static memory location.
6.32.2.2 Performance

The result of PJLIB’s memory design and careful implementation is a memory allocation strategy that can speed-up the memory allocations and deallocations by up to **30 times** compared to standard malloc()/free() (more than 150 million allocations per second on a P4/3.0GHz Linux machine).

(Note: your mileage may vary, of course. You can see how much PJLIB’s pool improves the performance over malloc()/free() in your target system by running pjlib-test application).

6.32.2.3 Caveats

There are some caveats though!

When creating pool, PJLIB requires applications to specify the initial pool size, and as soon as the pool is created, PJLIB allocates memory from the system by that size. Application designers MUST choose the initial pool size carefully, since choosing too big value will result in wasting system’s memory.

But the pool can grow. Application designer can specify how the pool will grow in size, by specifying the size increment when creating the pool.

The pool, however, **cannot** shrink! Since there is no function to deallocate memory chunks, there is no way for the pool to release back unused memory to the system. Application designers must be aware that constant memory allocations from pool that has infinite life-time may cause the memory usage of the application to grow over time.

6.32.3 Using Memory Pool

This section describes how to use PJLIB’s memory pool framework. As we hope the readers will witness, PJLIB’s memory pool API is quite straightforward.

6.32.3.1 Create Pool Factory

First, application needs to initialize a pool factory (this normally only needs to be done once in one application). PJLIB provides a pool factory implementation called caching pool (see Caching Pool Factory), and it is initialized by calling `pj_caching_pool_init()`.

6.32.3.2 Create The Pool

Then application creates the pool object itself with `pj_pool_create()`, specifying among other thing the pool factory where the pool should be created from, the pool name, initial size, and increment/expansion size.

6.32.3.3 Allocate Memory as Required

Then whenever application needs to allocate dynamic memory, it would call `pj_pool_alloc()`, `pj_pool_calloc()`, or `pj_pool_zalloc()` to allocate memory chunks from the pool.

6.32.3.4 Destroy the Pool

When application has finished with the pool, it should call `pj_pool_release()` to release the pool object back to the factory. Depending on the types of the factory, this may release the memory back to the operating system.
6.32 Fast Memory Pool

6.32.3.5 Destroy the Pool Factory

And finally, before application quits, it should deinitialize the pool factory, to make sure that all memory blocks allocated by the factory are released back to the operating system. After this, of course no more memory pool allocation can be requested.

6.32.3.6 Example

Below is a sample complete program that utilizes PJLIB’s memory pool.

```c
#include <pjlib.h>
#define THIS_FILE "pool_sample.c"

static void my_perror(const char *title, pj_status_t status)
{
    char errmsg[PJ_ERR_MSG_SIZE];
    pj_strerror(status, errmsg, sizeof(errmsg));
    PJ_LOG(1,(THIS_FILE, "%s: %s [status=%d]", title, errmsg, status));
}

static void pool_demo_1(pj_pool_factory *pfactory)
{
    unsigned i;
    pj_pool_t *pool;

    // Must create pool before we can allocate anything
    pool = pj_pool_create(pfactory, "pool1", 4000, 4000, NULL);
    if (pool == NULL) {
        my_perror("Error creating pool", PJ_ENOMEM);
        return;
    }

    // Demo: allocate some memory chunks
    for (i=0; i<1000; ++i) {
        void *p;
        p = pj_pool_alloc(pool, (pj_rand()+1) % 512);
        // Do something with p
        ...
        // Look! No need to free p!!
    }

    // Done with silly demo, must free pool to release all memory.
    pj_pool_release(pool);
}

int main()
{
    pj_caching_pool cp;
    pj_status_t status;

    // Must init PJLIB before anything else
    status = pj_init();
    if (status != PJ_SUCCESS) {
        my_perror("Error initializing PJLIB", status);
        return 1;
    }
}
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
```c
// Create the pool factory, in this case, a caching pool,
// using default pool policy.
pj_caching_pool_init(&cp, NULL, 1024*1024);

// Do a demo
pool_demo_1(&cp.factory);

// Done with demos, destroy caching pool before exiting app.
pj_caching_pool_destroy(&cp);

return 0;
```

More information about pool factory, the pool object, and caching pool can be found on the Module Links below.

**Modules**

- **Memory Pool Object**
  
  The memory pool is an opaque object created by pool factory. Application uses this object to request a memory chunk, by calling `pj_pool_alloc()`, `pj_pool_calloc()`, or `pj_pool_zalloc()`. When the application has finished using the pool, it must call `pj_pool_release()` to free all the chunks previously allocated and release the pool back to the factory.

- **Pool Factory and Policy**

  A pool object must be created through a factory. A factory not only provides generic interface functions to create and release pool, but also provides strategy to manage the life time of pools. One sample implementation, `pj_caching_pool`, can be set to keep the pools released by application for future use as long as the total memory is below the limit.

- **Caching Pool Factory**

  Caching pool is one sample implementation of pool factory where the factory can reuse memory to create a pool. Application defines what the maximum memory the factory can hold, and when a pool is released the factory decides whether to destroy the pool or to keep it for future use. If the total amount of memory in the internal cache is still within the limit, the factory will keep the pool in the internal cache, otherwise the pool will be destroyed, thus releasing the memory back to the system.

- **Stack/Buffer Based Memory Pool Allocator**

  Stack/buffer based pool.
6.33 Memory Pool Object

6.33.1 Detailed Description

The memory pool is an opaque object created by pool factory. Application uses this object to request a memory chunk, by calling `pj_pool_alloc()`, `pj_pool_calloc()`, or `pj_pool_zalloc()`. When the application has finished using the pool, it must call `pj_pool_release()` to free all the chunks previously allocated and release the pool back to the factory.

A memory pool is initialized with an initial amount of memory, which is called a block. Pool can be configured to dynamically allocate more memory blocks when it runs out of memory.

The pool doesn’t keep track of individual memory allocations by user, and the user doesn’t have to free these individual allocations. This makes memory allocation simple and very fast. All the memory allocated from the pool will be destroyed when the pool itself is destroyed.

6.33.2 More on Threading Policies

- By design, memory allocation from a pool is not thread safe. We assumed that a pool will be owned by an object, and thread safety should be handled by that object. Thus these functions are not thread safe:
  - `pj_pool_alloc`,
  - `pj_pool_calloc`,
  - and other pool statistic functions.
- Threading in the pool factory is decided by the policy set for the factory when it was created.

6.33.3 Examples

For some sample codes on how to use the pool, please see:

- Test: Pool

Data Structures

- `struct pj_pool_block`
- `struct pj_pool_t`

Defines

- `#define PJ_POOL_SIZE (sizeof(struct pj_pool_t))`
- `#define PJ_POOL_ALIGNMENT 4`
- `#define PJ_POOL_ALLOC_T(pool, type) ((type*)pj_pool_alloc(pool, sizeof(type)))`
- `#define PJ_POOL_ZALLOC_T(pool, type) ((type*)pj_pool_zalloc(pool, sizeof(type)))`

Typedefs

- `typedef void pj_pool_callback (pj_pool_t *pool, pj_size_t size)`
Functions

- `pj_pool_t * pj_pool_create (pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)`
- void `pj_pool_release (pj_pool_t *pool)`
- const char * `pj_pool_getobjname (const pj_pool_t *pool)`
- void `pj_pool_reset (pj_pool_t *pool)`
- `pj_size_t pj_pool_get_capacity (pj_pool_t *pool)`
- `pj_size_t pj_pool_get_used_size (pj_pool_t *pool)`
- void * `pj_pool_alloc (pj_pool_t *pool, pj_size_t size)`
- void * `pj_pool_calloc (pj_pool_t *pool, pj_size_t count, pj_size_t elem)`
- void * `pj_pool_zalloc (pj_pool_t *pool, pj_size_t size)`
- void * `pj_pool_alloc_from_block (pj_pool_block *block, pj_size_t size)`
- void * `pj_pool_allocate_find (pj_pool_t *pool, unsigned size)`

### 6.33.4 Define Documentation

#### 6.33.4.1 `#define PJ_POOL_ALIGNMENT 4`

Pool memory alignment (must be power of 2).

#### 6.33.4.2 `#define PJ_POOL_ALLOC_T(pool, type) ((type *)pj_pool_alloc(pool, sizeof(type)))`

This macro allocates memory from the pool and returns the instance of the specified type. It provides a stricter type safety than `pj_pool_alloc()` since the return value of this macro will be type-casted to the specified type.

**Parameters:**
- `pool` The pool
- `type` The type of object to be allocated

**Returns:**
Memory buffer of the specified type.

#### 6.33.4.3 `#define PJ_POOL_SIZE (sizeof(struct pj_pool_t))`

Guidance on how much memory required for initial pool administrative data.

#### 6.33.4.4 `#define PJ_POOL_ZALLOC_T(pool, type) ((type *)pj_pool_zalloc(pool, sizeof(type)))`

This macro allocates memory from the pool, zeroes the buffer, and returns the instance of the specified type. It provides a stricter type safety than `pj_pool_zalloc()` since the return value of this macro will be type-casted to the specified type.

**Parameters:**
- `pool` The pool
- `type` The type of object to be allocated
Returns:
Memory buffer of the specified type.

6.33.5 Typedef Documentation

6.33.5.1 typedef void pj_pool_callback(pj_pool_t *pool, pj_size_t size)

The type for function to receive callback from the pool when it is unable to allocate memory. The elegant way to handle this condition is to throw exception, and this is what is expected by most of this library components.

6.33.6 Function Documentation

6.33.6.1 void* pj_pool_alloc (pj_pool_t * pool, pj_size_t size)

Allocate storage with the specified size from the pool. If there’s no storage available in the pool, then the pool can allocate more blocks if the increment size is larger than the requested size.

Parameters:

  pool  the pool.
  size  the requested size.

Returns:
pointer to the allocated memory.

See also:
PJ_POOL_ALLOC_T

6.33.6.2 void* pj_pool_calloc (pj_pool_t * pool, pj_size_t count, pj_size_t elem)

Allocate storage from the pool, and initialize it to zero. This function behaves like pj_pool_alloc(), except that the storage will be initialized to zero.

Parameters:

  pool  the pool.
  count the number of elements in the array.
  elem  the size of individual element.

Returns:
pointer to the allocated memory.

6.33.6.3 pj_pool_t* pj_pool_create (pj_pool_factory * factory, const char * name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback * callback)

Create a new pool from the pool factory. This wrapper will call create_pool member of the pool factory.
Parameters:

factory  The pool factory.
name  The name to be assigned to the pool. The name should not be longer than PJ_MAX_OBJ_NAME (32 chars), or otherwise it will be truncated.
initial_size  The size of initial memory blocks taken by the pool. Note that the pool will take 68+20 bytes for administrative area from this block.
increment_size  the size of each additional blocks to be allocated when the pool is running out of memory. If user requests memory which is larger than this size, then an error occurs. Note that each time a pool allocates additional block, it needs PJ_POOL_SIZE more to store some administrative info.
callback  Callback to be called when error occurs in the pool. If this value is NULL, then the callback from pool factory policy will be used. Note that when an error occurs during pool creation, the callback itself is not called. Instead, NULL will be returned.

Returns:

The memory pool, or NULL.

6.33.6.4  pj_size_t pj_pool_get_capacity (pj_pool_t * pool)

Get the pool capacity, that is, the system storage that have been allocated by the pool, and have been used/will be used to allocate user requests. There’s no guarantee that the returned value represent a single contiguous block, because the capacity may be spread in several blocks.

Parameters:

pool  the pool.

Returns:

the capacity.

6.33.6.5  pj_size_t pj_pool_get_used_size (pj_pool_t * pool)

Get the total size of user allocation request.

Parameters:

pool  the pool.

Returns:

the total size.

6.33.6.6  const char* pj_pool_getobjname (const pj_pool_t * pool)

Get pool object name.

Parameters:

pool  the pool.
6.33 Memory Pool Object

Returns:

pool name as NULL terminated string.

6.33.6.7 void pj_pool_release (pj_pool_t * pool)

Release the pool back to pool factory.

Parameters:

pool Memory pool.

6.33.6.8 void pj_pool_reset (pj_pool_t * pool)

Reset the pool to its state when it was initialized. This means that if additional blocks have been allocated during runtime, then they will be freed. Only the original block allocated during initialization is retained. This function will also reset the internal counters, such as pool capacity and used size.

Parameters:

pool the pool.

6.33.6.9 void * pj_pool_zalloc (pj_pool_t * pool, pj_size_t size)

Allocate storage from the pool and initialize it to zero.

Parameters:

pool The pool.

size The size to be allocated.

Returns:

Pointer to the allocated memory.

See also:

PJ_POOL_ZALLOC_T
6.34  Pool Factory and Policy

6.34.1  Detailed Description

A pool object must be created through a factory. A factory not only provides generic interface functions to create and release pool, but also provides strategy to manage the life time of pools. One sample implementation, \texttt{pj_caching_pool}, can be set to keep the pools released by application for future use as long as the total memory is below the limit.

The pool factory interface declared in PJLIB is designed to be extensible. Application can define its own strategy by creating its own pool factory implementation, and this strategy can be used even by existing library without recompilation.

6.34.2  Pool Factory Interface

The pool factory defines the following interface:

- \texttt{policy}: the memory pool factory policy.
- \texttt{create_pool}(): create a new memory pool.
- \texttt{release_pool}(): release memory pool back to factory.

6.34.3  Pool Factory Policy.

A pool factory only defines functions to create and release pool and how to manage pools, but the rest of the functionalities are controlled by policy. A pool policy defines:

- how memory block is allocated and deallocated (the default implementation allocates and deallocate memory by calling malloc() and free()).
- callback to be called when memory allocation inside a pool fails (the default implementation will throw \texttt{PJ\_NO\_MEMORY\_EXCEPTION} exception).
- concurrency when creating and releasing pool from/to the factory.

A pool factory can be given different policy during creation to make it behave differently. For example, caching pool factory can be configured to allocate and deallocate from a static/contiguous/preallocated memory instead of using malloc()/free().

What strategy/factory and what policy to use is not defined by PJLIB, but instead is left to application to make use whichever is most efficient for itself.

The pool factory policy controls the behaviour of memory factories, and defines the following interface:

- \texttt{block_alloc}(): allocate memory block from backend memory mgmt/system.
- \texttt{block_free}(): free memory block back to backend memory mgmt/system.

Data Structures

- \texttt{struct pj_pool_factory_policy}
- \texttt{struct pj_pool_factory}
Functions

- PJ_DECL_DATA (int) PJ_NO_MEMORY_EXCEPTION
- int pj_NO_MEMORY_EXCEPTION ()
- PJ_DECL_DATA (pj_pool_factory_policy) pj_pool_factory_default_policy
- const pj_pool_factory_policy * pj_pool_factory_get_default_policy (void)
- pj_pool_t * pj_pool_create_int (pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)
- void pj_pool_init_int (pj_pool_t *pool, const char *name, pj_size_t increment_size, pj_pool_callback *callback)
- void pj_pool_destroy_int (pj_pool_t *pool)
- void pj_pool_factory_dump (pj_pool_factory *pf, pj_bool_t detail)

6.34.4 Function Documentation

6.34.4.1 PJ_DECL_DATA (pj_pool_factory_policy)
This global variable points to default memory pool factory policy. The behaviour of the default policy is:

- block allocation and deallocation use malloc() and free().
- callback will raise PJ_NO_MEMORY_EXCEPTION exception.
- access to pool factory is not serialized (i.e. not thread safe).

See also:

pj_pool_factory_get_default_policy

6.34.4.2 PJ_DECL_DATA (int)
This constant denotes the exception number that will be thrown by default memory factory policy when memory allocation fails.

See also:

pj_NO_MEMORY_EXCEPTION()

6.34.4.3 int pj_NO_MEMORY_EXCEPTION ()
Get PJ_NO_MEMORY_EXCEPTION constant.

6.34.4.4 pj_pool_t * pj_pool_create_int (pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)
This function is intended to be used by pool factory implementors.

Parameters:

factory Pool factory.
name Pool name.
initial_size Initial size.
increment_size Increment size.
callback Callback.

Returns:
The pool object, or NULL.

6.34.4.5 void pj_pool_destroy_int (pj_pool_t * pool)
This function is intended to be used by pool factory implementors.

Parameters:
pool The memory pool.

6.34.4.6 void pj_pool_factory_dump (pj_pool_factory * pf, pj_bool_t detail)
Dump pool factory state.

Parameters:
pf The pool factory.
detail Detail state required.

6.34.4.7 const pj_pool_factory_policy* pj_pool_factory_get_default_policy (void)
Get the default pool factory policy.

Returns:
the pool policy.

6.34.4.8 void pj_pool_init_int (pj_pool_t * pool, const char * name, pj_size_t increment_size,
   pj_pool_callback * callback)
This function is intended to be used by pool factory implementors.

Parameters:
pool The pool.
name Pool name.
increment_size Increment size.
callback Callback function.
6.35 Caching Pool Factory

6.35.1 Detailed Description

Caching pool is one sample implementation of pool factory where the factory can reuse memory to create a pool. Application defines what the maximum memory the factory can hold, and when a pool is released the factory decides whether to destroy the pool or to keep it for future use. If the total amount of memory in the internal cache is still within the limit, the factory will keep the pool in the internal cache, otherwise the pool will be destroyed, thus releasing the memory back to the system.

Data Structures

- struct pj_caching_pool

Defines

- #define PJ_CACHING_POOL_ARRAY_SIZE 16

Functions

- void pj_caching_pool_init (pj_caching_pool *ch_pool, const pj_pool_factory_policy *policy, pj_size_t max_capacity)
- void pj_caching_pool_destroy (pj_caching_pool *ch_pool)

6.35.2 Define Documentation

6.35.2.1 #define PJ_CACHING_POOL_ARRAY_SIZE 16

Number of unique sizes, to be used as index to the free list. Each pool in the free list is organized by it’s size.

6.35.3 Function Documentation

6.35.3.1 void pj_caching_pool_destroy (pj_caching_pool *ch_pool)

Destroy caching pool, and release all the pools in the recycling list.

Parameters:

- ch_pool The caching pool.

6.35.3.2 void pj_caching_pool_init (pj_caching_pool *ch_pool, const pj_pool_factory_policy *policy, pj_size_t max_capacity)

Initialize caching pool.

Parameters:

- ch_pool The caching pool factory to be initialized.
**policy**  Pool factory policy.

**max_capacity**  The total capacity to be retained in the cache. When the pool is returned to the cache, it will be kept in recycling list if the total capacity of pools in this list plus the capacity of the pool is still below this value.
6.36 Stack/Buffer Based Memory Pool Allocator

6.36.1 Detailed Description

Stack/buffer based pool.

This section describes an implementation of memory pool which uses memory allocated from the stack. Application creates this pool by specifying a buffer (which can be allocated from static memory or stack variable), and then use normal pool API to access/use the pool.

If the buffer specified during pool creation is a buffer located in the stack, the pool will be invalidated (or implicitly destroyed) when the execution leaves the enclosing block containing the buffer. Note that application must make sure that any objects allocated from this pool (such as mutexes) have been destroyed before the pool gets invalidated.

Sample usage:

```c
#include <pjlib.h>
static void test()
{
    char buffer[500];
    pj_pool_t *pool;
    void *p;
    pool = pj_pool_create_on_buf("thepool", buffer, sizeof(buffer));
    // Use the pool as usual
    p = pj_pool_alloc(pool, ...);
    ...
    // No need to release the pool
}
int main()
{
    pj_init();
    test();
    return 0;
}
```

Functions

- PJ_BEGIN_DECL pj_pool_t *pj_pool_create_on_buf (const char *name, void *buf, pj_size_t size)

6.36.2 Function Documentation

6.36.2.1 PJ_BEGIN_DECL pj_pool_t* pj_pool_create_on_buf (const char * name, void *buf,
pj_size_t size)

Create the pool using the specified buffer as the pool’s memory. Subsequent allocations made from the pool will use the memory from this buffer.

If the buffer specified in the parameter is a buffer located in the stack, the pool will be invalid (or implicitly destroyed) when the execution leaves the enclosing block containing the buffer. Note that application must make sure that any objects allocated from this pool (such as mutexes) have been destroyed before the pool gets invalidated.
Parameters:

- **name**: Optional pool name.
- **buf**: Buffer to be used by the pool.
- **size**: The size of the buffer.

Returns:

The memory pool instance.
6.37 Random Number Generator

6.37.1 Detailed Description

This module contains functions for generating random numbers. This abstraction is needed not only because not all platforms have `rand()` and `srand()`, but also on some platforms `rand()` only has 16-bit randomness, which is not good enough.

Functions

- void `pj_srand` (unsigned int `seed`)
- int `pj_rand` (void)

6.37.2 Function Documentation

6.37.2.1 int `pj_rand` (void)

Generate random integer with 32bit randomness.

Returns:

a random integer.

6.37.2.2 void `pj_srand` (unsigned int `seed`)

Put in seed to random number generator.

Parameters:

`seed` Seed value.
6.38 Red/Black Balanced Tree

6.38.1 Detailed Description

Red/Black tree is the variant of balanced tree, where the search, insert, and delete operation is guaranteed to take at most $O(\lg(n))$.

Data Structures

- struct pj_rbtree_node
- struct pj_rbtree

Defines

- #define PJ_RBTREE_NODE_SIZE (sizeof(pj_rbtree_node))
- #define PJ_RBTREE_SIZE (sizeof(pj_rbtree))

Typedefs

- typedef int pj_rbtree_comp (const void *key1, const void *key2)

Enumerations

- enum pj_rbcolor_t { PJ_RBCOLOR_BLACK, PJ_RBCOLOR_RED } 

Functions

- void pj_rbtree_init (pj_rbtree *tree, pj_rbtree_comp *comp)
- pj_rbtree_node * pj_rbtree_first (pj_rbtree *tree)
- pj_rbtree_node * pj_rbtree_last (pj_rbtree *tree)
- pj_rbtree_node * pj_rbtree_next (pj_rbtree *tree, pj_rbtree_node *node)
- pj_rbtree_node * pj_rbtree_prev (pj_rbtree *tree, pj_rbtree_node *node)
- int pj_rbtree_insert (pj_rbtree *tree, pj_rbtree_node *node)
- pj_rbtree_node * pj_rbtree_find (pj_rbtree *tree, const void *key)
- pj_rbtree_node * pj_rbtree_erase (pj_rbtree *tree, pj_rbtree_node *node)
- unsigned pj_rbtree_max_height (pj_rbtree *tree, pj_rbtree_node *node)
- unsigned pj_rbtree_min_height (pj_rbtree *tree, pj_rbtree_node *node)

6.38.2 Define Documentation

6.38.2.1 #define PJ_RBTREE_NODE_SIZE (sizeof(pj_rbtree_node))

Guidance on how much memory required for each of the node.

6.38.2.2 #define PJ_RBTREE_SIZE (sizeof(pj_rbtree))

Guidance on memory required for the tree.
6.38 Red/Black Balanced Tree

6.38.3 Typedef Documentation

6.38.3.1 typedef int pj_rbtree_comp(const void *key1, const void *key2)

The type of function used to compare key value of tree node.

Returns:
  0 if the keys are equal <0 if key1 is lower than key2 >0 if key1 is greater than key2.

6.38.4 Enumeration Type Documentation

6.38.4.1 enum pj_rbcolor_t

Color type for Red-Black tree.

6.38.5 Function Documentation

6.38.5.1 pj_rbtree_node* pj_rbtree_erase (pj_rbtree *tree, pj_rbtree_node *node)

Erase a node from the tree.

Parameters:
  tree the tree.
  node the node to be erased.

Returns:
  the tree node itself.

6.38.5.2 pj_rbtree_node* pj_rbtree_find (pj_rbtree *tree, const void *key)

Find a node which has the specified key.

Parameters:
  tree the tree.
  key the key to search.

Returns:
  the tree node with the specified key, or NULL if the key can not be found.

6.38.5.3 pj_rbtree_node* pj_rbtree_first (pj_rbtree *tree)

Get the first element in the tree. The first element always has the least value for the key, according to the comparison function.

Parameters:
  tree the tree.
Returns:
the tree node, or NULL if the tree has no element.

6.38.5.4 void pj_rbtree_init (pj_rbtree * tree, pj_rbtree_comp * comp)
Initialize the tree.
Parameters:
\begin{itemize}
  \item tree the tree to be initialized.
  \item comp key comparison function to be used for this tree.
\end{itemize}

6.38.5.5 int pj_rbtree_insert (pj_rbtree * tree, pj_rbtree_node * node)
Insert a new node. The node will be inserted at sorted location. The key of the node must be UNIQUE, i.e. it hasn’t existed in the tree.
Parameters:
\begin{itemize}
  \item tree the tree.
  \item node the node to be inserted.
\end{itemize}
Returns:
zero on success, or -1 if the key already exist.

6.38.5.6 pj_rbtree_node* pj_rbtree_last (pj_rbtree * tree)
Get the last element in the tree. The last element always has the greatest key value, according to the comparison function defined for the tree.
Parameters:
\begin{itemize}
  \item tree the tree.
\end{itemize}
Returns:
the tree node, or NULL if the tree has no element.

6.38.5.7 unsigned pj_rbtree_max_height (pj_rbtree * tree, pj_rbtree_node * node)
Get the maximum tree height from the specified node.
Parameters:
\begin{itemize}
  \item tree the tree.
  \item node the node, or NULL to get the root of the tree.
\end{itemize}
Returns:
the maximum height, which should be at most \(\log_2(N)\)
6.38 Red/Black Balanced Tree

6.38.5.8  unsigned pj_rbtree_min_height (pj_rbtree ∗ tree, pj_rbtree_node ∗ node)

Get the minimum tree height from the specified node.

Parameters:

  tree  the tree.
  node  the node, or NULL to get the root of the tree.

Returns:

  the height

6.38.5.9  pj_rbtree_node ∗ pj_rbtree_next (pj_rbtree ∗ tree, pj_rbtree_node ∗ node)

Get the successive element for the specified node. The successive element is an element with greater key value.

Parameters:

  tree  the tree.
  node  the node.

Returns:

  the successive node, or NULL if the node has no successor.

6.38.5.10  pj_rbtree_node ∗ pj_rbtree_prev (pj_rbtree ∗ tree, pj_rbtree_node ∗ node)

Get the previous node for the specified node. The previous node is an element with less key value.

Parameters:

  tree  the tree.
  node  the node.

Returns:

  the previous node, or NULL if the node has no previous node.
6.39 Socket Abstraction

6.39.1 Detailed Description

The PJLIB socket abstraction layer is a thin and very portable abstraction for socket API. It provides API similar to BSD socket API. The abstraction is needed because BSD socket API is not always available on all platforms, therefore it wouldn’t be possible to create a trully portable network programs unless we provide such abstraction.

Applications can use this API directly in their application, just as they would when using traditional BSD socket API, provided they call `pj_init()` first.

6.39.2 Examples

For some examples on how to use the socket API, please see:

- Test: Socket
- Test: Socket Select()
- Test: Socket Performance

Data Structures

- struct `pj_in_addr`
- struct `pj_sockaddr_in`
- struct `pj_in6_addr`
- struct `pj_sockaddr_in6`
- struct `pj_addr_hdr`
- union `pj_sockaddr`

Defines

- #define `PJ_AF_LOCAL` `PJ_AF_UNIX`
- #define `PJ_INADDR_ANY` `((pj_uint32_t)0)`
- #define `PJ_INADDR_NONE` `((pj_uint32_t)0xffffffff)`
- #define `PJ_INADDR_BROADCAST` `((pj_uint32_t)0xffffffff)`
- #define `PJ_SOMAXCONN` 5
- #define `PJ_INVALID_SOCKET` (-1)
- #define `PJ_IN6ADDR_ANY_INIT` `{{ { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 } } }
- #define `PJ_IN6ADDR_LOOPBACK_INIT` `{{ { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 } } }

Enumerations

- enum `pj_socket_sd_type` {
  
  `PJ_SD_RECEIVE` = 0, `PJ_SHUT_RD` = 0, `PJ_SD_SEND` = 1, `PJ_SHUT_WR` = 1,
  `PJ_SD_BOTH` = 2, `PJ_SHUT_RDWR` = 2
  
}
Functions

- `pj_uint16_t pj_AF_UNIX (void)`
- `pj_uint16_t pj_AF_INET (void)`
- `pj_uint16_t pj_AF_INET6 (void)`
- `pj_uint16_t pj_AF_PACKET (void)`
- `pj_uint16_t pj_AF_IRDA (void)`
- `int pj SOCK_STREAM (void)`
- `int pj SOCK_DGRAM (void)`
- `int pj SOCK_RAW (void)`
- `int pj SOCK_RDM (void)`
- `pj_uint16_t pj_SOL_SOCKET (void)`
- `pj_uint16_t pj_SOL_IP (void)`
- `pj_uint16_t pj_SOL_TCP (void)`
- `pj_uint16_t pj_SOL_UDP (void)`
- `pj_uint16_t pj_SOL_IPV6 (void)`
- `int pj_IP_TOS (void)`
- `int pj IPTOS_LOWDELAY (void)`
- `int pj IPTOS_THROUGHPUT (void)`
- `int pj IPTOS_RELIABILITY (void)`
- `int pj IPTOS_MINCOST (void)`
- `pj_uint16_t pj_SO_TYPE (void)`
- `pj_uint16_t pj_SO_RCVBUF (void)`
- `pj_uint16_t pj_SO_SNDBUF (void)`
- `int pj MSG_OOB (void)`
- `int pj MSG_PEEK (void)`
- `int pj MSG_DONTROUTE (void)`
- `pj_uint16_t pj_sockaddr_in_get_port (const pj_sockaddr_in *addr)`
- `void pj_sockaddr_in_set_port (pj_sockaddr_in *addr, pj_uint16_t hostport)`
- `pj_in_addr pj_sockaddr_in_get_addr (const pj_sockaddr_in *addr)`
- `void pj_sockaddr_in_set_addr (pj_sockaddr_in *addr, pj_uint32_t hostaddr)`
- `const pj_str_t *pj_gethostname (void)`
- `int pj_sock_socket (int family, int type, int protocol, pj_sock_t *sock)`
- `int pj_sock_close (pj_sock_t sockfd)`
- `pj_status_t pj_sock_bind (pj_sock_t sockfd, const pj_sockaddr_t *my_addr, int addrlen)`
- `pj_status_t pj_sock_bind_in (pj_sock_t sockfd, pj_uint32_t addr, pj_uint16_t port)`
- `pj_status_t pj_sock_listen (pj_sock_t sockfd, int backlog)`
- `pj_status_t pj_sock_accept (pj_sock_t serverfd, pj_sock_t *newsock, pj_sockaddr_t *addr, int *addrlen)`
• `pj_status_t pj_sock_connect (pj_sock_t sockfd, const pj_sockaddr_t *serv_addr, int addrlen)`
• `pj_status_t pj_sock_getpeername (pj_sock_t sockfd, pj_sockaddr_t *addr, int *namelen)`
• `pj_status_t pj_sock_getsockname (pj_sock_t sockfd, pj_sockaddr_t *addr, int *namelen)`
• `pj_status_t pj_sock_getsockopt (pj_sock_t sockfd, pj_uint16_t level, pj_uint16_t optname, void *optval, int *optlen)`
• `pj_status_t pj_sock_setsockopt (pj_sock_t sockfd, pj_uint16_t level, pj_uint16_t optname, const void *optval, int optlen)`
• `pj_status_t pj_sock_recv (pj_sock_t sockfd, void *buf, pj_ssize_t *len, unsigned flags)`
• `pj_status_t pj_sock_recvfrom (pj_sock_t sockfd, void *buf, pj_ssize_t *len, unsigned flags, pj_sockaddr_t *from, int *fromlen)`
• `pj_status_t pj_sock_send (pj_sock_t sockfd, const void *buf, pj_ssize_t *len, unsigned flags)`
• `pj_status_t pj_sock_sendto (pj_sock_t sockfd, const void *buf, pj_ssize_t *len, unsigned flags, const pj_sockaddr_t *to, int tolen)`
• `pj_status_t pj_sock_shutdown (pj_sock_t sockfd, int how)`

Variables

• const `pj_uint16_t PJ_AF_UNIX`
• const `pj_uint16_t PJ_AF_INET`
• const `pj_uint16_t PJ_AF_INET6`
• const `pj_uint16_t PJ_AF_PACKET`
• const `pj_uint16_t PJ_AF_IRDA`
• const `pj_uint16_t PJ_SOCK_STREAM`
• const `pj_uint16_t PJ_SOCK_DGRAM`
• const `pj_uint16_t PJ_SOCK_RAW`
• const `pj_uint16_t PJ_SOCK_RDM`
• const `pj_uint16_t PJ_SOL_SOCKET`
• const `pj_uint16_t PJ_SOL_IP`
• const `pj_uint16_t PJ_SOL_TCP`
• const `pj_uint16_t PJ_SOL_UDP`
• const `pj_uint16_t PJ_SOL_IPV6`
• const `pj_uint16_t PJ_SO_TYPE`
• const `pj_uint16_t PJ_SO_RCVBUF`
• const `pj_uint16_t PJ_SO_SNDBUF`
• const int `PJ_MSG_OOB`
• const int `PJ_MSG_PEEK`
• const int `PJ_MSG_DONTROUTE`

6.39.3 Define Documentation

6.39.3.1 #define PJ_AF_LOCAL PJ_AF_UNIX;

POSIX name for AF_UNIX
6.39 Socket Abstraction

6.39.3.2  
#define PJ_IN6ADDR_ANY_INIT { { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 } } }  
Initializer value for `pj_in6_addr`.

6.39.3.3  
#define PJ_IN6ADDR_LOOPBACK_INIT { { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 } } }  
Initializer value for `pj_in6_addr`.

6.39.3.4  
#define PJ_INADDR_ANY ((pj_uint32_t)0)  
Address to accept any incoming messages.

6.39.3.5  
#define PJ_INADDR.Broadcast ((pj_uint32_t)0xffffffff)  
Address to send to all hosts.

6.39.3.6  
#define PJ_INADDR_NONE ((pj_uint32_t)0xffffffff)  
Address indicating an error return

6.39.3.7  
#define PJ_INVALID_SOCKET (-1)  
Constant for invalid socket returned by `pj_sock_socket()` and `pj_sock_accept()`.

6.39.3.8  
#define PJ_SOMAXCONN 5  
Maximum length specifiable by `pj_sock_listen()`. If the build system doesn’t override this value, then the lowest denominator (five, in Win32 systems) will be used.

6.39.4  Enumeration Type Documentation

6.39.4.1  
enum pj_socket_sd_type  
Flag to be specified in `pj_sock_shutdown()`.

**Enumerator:**

- `PJ_SD_RECEIVE`  No more receive.
- `PJ_SHUT_RD`  Alias for SD_RECEIVE.
- `PJ_SD_SEND`  No more sending.
- `PJ_SHUT_WR`  Alias for SD_SEND.
- `PJ_SD_BOTH`  No more send and receive.
- `PJ_SHUT_RDWR`  Alias for SD_BOTH.
6.39.5 Function Documentation

6.39.5.1 `pj_uint16_t pj_AF_INET (void)`

Get PJ_AF_INET value.

6.39.5.2 `pj_uint16_t pj_AF_INET6 (void)`

Get PJ_AF_INET6 value.

6.39.5.3 `pj_uint16_t pj_AF_IRDA (void)`

Get PJ_AF_IRDA value.

6.39.5.4 `pj_uint16_t pj_AF_PACKET (void)`

Get PJ_AF_PACKET value.

6.39.5.5 `pj_uint16_t pj_AF_UNIX (void)`

Get PJ_AF_UNIX value.

6.39.5.6 `pj_in_addr pj_gethostaddr (void)`

Get host’s IP address, which the first IP address that is resolved from the hostname.

**Returns:**

The host’s IP address, PJ_INADDR_NONE if the host IP address can not be identified.

6.39.5.7 `const pj_str_t* pj_gethostname (void)`

Get system’s host name.

**Returns:**

The hostname, or empty string if the hostname can not be identified.

6.39.5.8 `pj_uint32_t pj_htonl (pj_uint32_t hostlong)`

Convert 32-bit value from host byte order to network byte order.

**Parameters:**

`hostlong` 32-bit host value.

**Returns:**

32-bit network value.
6.39 Socket Abstraction

6.39.5.9  `pj_uint16_t pj_htons (pj_uint16_t hostshort)`

Convert 16-bit value from host byte order to network byte order.

**Parameters:**

- `hostshort` 16-bit host value.

**Returns:**

16-bit network value.

6.39.5.10  `pj_in_addr pj_inet_addr (const pj_str_t * cp)`

Convert address string with numbers and dots to binary IP address.

**Parameters:**

- `cp` The IP address in numbers and dots notation.

**Returns:**

If success, the IP address is returned in network byte order. If failed, PJ_INADDR_NONE will be returned.

**Remarks:**

This is an obsolete interface to `pj_inet_aton()`; it is obsolete because -1 is a valid address (255.255.255.255), and `pj_inet_aton()` provides a cleaner way to indicate error return.

6.39.5.11  `pj_in_addr pj_inet_addr2 (const char * cp)`

Convert address string with numbers and dots to binary IP address.

**Parameters:**

- `cp` The IP address in numbers and dots notation.

**Returns:**

If success, the IP address is returned in network byte order. If failed, PJ_INADDR_NONE will be returned.

**Remarks:**

This is an obsolete interface to `pj_inet_aton()`; it is obsolete because -1 is a valid address (255.255.255.255), and `pj_inet_aton()` provides a cleaner way to indicate error return.

6.39.5.12  `int pj_inet_aton (const pj_str_t * cp, struct pj_in_addr * inp)`

This function converts the Internet host address `cp` from the standard numbers-and-dots notation into binary data and stores it in the structure that `inp` points to.
Parameters:

- **cp** IP address in standard numbers-and-dots notation.
- **inp** Structure that holds the output of the conversion.

Returns:

nonzero if the address is valid, zero if not.

6.39.5.13 char* pj_inet_ntoa (pj_in_addr inaddr)

Convert an Internet host address given in network byte order to string in standard numbers and dots notation.

Parameters:

- **inaddr** The host address.

Returns:

The string address.

6.39.5.14 int pj_IP_TOS (void)

Get PJ_IP_TOS constant

6.39.5.15 int pj_IPTOS_LOWDELAY (void)

Get PJ_IPTOS_LOWDELAY constant

6.39.5.16 int pj_IPTOS_MINCOST (void)

Get PJ_IPTOS_MINCOST constant

6.39.5.17 int pj_IPTOS_RELIABILITY (void)

Get PJ_IPTOS_RELIABILITY constant

6.39.5.18 int pj_IPTOS_THROUGHPUT (void)

Get PJ_IPTOS_THROUGHPUT constant

6.39.5.19 int pj_MSG_DONTROUTE (void)

Get PJ_MSG_DONTROUTE constant

6.39.5.20 int pj_MSG_OOB (void)

Get PJ_MSG_OOB constant
6.39 Socket Abstraction

6.39.5.21 int pj_MSG_PEEK (void)

Get PJ_MSG_PEEK constant

6.39.5.22 pj_uint32_t pj_ntohl (pj_uint32_t netlong)

Convert 32-bit value from network byte order to host byte order.

Parameters:

netlong 32-bit network value.

Returns:

32-bit host value.

6.39.5.23 pj_uint16_t pj_ntohs (pj_uint16_t netshort)

Convert 16-bit value from network byte order to host byte order.

Parameters:

netshort 16-bit network value.

Returns:

16-bit host value.

6.39.5.24 pj_uint16_t pj_SO_RCVBUF (void)

Get PJ_SO_RCVBUF constant

6.39.5.25 pj_uint16_t pj_SO_SNDBUF (void)

Get PJ_SO_SNDBUF constant

6.39.5.26 pj_uint16_t pj_SO_TYPE (void)

Get PJ_SO_TYPE constant

6.39.5.27 pj_status_t pj_sock_accept (pj_sock_t serverfd, pj_sock_t *newsock, pj_sockaddr_t *addr, int *addrlen)

Accept new connection on the specified connection oriented server socket.

Parameters:

serverfd The server socket.

newsock New socket on success, of PJ_INVALID_SOCKET if failed.
addr A pointer to sockaddr type. If the argument is not NULL, it will be filled by the address of connecting entity.

addrlen Initially specifies the length of the address, and upon return will be filled with the exact address length.

Returns:
Zero on success, or the error number.

6.39.5.28  pj_status_t pj_sock_bind (pj_sock_t sockfd, const pj_sockaddr_t *my_addr, int addrlen)
This function gives the socket sockfd the local address my_addr. my_addr is addrlen bytes long. Traditionally, this is called assigning a name to a socket. When a socket is created with pj_sock_socket(), it exists in a name space (address family) but has no name assigned.

Parameters:
sockfd  The socket descriptor.
my_addr  The local address to bind the socket to.
addrlen  The length of the address.

Returns:
Zero on success.

6.39.5.29  pj_status_t pj_sock_bind_in (pj_sock_t sockfd, pj_uint32_t addr, pj_uint16_t port)
Bind the IP socket sockfd to the given address and port.

Parameters:
sockfd  The socket descriptor.
addr  Local address to bind the socket to, in host byte order.
port  The local port to bind the socket to, in host byte order.

Returns:
Zero on success.

6.39.5.30  pj_status_t pj_sock_close (pj_sock_t sockfd)
Close the socket descriptor.

Parameters:
sockfd  The socket descriptor.

Returns:
Zero on success.
6.39 Socket Abstraction

6.39.5.31 \texttt{pj\_status\_t pj\_sock\_connect (pj\_t sockfd, const pj\_sockaddr\_t * serv\_addr, int addrlen)}

The file descriptor sockfd must refer to a socket. If the socket is of type PJ\_SOCK\_DGRAM then the serv\_addr address is the address to which datagrams are sent by default, and the only address from which datagrams are received. If the socket is of type PJ\_SOCK\_STREAM or PJ\_SOCK\_SEQPACKET, this call attempts to make a connection to another socket. The other socket is specified by serv\_addr, which is an address (of length addrlen) in the communications space of the socket. Each communications space interprets the serv\_addr parameter in its own way.

\textbf{Parameters:}

- \texttt{sockfd} The socket descriptor.
- \texttt{serv\_addr} Server address to connect to.
- \texttt{addrlen} The length of server address.

\textbf{Returns:}

Zero on success.

6.39.5.32 \texttt{int pj\_SOCK\_DGRAM (void)}

Get PJ\_SOCK\_DGRAM constant

6.39.5.33 \texttt{pj\_status\_t pj\_sock\_getpeername (pj\_t sockfd, pj\_sockaddr\_t * addr, int * namelen)}

Return the address of peer which is connected to socket sockfd.

\textbf{Parameters:}

- \texttt{sockfd} The socket descriptor.
- \texttt{addr} Pointer to sockaddr structure to which the address will be returned.
- \texttt{namelen} Initially the length of the addr. Upon return the value will be set to the actual length of the address.

\textbf{Returns:}

Zero on success.

6.39.5.34 \texttt{pj\_status\_t pj\_sock\_getsockname (pj\_t sockfd, pj\_sockaddr\_t * addr, int * namelen)}

Return the current name of the specified socket.

\textbf{Parameters:}

- \texttt{sockfd} The socket descriptor.
- \texttt{addr} Pointer to sockaddr structure to which the address will be returned.
- \texttt{namelen} Initially the length of the addr. Upon return the value will be set to the actual length of the address.
Returns:
Zero on success.

6.39.5.35  `pj_status_t pj_sock_getsockopt (pj_sock_t sockfd, pj_uint16_t level, pj_uint16_t optname, void *optval, int *optlen)`

Get socket option associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

Parameters:
- `sockfd` The socket descriptor.
- `level` The level which to get the option from.
- `optname` The option name.
- `optval` Identifies the buffer which the value will be returned.
- `optlen` Initially contains the length of the buffer, upon return will be set to the actual size of the value.

Returns:
Zero on success.

6.39.5.36  `pj_status_t pj_sock_listen (pj_sock_t sockfd, int backlog)`

Listen for incoming connection. This function only applies to connection oriented sockets (such as PJ.SOCK_STREAM or PJ.SOCK_SEQPACKET), and it indicates the willingness to accept incoming connections.

Parameters:
- `sockfd` The socket descriptor.
- `backlog` Defines the maximum length the queue of pending connections may grow to.

Returns:
Zero on success.

6.39.5.37  `int pj.SOCK_RAW (void)`

Get PJ.SOCK_RAW constant

6.39.5.38  `int pj.SOCK_RDM (void)`

Get PJ.SOCK_RDM constant
6.39 Socket Abstraction

6.39.5.39  
**pj_status_t pj_sock_recv (pj_sock_t sockfd, void * buf, pj_ssize_t * len, unsigned flags)**

Receives data stream or message coming to the specified socket.

**Parameters:**
- *sockfd* The socket descriptor.
- *buf* The buffer to receive the data or message.
- *len* On input, the length of the buffer. On return, contains the length of data received.
- *flags* Flags (such as *pj_MSG_PEEK()*).

**Returns:**
- PJ_SUCCESS or the error code.

6.39.5.40  
**pj_status_t pj_sock_recvfrom (pj_sock_t sockfd, void * buf, pj_ssize_t * len, unsigned flags, pj_sockaddr_t * from, int * fromlen)**

Receives data stream or message coming to the specified socket.

**Parameters:**
- *sockfd* The socket descriptor.
- *buf* The buffer to receive the data or message.
- *len* On input, the length of the buffer. On return, contains the length of data received.
- *flags* Flags (such as *pj_MSG_PEEK()*).
- *from* If not NULL, it will be filled with the source address of the connection.
- *fromlen* Initially contains the length of from address, and upon return will be filled with the actual length of the address.

**Returns:**
- PJ_SUCCESS or the error code.

6.39.5.41  
**pj_status_t pj_sock_send (pj_sock_t sockfd, const void * buf, pj_ssize_t * len, unsigned flags)**

Transmit data to the socket.

**Parameters:**
- *sockfd* Socket descriptor.
- *buf* Buffer containing data to be sent.
- *len* On input, the length of the data in the buffer. Upon return, it will be filled with the length of data sent.
- *flags* Flags (such as *pj_MSG_DONTROUTE()*).

**Returns:**
- PJ_SUCCESS or the status code.
6.39.5.42  
\texttt{pj\_status\_t pj\_sock\_sendto (pj\_sock\_t sockfd, const void * buf, pj\_ssize\_t * len, unsigned flags, const pj\_sockaddr\_t * to, int tolen)}

Transmit data to the socket to the specified address.

**Parameters:**

- \textit{sockfd}  Socket descriptor.
- \textit{buf} Buffer containing data to be sent.
- \textit{len} On input, the length of the data in the buffer. Upon return, it will be filled with the length of data sent.
- \textit{flags} Flags (such as \texttt{pj\_MSG\_DONTROUTE()}).
- \textit{to} The address to send.
- \textit{tolen} The length of the address in bytes.

**Returns:**

- \texttt{PJ\_SUCCESS} or the status code.

6.39.5.43  
\texttt{pj\_status\_t pj\_sock\_setsockopt (pj\_sock\_t sockfd, pj\_uint16\_t level, pj\_uint16\_t optname, const void * optval, int optlen)}

Manipulate the options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

**Parameters:**

- \textit{sockfd} The socket descriptor.
- \textit{level} The level which to get the option from.
- \textit{optname} The option name.
- \textit{optval} Identifies the buffer which contain the value.
- \textit{optlen} The length of the value.

**Returns:**

- \texttt{PJ\_SUCCESS} or the status code.

6.39.5.44  
\texttt{pj\_status\_t pj\_sock\_shutdown (pj\_sock\_t sockfd, int how)}

The shutdown call causes all or part of a full-duplex connection on the socket associated with sockfd to be shut down.

**Parameters:**

- \textit{sockfd} The socket descriptor.
- \textit{how} If how is PJ\_SHUT\_RD, further receptions will be disallowed. If how is PJ\_SHUT\_WR, further transmissions will be disallowed. If how is PJ\_SHUT\_RDWR, further receptions and transmissions will be disallowed.

**Returns:**

- Zero on success.
Create new socket/endpoint for communication.

Parameters:

- **family** Specifies a communication domain; this selects the protocol family which will be used for communication.
- **type** The socket has the indicated type, which specifies the communication semantics.
- **protocol** Specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family, in which a case protocol can be specified as 0.
- **sock** New socket descriptor, or PJ_INVALID_SOCKET on error.

Returns:

Zero on success.

Get PJ_SOCK_STREAM constant

Get the IP address of an Internet socket address. The address is returned as 32bit value in host byte order.

Parameters:

- **addr** The IP socket address.

Returns:

32bit address, in host byte order.

Get the transport layer port number of an Internet socket address. The port is returned in host byte order.

Parameters:

- **addr** The IP socket address.

Returns:

Port number, in host byte order.
6.39.5.49  pj_status_t pj_sockaddr_in_init (pj_sockaddr_in * addr, const pj_str_t * cp, pj_uint16_t port)

Set the IP address and port of an IP socket address. The string address may be in a standard numbers and dots notation or may be a hostname. If hostname is specified, then the function will resolve the host into the IP address.

Parameters:
   addr  The IP socket address to be set.
   cp    The address string, which can be in a standard dotted numbers or a hostname to be resolved.
   port  The port number, in host byte order.

Returns:
   Zero on success.

6.39.5.50  void pj_sockaddr_in_set_addr (pj_sockaddr_in * addr, pj_uint32_t hostaddr)

Set the IP address of an Internet socket address.

Parameters:
   addr  The IP socket address.
   hostaddr  The host address, in host byte order.

6.39.5.51  void pj_sockaddr_in_set_port (pj_sockaddr_in * addr, pj_uint16_t hostport)

Set the port number of an Internet socket address.

Parameters:
   addr  The IP socket address.
   hostport  The port number, in host byte order.

6.39.5.52  pj_status_t pj_sockaddr_in_set_str_addr (pj_sockaddr_in * addr, const pj_str_t * cp)

Set the IP address of an IP socket address from string address, with resolving the host if necessary. The string address may be in a standard numbers and dots notation or may be a hostname. If hostname is specified, then the function will resolve the host into the IP address.

Parameters:
   addr  The IP socket address to be set.
   cp    The address string, which can be in a standard dotted numbers or a hostname to be resolved.

Returns:
   Zero on success.
6.39 Socket Abstraction

6.39.5.53  

`pj_uint16_t pj_SOL_IP (void)`

Get `PJ_SOL_IP` constant

6.39.5.54  

`pj_uint16_t pj_SOL_IPV6 (void)`

Get `PJ_SOL_IPV6` constant

6.39.5.55  

`pj_uint16_t pj_SOL_SOCKET (void)`

Get `PJ_SOL_SOCKET` constant

6.39.5.56  

`pj_uint16_t pj_SOL_TCP (void)`

Get `PJ_SOL_TCP` constant

6.39.5.57  

`pj_uint16_t pj_SOL_UDP (void)`

Get `PJ_SOL_UDP` constant

6.39.6  

Variable Documentation

6.39.6.1  

`const pj_uint16_t PJ_AF_INET`  

Internet IP protocol.

See also:

pj_AF_INET()

6.39.6.2  

`const pj_uint16_t PJ_AF_INET6`  

IP version 6.

See also:

pj_AF_INET6()

6.39.6.3  

`const pj_uint16_t PJ_AF_IRDA`  

IRDA sockets.

See also:

pj_AF_IRDA()
6.39.6.4 const pj_uint16_t PJ_AF_PACKET

Packet family.

See also:
   pj_AF_PACKET()

6.39.6.5 const pj_uint16_t PJ_AF_UNIX

Unix domain socket.

See also:
   pj_AF_UNIX()

6.39.6.6 const pj_uint16_t PJ_IP_TOS

IP_TOS optname in setsockopt()。

See also:
   pj_IP_TOS()

6.39.6.7 const pj_uint16_t PJ_IPTOS_LOWDELAY

Minimize delays.

See also:
   pj_IPTOS_LOWDELAY()

6.39.6.8 const pj_uint16_t PJ_IPTOS_MINCOST

"filler data" where slow transmission doesn’t matter.

See also:
   pj_IPTOS_MINCOST()

6.39.6.9 const pj_uint16_t PJ_IPTOS_RELIABILITY

Optimize for reliability.

See also:
   pj_IPTOS_RELIABILITY()
6.39 Socket Abstraction

6.39.6.10 const pj_uint16_t PJ_IPTOS_THROUGHPUT

Optimize throughput.

See also:

pj_IPTOS_THROUGHPUT()

6.39.6.11 const int PJ_MSG_DONTROUTE

Don’t route.

See also:

pj_MSG_DONTROUTE()

6.39.6.12 const int PJ_MSG_OOB

Out-of-band messages.

See also:

pj_MSG_OOB()

6.39.6.13 const int PJ_MSG_PEEK

Peek, don’t remove from buffer.

See also:

pj_MSG_PEEK()

6.39.6.14 const pj_uint16_t PJ_SO_RCVBUF

Buffer size for receive.

See also:

pj_SO_RCVBUF()

6.39.6.15 const pj_uint16_t PJ_SO_SNDBUF

Buffer size for send.

See also:

pj_SO_SNDBUF()
6.39.6.16  const pj_uint16_t PJ_SO_TYPE

Socket type.

See also:
  pj_SO_TYPE()

6.39.6.17  const pj_uint16_t PJ_SOCK_DGRAM

Connectionless, unreliable datagrams of fixed maximum lengths.

See also:
  pj_SOCK_DGRAM()

6.39.6.18  const pj_uint16_t PJ_SOCK_RAW

Raw protocol interface.

See also:
  pj_SOCK_RAW()

6.39.6.19  const pj_uint16_t PJ_SOCK_RDM

Reliably-delivered messages.

See also:
  pj_SOCK_RDM()

6.39.6.20  const pj_uint16_t PJ_SOCK_STREAM

Sequenced, reliable, connection-based byte streams.

See also:
  pj_SOCK_STREAM()

6.39.6.21  const pj_uint16_t PJ_SOL_IP

IP level.

See also:
  pj_SOL_IP()
6.39.6.22  const pj_uint16_t PJ_SOL_IPV6

IP version 6.

See also:

pj_SOL_IPV6()

6.39.6.23  const pj_uint16_t PJ_SOL_SOCKET

Socket level.

See also:

pj_SOL_SOCKET()

6.39.6.24  const pj_uint16_t PJ_SOL_TCP

TCP level.

See also:

pj_SOL_TCP()

6.39.6.25  const pj_uint16_t PJ_SOL_UDP

UDP level.

See also:

pj_SOL_UDP()
6.40 Socket select() API.

6.40.1 Detailed Description

This module provides portable abstraction for `select()` like API. The abstraction is needed so that it can utilize various event dispatching mechanisms that are available across platforms. The API is very similar to normal `select()` usage.

6.40.2 Examples

For some examples on how to use the select API, please see:

- Test: Socket Select()

Data Structures

- struct pj_fd_set_t

Functions

- void PJ_FD_ZERO (pj_fd_set_t *fdsetp)
- pj_size_t PJ_FD_COUNT (const pj_fd_set_t *fdsetp)
- void PJ_FD_SET (pj_sock_t fd, pj_fd_set_t *fdsetp)
- void PJ_FD_CLR (pj_sock_t fd, pj_fd_set_t *fdsetp)
- pj_bool_t PJ_FD_ISSET (pj_socket_t fd, const pj_fd_set_t *fdsetp)
- int pj_sock_select (int n, pj_fd_set_t *readfds, pj_fd_set_t *writefds, pj_fd_set_t *exceptfds, const pj_time_val *timeout)

6.40.3 Function Documentation

6.40.3.1 void PJ_FD_CLR (pj_sock_t fd, pj_fd_set_t *fdsetp)

Remove the file descriptor fd from the set pointed to by fdsetp. If fd is not a member of this set, there shall be no effect on the set, nor will an error be returned.

Parameters:

- *fd* The socket descriptor.
- *fdsetp* The descriptor set.

6.40.3.2 pj_size_t PJ_FD_COUNT (const pj_fd_set_t *fdsetp)

This is an internal function, application shouldn’t use this.

Get the number of descriptors in the set. This is defined in sock_select.c This function will only return the number of sockets set from PJ_FD_SET operation. When the set is modified by other means (such as by select()), the count will not be reflected here.
6.40 Socket select() API

Parameters:

- \textit{fdsetp} The descriptor set.

Returns:

- Number of descriptors in the set.

6.40.3.3 \texttt{pj_bool_t PJ_FD_ISSET (pj_sock_t fd, const \texttt{pj_fd_set_t} * \texttt{fdsetp})}

Evaluate to non-zero if the file descriptor \textit{fd} is a member of the set pointed to by \textit{fdsetp}, and shall evaluate to zero otherwise.

Parameters:

- \textit{fd} The socket descriptor.
- \textit{fdsetp} The descriptor set.

Returns:

- Nonzero if \textit{fd} is member of the descriptor set.

6.40.3.4 \texttt{void PJ_FD_SET (pj_sock_t fd, \texttt{pj_fd_set_t} * \texttt{fdsetp})}

Add the file descriptor \textit{fd} to the set pointed to by \textit{fdsetp}. If the file descriptor \textit{fd} is already in this set, there shall be no effect on the set, nor will an error be returned.

Parameters:

- \textit{fd} The socket descriptor.
- \textit{fdsetp} The descriptor set.

6.40.3.5 \texttt{void PJ_FD_ZERO (\texttt{pj_fd_set_t} * \texttt{fdsetp})}

Initialize the descriptor set pointed to by \textit{fdsetp} to the null set.

Parameters:

- \textit{fdsetp} The descriptor set.

6.40.3.6 \texttt{int pj_sock_select (int \textit{n}, \texttt{pj_fd_set_t} * \texttt{readfds}, \texttt{pj_fd_set_t} * \texttt{writefds}, \texttt{pj_fd_set_t} * \texttt{exceptfds}, \texttt{const pj_time_val} * \texttt{timeout})}

This function wait for a number of file descriptors to change status. The behaviour is the same as select() function call which appear in standard BSD socket libraries.

Parameters:

- \textit{n} On Unices, this specifies the highest-numbered descriptor in any of the three set, plus 1. On Windows, the value is ignored.
readfds Optional pointer to a set of sockets to be checked for readability.
writefds Optional pointer to a set of sockets to be checked for writability.
exceptfds Optional pointer to a set of sockets to be checked for errors.
timeout Maximum time for select to wait, or null for blocking operations.

Returns:

The total number of socket handles that are ready, or zero if the time limit expired, or -1 if an error occurred.
6.41 String Operations

6.41.1 Detailed Description

This module provides string manipulation API.

6.41.2 PJLIB String is NOT Null Terminated!

That is the first information that developers need to know. Instead of using normal C string, strings in PJLIB are represented as `pj_str_t` structure below:

```c
typedef struct pj_str_t
{
    char     *ptr;
    pj_size_t slen;
} pj_str_t;
```

There are some advantages of using this approach:

- the string can point to arbitrary location in memory even if the string in that location is not null terminated. This is most useful for text parsing, where the parsed text can just point to the original text in the input. If we use C string, then we will have to copy the text portion from the input to a string variable.

- because the length of the string is known, string copy operation can be made more efficient.

Most of APIs in PJLIB that expect or return string will represent the string as `pj_str_t` instead of normal C string.

6.41.3 Examples

For some examples, please see:

- **Test: String**

Defines

- `#define strnicmp_alnum pj_ansi_strnicmp`
- `#define pj_stricmp_alnum pj_stricmp`

Functions

- `pj_str_t pj_str (char *str)`
- `const pj_str_t * pj_cstr (pj_str_t *str, const char *s)`
- `pj_str_t * pj_strset (pj_str_t *str, char *ptr, pj_size_t length)`
- `pj_str_t * pj_strset2 (pj_str_t *str, char *src)`
- `pj_str_t * pj_strset3 (pj_str_t *str, char *begin, char *end)`
- `pj_str_t * pj_strassign (pj_str_t *dst, pj_str_t *src)`
6.41.4 Define Documentation

6.41.4.1 #define pj_stricmp_alnum pj_stricmp

Perform lowercase comparison to the strings which consists of only alnum characters. More over, it will only return non-zero if both strings are not equal, not the usual negative or positive value.

If non-alnum inputs are given, then the function may mistakenly treat two strings as equal.

Parameters:

str1 The string to compare.
str2 The string to compare.
6.41 String Operations

Returns:

- 0 if str1 is equal to str2
- (-1) if not equal.

6.41.4.2 #define strnicmp_alnum pj_ansi_strnicmp

Perform lowercase comparison to the strings which consists of only alnum characters. More over, it will only return non-zero if both strings are not equal, not the usual negative or positive value.

If non-alnum inputs are given, then the function may mistakenly treat two strings as equal.

Parameters:

- str1 The string to compare.
- str2 The string to compare.
- len The length to compare.

Returns:

- 0 if str1 is equal to str2
- (-1) if not equal.

6.41.5 Function Documentation

6.41.5.1 void pj_bzero (void * dst, pj_size_t size)

Fill the memory location with zero.

Parameters:

- dst The destination buffer.
- size The number of bytes.

6.41.5.2 char* pj_create_random_string (char * str, pj_size_t length)

Initialize the buffer with some random string.

Parameters:

- str the string to store the result.
- length the length of the random string to generate.

Returns:

the string.
6.41.5.3 const pj_str_t pj_cstr (pj_str_t * str, const char * s)

Create constant string from normal C string.

Parameters:

str  The string to be initialized.

s    Null terminated string.

Returns:

pj_str_t.

6.41.5.4 void* pj_memchr (const void * buf, int c, pj_size_t size)

Find character in the buffer.

Parameters:

buf  The buffer.

c    The character to find.

size The size to check.

Returns:

the pointer to location where the character is found, or NULL if not found.

6.41.5.5 int pj_memcmp (const void * buf1, const void * buf2, pj_size_t size)

Compare buffers.

Parameters:

buf1  The first buffer.

buf2  The second buffer.

size  The size to compare.

Returns:

negative, zero, or positive value.

6.41.5.6 void* pj_memcpy (void * dst, const void * src, pj_size_t size)

Copy buffer.

Parameters:

dst  The destination buffer.

src  The source buffer.

size The size to copy.

Returns:

the destination buffer.
6.41.5.7  \texttt{void* pj_memmove (void *dst, const void *src, pj_size_t size)}

Move memory.

**Parameters:**
- \texttt{dst} The destination buffer.
- \texttt{src} The source buffer.
- \texttt{size} The size to copy.

**Returns:**
the destination buffer.

6.41.5.8  \texttt{void* pj_memset (void *dst, int c, pj_size_t size)}

Fill the memory location with value.

**Parameters:**
- \texttt{dst} The destination buffer.
- \texttt{c} Character to set.
- \texttt{size} The number of characters.

**Returns:**
the value of \texttt{dst}.

6.41.5.9  \texttt{pj_str_t pj_str (char *str)}

Create string initializer from a normal C string.

**Parameters:**
- \texttt{str} Null terminated string to be stored.

**Returns:**
\texttt{pj_str_t}.

6.41.5.10 \texttt{pj_str_t* pj_strassign (pj_str_t *dst, pj_str_t *src)}

Assign string.

**Parameters:**
- \texttt{dst} The target string.
- \texttt{src} The source string.

**Returns:**
the target string.
6.41.5.11 const char* pj_strbuf (const pj_str_t * str)

Return the pointer to the string data.

Parameters:

str The string.

Returns:

the pointer to the string buffer.

6.41.5.12 void pj_strcat (pj_str_t * dst, const pj_str_t * src)

Concatenate strings.

Parameters:

dst The destination string.

src The source string.

6.41.5.13 void pj_strcat2 (pj_str_t * dst, const char * src)

Concatenate strings.

Parameters:

dst The destination string.

src The source string.

6.41.5.14 char* pj_strchr (const pj_str_t * str, int chr)

Finds a character in a string.

Parameters:

str The string.

chr The character to find.

Returns:

the pointer to first character found, or NULL.

6.41.5.15 int pj_strcmp (const pj_str_t * str1, const pj_str_t * str2)

Compare strings.

Parameters:

str1 The string to compare.
6.41 String Operations

`str2` The string to compare.

**Returns:**

- `< 0` if `str1` is less than `str2`
- `0` if `str1` is identical to `str2`
- `> 0` if `str1` is greater than `str2`

6.41.5.16 `int pj_strcmp2 (const pj_str_t *str1, const char *str2)`

Compare strings.

**Parameters:**

- `str1` The string to compare.
- `str2` The string to compare.

**Returns:**

- `< 0` if `str1` is less than `str2`
- `0` if `str1` is identical to `str2`
- `> 0` if `str1` is greater than `str2`

6.41.5.17 `pj_str_t* pj_strcpy (pj_str_t *dst, const pj_str_t *src)`

Copy string contents.

**Parameters:**

- `dst` The target string.
- `src` The source string.

**Returns:**

the target string.

6.41.5.18 `pj_str_t* pj_strcpy2 (pj_str_t *dst, const char *src)`

Copy string contents.

**Parameters:**

- `dst` The target string.
- `src` The source string.

**Returns:**

the target string.
6.41.5.19  **pj_str_t* pj_strdup (pj_pool_t * pool, pj_str_t * dst, const pj_str_t * src)**  
Duplicate string.

**Parameters:**
- `pool` The pool.
- `dst` The string result.
- `src` The string to duplicate.

**Returns:**
the string result.

6.41.5.20  **pj_str_t* pj_strdup2 (pj_pool_t * pool, pj_str_t * dst, const char * src)**  
Duplicate string.

**Parameters:**
- `pool` The pool.
- `dst` The string result.
- `src` The string to duplicate.

**Returns:**
the string result.

6.41.5.21  **pj_str_t* pj_strdup2_with_null (pj_pool_t * pool, pj_str_t * dst, const char * src)**  
Duplicate string and NULL terminate the destination string.

**Parameters:**
- `pool` The pool.
- `dst` The string result.
- `src` The string to duplicate.

**Returns:**
The string result.

6.41.5.22  **pj_str_t pj_strdup3 (pj_pool_t * pool, const char * src)**  
Duplicate string.

**Parameters:**
- `pool` The pool.
- `src` The string to duplicate.

**Returns:**
the string result.
6.41 String Operations

6.41.5.23  *pj_strdup_with_null (pj_pool_t * pool, pj_str_t * dst, const pj_str_t * src)*

Duplicate string and NULL terminate the destination string.

Parameters:
- *pool*  The pool.
- *dst* The string result.
- *src* The string to duplicate.

Returns:
The string result.

6.41.5.24  int pj_stricmp (const pj_str_t * str1, const pj_str_t * str2)

Perform case-insensitive comparison to the strings.

Parameters:
- *str1* The string to compare.
- *str2* The string to compare.

Returns:
- \(< 0\) if *str1* is less than *str2*
- \(0\) if *str1* is equal to *str2*
- \(> 0\) if *str1* is greater than *str2*

6.41.5.25  int pj_stricmp2 (const pj_str_t * str1, const char * str2)

Perform case-insensitive comparison to the strings.

Parameters:
- *str1* The string to compare.
- *str2* The string to compare.

Returns:
- \(< 0\) if *str1* is less than *str2*
- \(0\) if *str1* is identical to *str2*
- \(> 0\) if *str1* is greater than *str2*
6.41.5.26 \texttt{pj\_size\_t pj\_strlen (const pj\_str\_t \ast str)}

Return the length of the string.

\textbf{Parameters:}

\textit{str} The string.

\textbf{Returns:}

the length of the string.

6.41.5.27 \texttt{pj\_str\_t \ast pj\_strltrim (pj\_str\_t \ast str)}

Remove (trim) leading whitespaces from the string.

\textbf{Parameters:}

\textit{str} The string.

\textbf{Returns:}

the string.

6.41.5.28 \texttt{int pj\_strncmp (const pj\_str\_t \ast str1, const pj\_str\_t \ast str2, pj\_size\_t len)}

Compare strings.

\textbf{Parameters:}

\textit{str1} The string to compare.
\textit{str2} The string to compare.
\textit{len} The maximum number of characters to compare.

\textbf{Returns:}

- \textless{} 0 if \texttt{str1} is less than \texttt{str2}
- 0 if \texttt{str1} is identical to \texttt{str2}
- \textgreater{} 0 if \texttt{str1} is greater than \texttt{str2}

6.41.5.29 \texttt{int pj\_strncmp2 (const pj\_str\_t \ast str1, const char \ast str2, pj\_size\_t len)}

Compare strings.

\textbf{Parameters:}

\textit{str1} The string to compare.
\textit{str2} The string to compare.
\textit{len} The maximum number of characters to compare.
6.41 String Operations

Returns:

- \(< 0\) if \(str1\) is less than \(str2\)
- \(0\) if \(str1\) is identical to \(str2\)
- \(> 0\) if \(str1\) is greater than \(str2\)

6.41.5.30 \(\text{pj\_str\_t* pj\_strncpy (pj\_str\_t * dst, const pj\_str\_t * src, pj\_ssize\_t max)}\)

Copy source string to destination up to the specified max length.

Parameters:

- \(dst\) The target string.
- \(src\) The source string.
- \(max\) Maximum characters to copy.

Returns:

the target string.

6.41.5.31 \(\text{pj\_str\_t* pj\_strncpy\_with\_null (pj\_str\_t * dst, const pj\_str\_t * src, pj\_ssize\_t max)}\)

Copy source string to destination up to the specified max length, and NULL terminate the destination. If source string length is greater than or equal to max, then max-1 will be copied.

Parameters:

- \(dst\) The target string.
- \(src\) The source string.
- \(max\) Maximum characters to copy.

Returns:

the target string.

6.41.5.32 \(\text{int pj\_strnicmp (const pj\_str\_t * str1, const pj\_str\_t * str2, pj\_size\_t len)}\)

Perform case-insensitive comparison to the strings.

Parameters:

- \(str1\) The string to compare.
- \(str2\) The string to compare.
- \(len\) The maximum number of characters to compare.

Returns:

- \(< 0\) if \(str1\) is less than \(str2\)
- \(0\) if \(str1\) is identical to \(str2\)
- \(> 0\) if \(str1\) is greater than \(str2\)
6.41.5.33 \textbf{int pj_strnicmp2 (const pj_str_t * str1, const char * str2, pj_size_t len)}

Perform case-insensitive comparison to the strings.

**Parameters:**
- \textit{str1} The string to compare.
- \textit{str2} The string to compare.
- \textit{len} The maximum number of characters to compare.

**Returns:**
- \(<0\) if \textit{str1} is less than \textit{str2}
- \(0\) if \textit{str1} is identical to \textit{str2}
- \(>0\) if \textit{str1} is greater than \textit{str2}

6.41.5.34 \textbf{pj_str_t* pj_strrtrim (pj_str_t * str)}

Remove (trim) the trailing whitespaces from the string.

**Parameters:**
- \textit{str} The string.

**Returns:**
- the string.

6.41.5.35 \textbf{pj_str_t* pj_strset (pj_str_t * str, char * ptr, pj_size_t length)}

Set the pointer and length to the specified value.

**Parameters:**
- \textit{str} the string.
- \textit{ptr} pointer to set.
- \textit{length} length to set.

**Returns:**
- the string.

6.41.5.36 \textbf{pj_str_t* pj_strset2 (pj_str_t * str, char * src)}

Set the pointer and length of the string to the source string, which must be NULL terminated.

**Parameters:**
- \textit{str} the string.
- \textit{src} pointer to set.

**Returns:**
- the string.
6.41.5.37  
\texttt{pj\_str\_t* pj\_strset3 (pj\_str\_t* str, char* begin, char* end)}

Set the pointer and the length of the string.

**Parameters:**
- \texttt{str} The target string.
- \texttt{begin} The start of the string.
- \texttt{end} The end of the string.

**Returns:**
the target string.

6.41.5.38  
\texttt{unsigned long pj\_strtoul (const pj\_str\_t* str)}

Convert string to unsigned integer.

**Parameters:**
- \texttt{str} the string.

**Returns:**
the unsigned integer.

6.41.5.39  
\texttt{unsigned long pj\_strtoull (const pj\_str\_t* str, pj\_str\_t* endptr, unsigned base)}

Convert strings to an unsigned long-integer value. This function stops reading the string input either when the number of characters has exceeded the length of the input or it has read the first character it cannot recognize as part of a number, that is a character greater than or equal to base.

**Parameters:**
- \texttt{str} The input string.
- \texttt{endptr} Optional pointer to receive the remainder/unparsed portion of the input.
- \texttt{base} Number base to use.

**Returns:**
the unsigned integer number.

6.41.5.40  
\texttt{pj\_str\_t* pj\_strtrim (pj\_str\_t* str)}

Remove (trim) leading and trailing whitespaces from the string.

**Parameters:**
- \texttt{str} The string.

**Returns:**
the string.
6.41.5.41  int pj_utoa (unsigned long val, char * buf)

Utility to convert unsigned integer to string. Note that the string will be NULL terminated.

Parameters:

val  the unsigned integer value.
buf  the buffer

Returns:

the number of characters written

6.41.5.42  int pj_utoa_pad (unsigned long val, char * buf, int min_dig, int pad)

Convert unsigned integer to string with minimum digits. Note that the string will be NULL terminated.

Parameters:

val  The unsigned integer value.
buf  The buffer.
min_dig Minimum digits to be printed, or zero to specify no minimum digit.
pad  The padding character to be put in front of the string when the digits is less than minimum.

Returns:

the number of characters written.
6.42 Timer Heap Management.

6.42.1 Detailed Description

The timer scheduling implementation here is based on ACE library’s ACE_Timer_Heap, with only little modification to suit our library’s style (I even left most of the comments in the original source).

To quote the original quote in ACE_Timer_Heap_T class:

This implementation uses a heap-based callout queue of absolute times. Therefore, in the average and worst case, scheduling, canceling, and expiring timers is O(log N) (where N is the total number of timers). In addition, we can also preallocate as many ACE_Timer_Nodes as there are slots in the heap. This allows us to completely remove the need for dynamic memory allocation, which is important for real-time systems.

You can find the fine ACE library at: http://www.cs.wustl.edu/~schmidt/ACE.html
ACE is Copyright (C)1993-2006 Douglas C. Schmidt <d.schmidt@vanderbilt.edu>

6.42.2 Examples

For some examples on how to use the timer heap, please see the link below.

- Test: Timer

Data Structures

- struct pj_timer_entry

Typedefs

- typedef int pj_timer_id_t
- typedef void pj_timer_heap_callback (pj_timer_heap_t *timer_heap, struct pj_timer_entry *entry)

Functions

- pj_size_t pj_timer_heap_mem_size (pj_size_t count)
- pj_status_t pj_timer_heap_create (pj_pool_t *pool, pj_size_t count, pj_timer_heap_t **ht)
- void pj_timer_heap_destroy (pj_timer_heap_t *ht)
- void pj_timer_heap_set_lock (pj_timer_heap_t *ht, pj_lock_t *lock, pj_bool_t auto_del)
- unsigned pj_timer_heap_set_max_timed_out_per_poll (pj_timer_heap_t *ht, unsigned count)
- pj_timer_entry * pj_timer_entry_init (pj_timer_entry *entry, int id, void *user_data, pj_timer_heap_callback *cb)
- pj_status_t pj_timer_heap_schedule (pj_timer_heap_t *ht, pj_timer_entry *entry, const pj_time_val *delay)
- int pj_timer_heap_cancel (pj_timer_heap_t *ht, pj_timer_entry *entry)
- pj_size_t pj_timer_heap_count (pj_timer_heap_t *ht)
- pj_status_t pj_timer_heap_earliest_time (pj_timer_heap_t *ht, pj_time_val *timeval)
- unsigned pj_timer_heap_poll (pj_timer_heap_t *ht, pj_time_val *next_delay)
6.42.3  Typedef Documentation

6.42.3.1 typedef void pj_timer_heap_callback(pj_timer_heap_t *timer_heap, struct pj_timer_entry *entry)

The type of callback function to be called by timer scheduler when a timer has expired.

Parameters:

  *timer_heap*  The timer heap.

  *entry*  Timer entry which timer’s has expired.

6.42.3.2 typedef int pj_timer_id_t

The type for internal timer ID.

6.42.4  Function Documentation

6.42.4.1 pj_timer_entry* pj_timer_entry_init (pj_timer_entry* entry, int id, void *user_data, pj_timer_heap_callback *cb)

Initialize a timer entry. Application should call this function at least once before scheduling the entry to the timer heap, to properly initialize the timer entry.

Parameters:

  *entry*  The timer entry to be initialized.

  *id*  Arbitrary ID assigned by the user/owner of this entry. Applications can use this ID to distinguish multiple timer entries that share the same callback and user_data.

  *user_data*  User data to be associated with this entry. Applications normally will put the instance of object that owns the timer entry in this field.

  *cb*  Callback function to be called when the timer elapses.

Returns:

  The timer entry itself.

6.42.4.2 int pj_timer_heap_cancel (pj_timer_heap_t *ht, pj_timer_entry *entry)

Cancel a previously registered timer.

Parameters:

  *ht*  The timer heap.

  *entry*  The entry to be cancelled.

Returns:

  The number of timer cancelled, which should be one if the entry has really been registered, or zero if no timer was cancelled.
6.42 Timer Heap Management.

6.42.4.3  \texttt{pj\_size\_t pj\_timer\_heap\_count (pj\_timer\_heap\_t \ast ht)}

Get the number of timer entries.

\textbf{Parameters:}

- \textit{ht} The timer heap.

\textbf{Returns:}

The number of timer entries.

6.42.4.4  \texttt{pj\_status\_t pj\_timer\_heap\_create (pj\_pool\_t \ast pool, pj\_size\_t count, pj\_timer\_heap\_t \ast\ast ht)}

Create a timer heap.

\textbf{Parameters:}

- \textit{pool} The pool where allocations in the timer heap will be allocated. The timer heap will dynamically allocate more storage from the pool if the number of timer entries registered is more than the size originally requested when calling this function.
- \textit{count} The maximum number of timer entries to be supported initially. If the application registers more entries during runtime, then the timer heap will resize.
- \textit{ht} Pointer to receive the created timer heap.

\textbf{Returns:}

PJ\_SUCCESS, or the appropriate error code.

6.42.4.5  \texttt{void pj\_timer\_heap\_destroy (pj\_timer\_heap\_t \ast ht)}

Destroy the timer heap.

\textbf{Parameters:}

- \textit{ht} The timer heap.

6.42.4.6  \texttt{pj\_status\_t pj\_timer\_heap\_earliest\_time (pj\_timer\_heap\_t \ast ht, pj\_time\_val \ast timeval)}

Get the earliest time registered in the timer heap. The timer heap MUST have at least one timer being scheduled (application should use \texttt{pj\_timer\_heap\_count()} before calling this function).

\textbf{Parameters:}

- \textit{ht} The timer heap.
- \textit{timeval} The time deadline of the earliest timer entry.

\textbf{Returns:}

PJ\_SUCCESS, or PJ\_ENOTFOUND if no entry is scheduled.
6.42.4.7  \texttt{pj\_size\_t pj\_timer\_heap\_mem\_size (pj\_size\_t count)}

Calculate memory size required to create a timer heap.

**Parameters:**

\emph{count} Number of timer entries to be supported.

**Returns:**

Memory size requirement in bytes.

6.42.4.8  \texttt{unsigned pj\_timer\_heap\_poll (pj\_timer\_heap\_t \ast ht, pj\_time\_val \ast next\_delay)}

Poll the timer heap, check for expired timers and call the callback for each of the expired timers. 
Note: polling the timer heap is not necessary in Symbian. Please see Symbian OS Specific for more info.

**Parameters:**

\emph{ht} The timer heap.

\emph{next\_delay} If this parameter is not NULL, it will be filled up with the time delay until the next timer elapsed, or -1 in the sec part if no entry exist.

**Returns:**

The number of timers expired.

6.42.4.9  \texttt{pj\_status\_t pj\_timer\_heap\_schedule (pj\_timer\_heap\_t \ast ht, pj\_timer\_entry \ast entry, const pj\_time\_val \ast delay)}

Schedule a timer entry which will expire AFTER the specified delay.

**Parameters:**

\emph{ht} The timer heap.

\emph{entry} The entry to be registered.

\emph{delay} The interval to expire.

**Returns:**

PJ\_SUCCESS, or the appropriate error code.

6.42.4.10  \texttt{void pj\_timer\_heap\_set\_lock (pj\_timer\_heap\_t \ast ht, pj\_lock\_t \ast lock, pj\_bool\_t auto\_del)}

Set lock object to be used by the timer heap. By default, the timer heap uses dummy synchronization.

**Parameters:**

\emph{ht} The timer heap.

\emph{lock} The lock object to be used for synchronization.

\emph{auto\_del} If nonzero, the lock object will be destroyed when the timer heap is destroyed.
6.42 Timer Heap Management.

6.42.4.11 unsigned pj_timer_heap_set_max_timed_out_per_poll (pj_timer_heap_t * ht, unsigned count)

Set maximum number of timed out entries to process in a single poll.

Parameters:

- \textit{ht} The timer heap.
- \textit{count} Number of entries.

Returns:

The old number.
6.43 PJ Library

Modules

- Build Configuration
- Error Codes
- Data Structure.
- Miscellaneous
- Operating System Dependent Functionality.
- Fast Memory Pool

Memory pools allow dynamic memory allocation comparable to malloc or the new in operator C++. Those implementations are not desirable for very high performance applications or real-time systems, because of the performance bottlenecks and it suffers from fragmentation issue.
6.44 Basic Data Types and Library Functionality.

Data Structures

- struct pj_str_t
- union pj_timestamp
- struct pj_hash_iterator_t

Defines

- #define PJ_T(literal_str) literal_str
- #define PJ_SUCCESS 0
- #define PJ_TRUE 1
- #define PJ_FALSE 0
- #define PJ_ARRAY_SIZE(a) (sizeof(a)/sizeof(a[0]))
- #define PJ_MAXINT32 0x7FFFFFFL
- #define PJ_MAX_OBJ_NAME 32

Typedefs

- typedef unsigned int pj_uint32_t
- typedef short pj_int16_t
- typedef unsigned short pj_uint16_t
- typedef signed char pj_int8_t
- typedef unsigned char pj_uint8_t
- typedef size_t pj_size_t
- typedef long pj_ssize_t
- typedef int pj_status_t
- typedef int pj_bool_t
- typedef char pj_char_t
- typedef pj_ssize_t pj_off_t
- typedef void pj_list_type
- typedef pj_list pj_list
- typedef pj_hash_table_t pj_hash_table_t
- typedef pj_hash_entry pj_hash_entry
- typedef pj_pool_factory pj_pool_factory
- typedef pj_pool_t pj_pool_t
- typedef pj_caching_pool pj_caching_pool
- typedef pj_str_t pj_str_t
- typedef pj_ioqueue_t pj_ioqueue_t
- typedef pj_ioqueue_key_t pj_ioqueue_key_t
- typedef pj_timer_heap_t pj_timer_heap_t
- typedef pj_timer_entry pj_timer_entry
- typedef pj_atomic_t pj_atomic_t
- typedef PJ_ATOMIC_VALUE_TYPE pj_atomic_value_t
- typedef pj_thread_t pj_thread_t
- typedef pj_lock_t pj_lock_t
- typedef pj_mutex_t pj_mutex_t
- typedef pj_sem_t pj_sem_t
typedef pj_event_t pj_event_t
typedef pj_pipe_t pj_pipe_t
typedef void * pj_oshandle_t
typedef long pj_sock_t
typedef void pj_sockaddr_t
typedef pj_sockaddr_in pj_sockaddr_in
typedef unsigned int pj_color_t
typedef int pj_exception_id_t
typedef void(*)(void) pj_exit_callback (void)

Functions

• pj_status_t pj_init (void)
• void pj_shutdown (void)
• pj_status_t pj_atexit (pj_exit_callback func)
• pj_int16_t pj_swap16 (pj_int16_t val16)
• pj_int32_t pj_swap32 (pj_int32_t val32)

Variables

• PJ_BEGIN_DECL typedef int pj_int32_t

6.44.1 Define Documentation

6.44.1.1 #define PJ_ARRAY_SIZE(a) (sizeof(a)/sizeof(a[0]))
Utility macro to compute the number of elements in static array.

6.44.1.2 #define PJ_FALSE 0
False value.

6.44.1.3 #define PJ_MAX_OBJ_NAME 32
Length of object names.

6.44.1.4 #define PJ_MAXINT32 0x7FFFFFFFL
Maximum value for signed 32-bit integer.

6.44.1.5 #define PJ_SUCCESS 0
Status is OK.

6.44.1.6 #define PJ_T(literal_str) literal_str
This macro creates Unicode or ANSI literal string depending whether native platform string is Unicode or ANSI.

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
6.44 Basic Data Types and Library Functionality.

6.44.1.7  
#define PJ_TRUE 1

True value.

6.44.2  Typedef Documentation

6.44.2.1  
typedef struct pj_atomic_t pj_atomic_t

Opaque data type for atomic operations.

6.44.2.2  
typedef PJ_ATOMIC_VALUE_TYPE pj_atomic_value_t

Value type of an atomic variable.

6.44.2.3  
typedef int pj_bool_t

Boolean.

6.44.2.4  
typedef struct pj_caching_pool pj_caching_pool

Forward declaration for caching pool, a pool factory implementation.

6.44.2.5  
typedef char pj_char_t

Native char type, which will be equal to wchar_t for Unicode and char for ANSI.

6.44.2.6  
typedef unsigned int pj_color_t

Color type.

6.44.2.7  
typedef struct pj_event_t pj_event_t

Event object.

6.44.2.8  
typedef int pj_exception_id_t

Exception id.

6.44.2.9  
typedef void(∗) pj_exit_callback(void)

Type of callback to register to pj_atexit().

6.44.2.10  
typedef struct pj_hash_entry pj_hash_entry

Opaque data type for hash entry (only used internally by hash table).
6.44.2.11 typedef struct pj_hash_table_t pj_hash_table_t
Opaque data type for hash tables.

6.44.2.12 typedef short pj_int16_t
Signed 16bit integer.

6.44.2.13 typedef signed char pj_int8_t
Signed 8bit integer.

6.44.2.14 typedef struct pj_ioqueue_key_t pj_ioqueue_key_t
Opaque data type for key that identifies a handle registered to the I/O queue framework.

6.44.2.15 typedef struct pj_ioqueue_t pj_ioqueue_t
Opaque data type for I/O Queue structure.

6.44.2.16 typedef struct pj_list pj_list
List.

6.44.2.17 typedef void pj_list_type
The opaque data type for linked list, which is used as arguments throughout the linked list operations.

6.44.2.18 typedef struct pj_lock_t pj_lock_t
Lock object.

6.44.2.19 typedef struct pj_mutex_t pj_mutex_t
Mutex handle.

6.44.2.20 typedef pj_ssize_t pj_off_t
File offset type.

6.44.2.21 typedef void* pj_oshandle_t
Operating system handle.
6.44 Basic Data Types and Library Functionality.

6.44.22 typedef struct pj_pipe_t pj_pipe_t

Unidirectional stream pipe object.

6.44.23 typedef struct pj_pool_factory pj_pool_factory

Forward declaration for memory pool factory.

6.44.24 typedef struct pj_pool_t pj_pool_t

Opaque data type for memory pool.

6.44.25 typedef struct pj_sem_t pj_sem_t

Semaphore handle.

6.44.26 typedef size_t pj_size_t

Large unsigned integer.

6.44.27 typedef long pj_sock_t

Socket handle.

6.44.28 typedef struct pj_sockaddr_in pj_sockaddr_in

Forward declaration.

6.44.29 typedef void pj_sockaddr_t

Generic socket address.

6.44.30 typedef long pj_ssize_t

Large signed integer.

6.44.31 typedef int pj_status_t

Status code.

6.44.32 typedef struct pj_str_t pj_str_t

This type is used as replacement to legacy C string, and used throughout the library.
6.44.2.33 typedef struct *pj_thread_t pj_thread_t
Thread handle.

6.44.2.34 typedef struct *pj_timer_entry pj_timer_entry
Forward declaration for timer entry.

6.44.2.35 typedef struct *pj_timer_heap_t pj_timer_heap_t
Opaque data to identify timer heap.

6.44.2.36 typedef unsigned short *pj_uint16_t
Unsigned 16bit integer.

6.44.2.37 typedef unsigned int *pj_uint32_t
Unsigned 32bit integer.

6.44.2.38 typedef unsigned char *pj_uint8_t
Unsigned 8bit integer.

6.44.3 Function Documentation

6.44.3.1 *pj_status_t pj_atexit (pj_exit_callback func)
Register cleanup function to be called by PJLIB when pj_shutdown() is called.

Parameters:

func The function to be registered.

Returns:

PJ_SUCCESS on success.

6.44.3.2 *pj_status_t pj_init (void)
Initialize the PJ Library. This function must be called before using the library. The purpose of this function
is to initialize static library data, such as character table used in random string generation, and to initialize
operating system dependent functionality (such as WSAStartup() in Windows).

Returns:

PJ_SUCCESS on success.
6.44.3.3 void pj_shutdown (void)

Shutdown PJLIB.

6.44.3.4 pj_int16_t pj_swap16 (pj_int16_t val16)

Swap the byte order of an 16bit data.

**Parameters:**

- `val16` The 16bit data.

**Returns:**

An 16bit data with swapped byte order.

6.44.3.5 pj_int32_t pj_swap32 (pj_int32_t val32)

Swap the byte order of an 32bit data.

**Parameters:**

- `val32` The 32bit data.

**Returns:**

An 32bit data with swapped byte order.

6.44.4 Variable Documentation

6.44.4.1 PJ_BEGIN_DECL typedef int pj_int32_t

Signed 32bit integer.
6.45 Unicode Support

Files

- file unicode.h
  
  Provides Unicode conversion for Unicode OSes.

Defines

- #define PJ_DECL_UNICODE_TEMP_BUF(var, size)
- #define PJ_STRING_TO_NATIVE(s, buf, max) ((char*)s)
- #define PJ_DECL_ANSI_TEMP_BUF(buf, size)
- #define PJ_NATIVE_TO_STRING(cs, buf, max) ((char*)(const char*)cs)

Functions

- wchar_t * pj_ansi_to_unicode (const char *str, pj_size_t len, wchar_t *wbuf, pj_size_t wbuf_count)
- char * pj_unicode_to_ansi (const wchar_t *wstr, pj_size_t len, char *buf, pj_size_t buf_size)

6.45.1 Define Documentation

6.45.1.1 #define PJ_DECL_ANSI_TEMP_BUF(buf, size)

This macro is used to declare temporary ANSI buffer for Unicode to ANSI conversion, and should be put in declaration section of a block. When PJ_NATIVE_STRING_IS_UNICODE macro is not defined, this macro will expand to nothing.

6.45.1.2 #define PJ_DECL_UNICODE_TEMP_BUF(var, size)

This macro is used to declare temporary Unicode buffer for ANSI to Unicode conversion, and should be put in declaration section of a block. When PJ_NATIVE_STRING_IS_UNICODE macro is not defined, this macro will expand to nothing.

6.45.1.3 #define PJ_NATIVE_TO_STRING(cs, buf, max) ((char*)(const char*)cs)

This macro will convert Unicode string to ANSI, when the platform’s native string is Unicode (PJ_NATIVE_STRING_IS_UNICODE is non-zero).

6.45.1.4 #define PJ_STRING_TO_NATIVE(s, buf, max) ((char*)s)

This macro will convert ANSI string to native, when the platform’s native string is Unicode (PJ_NATIVE_STRING_IS_UNICODE is non-zero).
6.45 Unicode Support

6.45.2 Function Documentation

6.45.2.1 wchar_t* pj_ansi_to_unicode (const char * str, pj_size_t len, wchar_t * wbuf, pj_size_t wbuf_count)

Convert ANSI strings to Unicode strings.

Parameters:

  *str*  The ANSI string to be converted.
  *len*  The length of the input string.
  *wbuf* Buffer to hold the Unicode string output.
  *wbuf_count* Buffer size, in number of elements (not bytes).

Returns:

  The Unicode string, NULL terminated.

6.45.2.2 char* pj_unicode_to Ansi (const wchar_t * wstr, pj_size_t len, char * buf, pj_size_t buf_size)

Convert Unicode string to ANSI string.

Parameters:

  *wstr*  The Unicode string to be converted.
  *len*  The length of the input string.
  *buf*  Buffer to hold the ANSI string output.
  *buf_size* Size of the output buffer.

Returns:

  The ANSI string, NULL terminated.
6.46 Time Data Type and Manipulation.

6.46.1 Detailed Description

This module provides API for manipulating time.

6.46.2 Examples

For examples, please see:

- Test: Sleep, Time, and Timestamp

Data Structures

- struct pj_time_val
- struct pj_parsed_time

Defines

- #define PJ_TIME_VAL_MSEC(t)
- #define PJ_TIME_VAL_EQ(t1, t2)
- #define PJ_TIME_VAL_GT(t1, t2)
- #define PJ_TIME_VAL_GTE(t1, t2)
- #define PJ_TIME_VAL_LT(t1, t2)
- #define PJ_TIME_VAL_LTE(t1, t2)
- #define PJ_TIME_VAL_ADD(t1, t2)
- #define PJ_TIME_VAL_SUB(t1, t2)

Functions

- pj_status_t pj_gettimeofday (pj_time_val *tv)
- pj_status_t pj_time_decode (const pj_time_val *tv, pj_parsed_time *pt)
- pj_status_t pj_time_encode (const pj_parsed_time *pt, pj_time_val *tv)
- pj_status_t pj_time_local_to_gmt (pj_time_val *tv)
- pj_status_t pj_time_gmt_to_local (pj_time_val *tv)
- void pj_time_val_normaze (pj_time_val *t)

6.46.3 Define Documentation

6.46.3.1 #define PJ_TIME_VAL_ADD(t1, t2)

Add t2 to t1 and store the result in t1. Effectively this macro will expand as: (t1 += t2).

Parameters:

- t1 The time value to add.
- t2 The time value to be added to t1.
6.46 Time Data Type and Manipulation.

6.46.3.2 #define PJ_TIME_VAL_EQ(t1, t2)

This macro will check if \( t1 \) is equal to \( t2 \).

**Parameters:**

- \( t1 \) The first time value to compare.
- \( t2 \) The second time value to compare.

**Returns:**

Non-zero if both time values are equal.

6.46.3.3 #define PJ_TIME_VAL_GT(t1, t2)

This macro will check if \( t1 \) is greater than \( t2 \).

**Parameters:**

- \( t1 \) The first time value to compare.
- \( t2 \) The second time value to compare.

**Returns:**

Non-zero if \( t1 \) is greater than \( t2 \).

6.46.3.4 #define PJ_TIME_VAL_GTE(t1, t2)

This macro will check if \( t1 \) is greater than or equal to \( t2 \).

**Parameters:**

- \( t1 \) The first time value to compare.
- \( t2 \) The second time value to compare.

**Returns:**

Non-zero if \( t1 \) is greater than or equal to \( t2 \).

6.46.3.5 #define PJ_TIME_VAL_LT(t1, t2)

This macro will check if \( t1 \) is less than \( t2 \).

**Parameters:**

- \( t1 \) The first time value to compare.
- \( t2 \) The second time value to compare.

**Returns:**

Non-zero if \( t1 \) is less than \( t2 \).
6.46.3.6  \#define PJ_TIME_VAL_LTE(t1, t2)

This macro will check if \( t1 \) is less than or equal to \( t2 \).

**Parameters:**

- \( t1 \) The first time value to compare.
- \( t2 \) The second time value to compare.

**Returns:**

Non-zero if \( t1 \) is less than or equal to \( t2 \).

6.46.3.7  \#define PJ_TIME_VAL_MSEC(t)

Get the total time value in miliseconds. This is the same as multiplying the second part with 1000 and then add the miliseconds part to the result.

**Parameters:**

- \( t \) The time value.

**Returns:**

Total time in miliseconds.

6.46.3.8  \#define PJ_TIME_VAL_SUB(t1, t2)

Subtract \( t2 \) from \( t1 \) and store the result in \( t1 \). Effectively this macro will expand as \((t1 -= t2)\).

**Parameters:**

- \( t1 \) The time value to subsctract.
- \( t2 \) The time value to be substracted from \( t1 \).

6.46.4  Function Documentation

6.46.4.1  \texttt{pj_status_t pj_gettimeofday (pj_time_val * tv)}

Get current time of day in local representation.

**Parameters:**

- \( tv \) Variable to store the result.

**Returns:**

zero if successful.
6.46 Time Data Type and Manipulation.

6.46.4.2  \texttt{pj\_status\_t pj\_time\_decode (const pj\_time\_val * tv, pj\_parsed\_time * pt)}

Parse time value into date/time representation.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{tv} The time.
  \item \textit{pt} Variable to store the date time result.
\end{itemize}

\textbf{Returns:}
\begin{itemize}
  \item zero if successful.
\end{itemize}

6.46.4.3  \texttt{pj\_status\_t pj\_time\_encode (const pj\_parsed\_time * pt, pj\_time\_val * tv)}

Encode date/time to time value.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{pt} The date/time.
  \item \textit{tv} Variable to store time value result.
\end{itemize}

\textbf{Returns:}
\begin{itemize}
  \item zero if successful.
\end{itemize}

6.46.4.4  \texttt{pj\_status\_t pj\_time\_gmt\_to\_local (pj\_time\_val * tv)}

Convert GMT to local time.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{tv} Time to convert.
\end{itemize}

\textbf{Returns:}
\begin{itemize}
  \item zero if successful.
\end{itemize}

6.46.4.5  \texttt{pj\_status\_t pj\_time\_local\_to\_gmt (pj\_time\_val * tv)}

Convert local time to GMT.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{tv} Time to convert.
\end{itemize}

\textbf{Returns:}
\begin{itemize}
  \item zero if successful.
\end{itemize}

6.46.4.6  \texttt{void pj\_time\_val\_normalize (pj\_time\_val * t)}

Normalize the value in time value.

\textbf{Parameters:}
\begin{itemize}
  \item \textit{t} Time value to be normalized.
\end{itemize}
Chapter 7

PJLIB Reference Data Structure Documentation

7.1  pj_addr_hdr Struct Reference

7.1.1  Detailed Description

This structure describes common attributes found in transport addresses. If PJ_SOCKADDR_HAS_LEN is not zero, then sa_zero_len member is added to this struct. As far the application is concerned, the value of this member will always be zero. Internally, PJLIB may modify the value before calling OS socket API, and reset the value back to zero before returning the struct to application.

Data Fields

•  pj_uint16_t sa_family

7.1.2  Field Documentation

7.1.2.1  pj_uint16_t pj_addr_hdr::sa_family

Common data: address family.
The documentation for this struct was generated from the following file:

•  sock.h
7.2  pj_caching_pool Struct Reference

7.2.1  Detailed Description

Declaration for caching pool. Application doesn’t normally need to care about the contents of this struct, it is only provided here because application need to define an instance of this struct (we can not allocate the struct from a pool since there is no pool factory yet!).

Data Fields

- `pj_pool_factory factory`
- `pj_size_t capacity`
- `pj_size_t max_capacity`
- `pj_size_t used_count`
- `pj_size_t used_size`
- `pj_size_t peak_used_size`
- `pj_list free_list [16]`
- `pj_list used_list`
- `char pool_buf [256 * (sizeof(long)/4)]`
- `pj_lock_t * lock`

7.2.2  Field Documentation

7.2.2.1  `pj_pool_factory pj_caching_pool::factory`

Pool factory interface, must be declared first.

7.2.2.2  `pj_size_t pj_caching_pool::capacity`

Current factory’s capacity, i.e. number of bytes that are allocated and available for application in this factory. The factory’s capacity represents the size of all pools kept by this factory in it’s free list, which will be returned to application when it requests to create a new pool.

7.2.2.3  `pj_size_t pj_caching_pool::max_capacity`

Maximum size that can be held by this factory. Once the capacity has exceeded `max_capacity`, further `pj_pool_release()` will flush the pool. If the capacity is still below the `max_capacity`, `pj_pool_release()` will save the pool to the factory’s free list.

7.2.2.4  `pj_size_t pj_caching_pool::used_count`

Number of pools currently held by applications. This number gets incremented everytime `pj_pool_create()` is called, and gets decremented when `pj_pool_release()` is called.

7.2.2.5  `pj_size_t pj_caching_pool::used_size`

Total size of memory currently used by application.
7.2.2.6 `pj_size_t pj_caching_pool::peak_used_size`

The maximum size of memory used by application throughout the life of the caching pool.

7.2.2.7 `pj_list pj_caching_pool::free_list[16]`

Lists of pools in the cache, indexed by pool size.

7.2.2.8 `pj_list pj_caching_pool::used_list`

List of pools currently allocated by applications.

7.2.2.9 `char pj_caching_pool::pool_buf[256 * (sizeof(long)/4)]`

Internal pool.

7.2.2.10 `pj_lock_t* pj_caching_pool::lock`

Mutex.

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

- pool.h
- pool_alt.h
7.3  pj_exception_state_t Struct Reference

7.3.1  Detailed Description

This structure (which should be invisible to user) manages the TRY handler stack.

Data Fields

- pj_exception_state_t * prev
- pj_jmp_buf state

7.3.2  Field Documentation

7.3.2.1  struct pj_exception_state_t* pj_exception_state_t::prev

Previous state in the list.

7.3.2.2  pj_jmp_buf pj_exception_state_t::state

jmp_buf.

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

- except.h
7.4 pj_fd_set_t Struct Reference

7.4.1 Detailed Description

Portable structure declarations for pj_fd_set. The implementation of `pj_sock_select()` does not use this structure per-se, but instead it will use the native fd_set structure. However, we must make sure that the size of `pj_fd_set_t` can accommodate the native fd_set structure.

Data Fields

- `pj_sock_t data [FD_SETSIZE+4]`

7.4.2 Field Documentation

7.4.2.1 `pj_sock_t pj_fd_set_t::data[FD_SETSIZE+4]`

Opaque buffer for fd_set

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

- `sock_select.h`
7.5  pj_file_stat Struct Reference

7.5.1  Detailed Description

This structure describes file information, to be obtained by calling `pj_file_getstat()`. The time information in this structure is in local time.

Data Fields

- `pj_off_t size`
- `pj_time_val atime`
- `pj_time_val mtime`
- `pj_time_val ctime`

7.5.2  Field Documentation

7.5.2.1  `pj_off_t pj_file_stat::size`

Total file size.

7.5.2.2  `pj_time_val pj_file_stat::atime`

Time of last access.

7.5.2.3  `pj_time_val pj_file_stat::mtime`

Time of last modification.

7.5.2.4  `pj_time_val pj_file_stat::ctime`

Time of last creation.

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

- `file_access.h`
7.6 pj_hash_iterator_t Struct Reference

7.6.1 Detailed Description

Data type for hash search iterator. This structure should be opaque, however applications need to declare concrete variable of this type, that’s why the declaration is visible here.

Data Fields

- `pj_uint32_t index`
- `pj_hash_entry * entry`

7.6.2 Field Documentation

7.6.2.1 `pj_uint32_t pj_hash_iterator_t::index`

Internal index.

7.6.2.2 `pj_hash_entry* pj_hash_iterator_t::entry`

Internal entry.

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

- `types.h`
7.7  pj_hostent Struct Reference

7.7.1  Detailed Description

This structure describes an Internet host address.

**Data Fields**

- char * h_name
- char ** h_aliases
- int h_addrtype
- int h_length
- char ** h_addr_list

7.7.2  Field Documentation

7.7.2.1  char* pj_hostent::h_name

The official name of the host.

7.7.2.2  char** pj_hostent::h_aliases

Aliases list.

7.7.2.3  int pj_hostent::h_addrtype

Host address type.

7.7.2.4  int pj_hostent::h_length

Length of address.

7.7.2.5  char** pj_hostent::h_addr_list

List of addresses.
The documentation for this struct was generated from the following file:

- addr_resolv.h
This structure describes IPv6 address.

Data Fields

• union {
  pj_uint8_t u6_addr8 [16]
  pj_uint16_t u6_addr16 [8]
  pj_uint32_t u6_addr32 [4]
} in6_u

Field Documentation

7.8.2.1 pj_uint8_t pj_in6_addr::u6_addr8[16]

u6_addr8

7.8.2.2 pj_uint16_t pj_in6_addr::u6_addr16[8]

u6_addr16

7.8.2.3 pj_uint32_t pj_in6_addr::u6_addr32[4]

u6_addr32

7.8.2.4 union { ... } pj_in6_addr::in6_u

Union of address formats.
The documentation for this struct was generated from the following file:

• sock.h
7.9    pj_in_addr Struct Reference

7.9.1 Detailed Description

This structure describes Internet address.

Data Fields

• pj_uint32_t s_addr

7.9.2 Field Documentation

7.9.2.1 pj_uint32_t pj_in_addr::s_addr

The 32bit IP address.
The documentation for this struct was generated from the following file:

• sock.h
7.10 pj_ioqueue_callback Struct Reference

7.10.1 Detailed Description

This structure describes the callbacks to be called when I/O operation completes.

Data Fields

- void(* on_read_complete )(pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_ssize_t bytes_read)
- void(* on_write_complete )(pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_ssize_t bytes_sent)
- void(* on_accept_complete )(pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_sock_t sock, pj_status_t status)
- void(* on_connect_complete )(pj_ioqueue_key_t *key, pj_status_t status)

7.10.2 Field Documentation

7.10.2.1 void(* pj_ioqueue_callback::on_read_complete)(pj_ioqueue_key_t *key,
pj_ioqueue_op_key_t *op_key, pj_ssize_t bytes_read)

This callback is called when pj_ioqueue_recv or pj_ioqueue_recvfrom completes.

Parameters:

- key The key.
- op_key Operation key.
- bytes_read >= 0 to indicate the amount of data read, otherwise negative value containing the error code. To obtain the pj_status_t error code, use (pj_status_t code = -bytes_read).

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

- ioqueue.h
7.11 pj_ioqueue_op_key_t Struct Reference

7.11.1 Detailed Description

This structure describes operation specific key to be submitted to I/O Queue when performing the asynchronous operation. This key will be returned to the application when completion callback is called.

Application normally wants to attach its specific data in the user_data field so that it can keep track of which operation has completed when the callback is called. Alternatively, application can also extend this struct to include its data, because the pointer that is returned in the completion callback will be exactly the same as the pointer supplied when the asynchronous function is called.

Data Fields

- void * internal__ [32]
- void * user_data

7.11.2 Field Documentation

7.11.2.1 void* pj_ioqueue_op_key_t::internal__[32]

Internal I/O Queue data.

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

- ioqueue.h
7.12 pj_ip_route_entry Union Reference

7.12.1 Detailed Description

This structure describes IP routing entry.

Data Fields

- struct {
  pj_in_addr if_addr
  pj_in_addr dst_addr
  pj_in_addr mask
} ipv4

7.12.2 Field Documentation

7.12.2.1 pj_in_addr pj_ip_route_entry::if_addr

Local interface IP address.

7.12.2.2 pj_in_addr pj_ip_route_entry::dst_addr

Destination IP address.

7.12.2.3 pj_in_addr pj_ip_route_entry::mask

Destination mask.

7.12.2.4 struct { ... } pj_ip_route_entry::ipv4

IP routing entry for IP version 4 routing

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

- ip_helper.h
7.13  pj_list Struct Reference

7.13.1  Detailed Description

This structure describes generic list node and list. The owner of this list must initialize the 'value' member to an appropriate value (typically the owner itself).

Data Fields

• void * prev
• void * next

7.13.2  Field Documentation

7.13.2.1  void* pj_list::prev

List a prev.

7.13.2.2  void* pj_list::next

List a next.
The documentation for this struct was generated from the following file:

• list.h
7.14 pj_parsed_time Struct Reference

7.14.1 Detailed Description

This structure represents the parsed representation of time. It is acquired by calling `pj_time_decode()`.

Data Fields

- int `wday`
- int `day`
- int `mon`
- int `year`
- int `sec`
- int `min`
- int `hour`
- int `msec`

7.14.2 Field Documentation

7.14.2.1 int `pj_parsed_time::wday`

This represents the day of the week where value zero means Sunday.

7.14.2.2 int `pj_parsed_time::day`

This represents the day of the month: 1-31.

7.14.2.3 int `pj_parsed_time::mon`

This represents the month, with the value being 0 - 11 (zero is January).

7.14.2.4 int `pj_parsed_time::year`

This represents the actual year (unlike in ANSI libc where the value must be added by 1900).

7.14.2.5 int `pj_parsed_time::sec`

This represents the second part, with the value 0-59.

7.14.2.6 int `pj_parsed_time::min`

This represents the minute part, with the value 0-59.

7.14.2.7 int `pj_parsed_time::hour`

This represents the hour part, with the value 0-23.
7.14.2.8 int pj_parsed_time::msec

This represents the milisecond part, with the value is 0-999

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

- types.h
7.15 pj_pool_block Struct Reference

7.15.1 Detailed Description

This class, which is used internally by the pool, describes a single block of memory from which user memory allocations will be allocated from.

Public Member Functions

- PJ_DECL_LIST_MEMBER (struct pj_pool_block)

Data Fields

- unsigned char * buf
- unsigned char * cur
- unsigned char * end

7.15.2 Member Function Documentation

7.15.2.1 pj_pool_block::PJ_DECL_LIST_MEMBER (struct pj_pool_block)

List’s prev and next.

7.15.3 Field Documentation

7.15.3.1 unsigned char * pj_pool_block::buf

Start of buffer.

7.15.3.2 unsigned char * pj_pool_block::cur

Current alloc ptr.

7.15.3.3 unsigned char * pj_pool_block::end

End of buffer.

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

- pool.h
7.16  

### pj_pool_factory Struct Reference

#### 7.16.1 Detailed Description

This structure contains the declaration for pool factory interface.

#### Data Fields

- `pj_pool_factory_policy policy`
- `pj_pool_t *(*create_pool)(pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)`
- `void(*release_pool)(pj_pool_factory *factory, pj_pool_t *pool)`
- `void(*dump_status)(pj_pool_factory *factory, pj_bool_t detail)`
- `pj_pool_t(*on_block_alloc)(pj_pool_factory *factory, pj_size_t size)`
- `void(*on_block_free)(pj_pool_factory *factory, pj_size_t size)`
- `int dummy`

#### 7.16.2 Field Documentation

##### 7.16.2.1 pj_pool_factory_policy pj_pool_factory::policy

Memory pool policy.

##### 7.16.2.2 pj_pool_t *(*create_pool)(pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)

Create a new pool from the pool factory.

**Parameters:**

- **factory** The pool factory.
- **name** the name to be assigned to the pool. The name should not be longer than PJ_MAX_OBJ_NAME (32 chars), or otherwise it will be truncated.
- **initial_size** the size of initial memory blocks taken by the pool. Note that the pool will take 68+20 bytes for administrative area from this block.
- **increment_size** the size of each additional blocks to be allocated when the pool is running out of memory. If user requests memory which is larger than this size, then an error occurs. Note that each time a pool allocates additional block, it needs 20 bytes (equal to sizeof(pj_pool_block)) to store some administrative info.
- **callback** Cllback to be called when error occurs in the pool. Note that when an error occurs during pool creation, the callback itself is not called. Instead, NULL will be returned.

**Returns:**

the memory pool, or NULL.
7.16.2.3  void(* pj_pool_factory::release_pool)(pj_pool_factory *factory, pj_pool_t *pool)

Release the pool to the pool factory.

Parameters:

    factory  The pool factory.
    pool     The pool to be released.

7.16.2.4  void(* pj_pool_factory::dump_status)(pj_pool_factory *factory, pj_bool_t detail)

Dump pool status to log.

Parameters:

    factory  The pool factory.

7.16.2.5  pj_bool_t(* pj_pool_factory::on_block_alloc)(pj_pool_factory *factory, pj_size_t size)

This is optional callback to be called by allocation policy when it allocates a new memory block. The factory may use this callback for example to keep track of the total number of memory blocks currently allocated by applications.

Parameters:

    factory  The pool factory.
    size     Size requested by application.

Returns:

    MUST return PJ_TRUE, otherwise the block allocation is cancelled.

7.16.2.6  void(* pj_pool_factory::on_block_free)(pj_pool_factory *factory, pj_size_t size)

This is optional callback to be called by allocation policy when it frees memory block. The factory may use this callback for example to keep track of the total number of memory blocks currently allocated by applications.

Parameters:

    factory  The pool factory.
    size     Size freed.

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

- pool.h
- pool_alt.h
7.17  pj_pool_factory_policy Struct Reference

7.17.1  Detailed Description

This structure declares pool factory interface.

**Data Fields**

- void *(block_alloc )(pj_pool_factory *factory, pj_size_t size)
- void *(block_free )(pj_pool_factory *factory, void *mem, pj_size_t size)
- pj_pool_callback * callback
- unsigned flags

7.17.2  Field Documentation

7.17.2.1  void *(pj_pool_factory_policy::block_alloc )(pj_pool_factory *factory, pj_size_t size)

Allocate memory block (for use by pool). This function is called by memory pool to allocate memory block.

**Parameters:**

- *factory*  Pool factory.
- *size*  The size of memory block to allocate.

**Returns:**

Memory block.

7.17.2.2  void *(pj_pool_factory_policy::block_free )(pj_pool_factory *factory, void *mem, pj_size_t size)

Free memory block.

**Parameters:**

- *factory*  Pool factory.
- *mem*  Memory block previously allocated by block_alloc().
- *size*  The size of memory block.

7.17.2.3  pj_pool_callback * pj_pool_factory_policy::callback

Default callback to be called when memory allocation fails.

7.17.2.4  unsigned pj_pool_factory_policy::flags

Option flags.

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

- pool.h
7.18  pj_pool_t Struct Reference

7.18.1  Detailed Description

This structure describes the memory pool. Only implementors of pool factory need to care about the contents of this structure.

Public Member Functions

- PJ_DECL_LIST_MEMBER (struct pj_pool_t)

Data Fields

- char obj_name [PJ_MAX_OBJ_NAME]
- pj_pool_factory * factory
- void * factory_data
- pj_size_t capacity
- pj_size_t increment_size
- pj_pool_block block_list
- pj_pool_callback * callback

7.18.2  Member Function Documentation

7.18.2.1  pj_pool_t::PJ_DECL_LIST_MEMBER (struct pj_pool_t)

Standard list elements.

7.18.3  Field Documentation

7.18.3.1  char pj_pool_t::obj_name[PJ_MAX_OBJ_NAME]

Pool name

7.18.3.2  pj_pool_factory* pj_pool_t::factory

Pool factory.

7.18.3.3  void* pj_pool_t::factory_data

Data put by factory

7.18.3.4  pj_size_t pj_pool_t::capacity

Current capacity allocated by the pool.
7.18.3.5  \texttt{pj\_size\_t pj\_pool\_t::increment\_size}

Size of memory block to be allocated when the pool runs out of memory.

7.18.3.6  \texttt{pj\_pool\_block pj\_pool\_t::block\_list}

List of memory blocks allocated by the pool.

7.18.3.7  \texttt{pj\_pool\_callback\* pj\_pool\_t::callback}

The callback to be called when the pool is unable to allocate memory.

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

- pool.h
7.19 pj_rbtree Struct Reference

7.19.1 Detailed Description

Declaration of a red-black tree. All elements in the tree must have UNIQUE key. A red black tree always maintains the balance of the tree, so that the tree height will not be greater than \( \lg(N) \). Insert, search, and delete operation will take \( \lg(N) \) on the worst case. But for insert and delete, there is additional time needed to maintain the balance of the tree.

Data Fields

- \texttt{pj_rbtree::null_node}
- \texttt{pj_rbtree_node * null}
- \texttt{pj_rbtree_node * root}
- \texttt{unsigned size}
- \texttt{pj_rbtree_comp * comp}

7.19.2 Field Documentation

7.19.2.1 \texttt{pj_rbtree_node pj_rbtree::null_node}

Constant to indicate NULL node.

7.19.2.2 \texttt{pj_rbtree_node* pj_rbtree::null}

Constant to indicate NULL node.

7.19.2.3 \texttt{pj_rbtree_node* pj_rbtree::root}

Root tree node.

7.19.2.4 \texttt{unsigned pj_rbtree::size}

Number of elements in the tree.

7.19.2.5 \texttt{pj_rbtree_comp* pj_rbtree::comp}

Key comparison function.

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

- \texttt{rbtree.h}
7.20  pj_rbtree_node Struct Reference

7.20.1  Detailed Description

The type of the node of the R/B Tree.

**Data Fields**

- `pj_rbtree_node * parent`
- `pj_rbtree_node * left`
- `pj_rbtree_node * right`
- `const void * key`
- `void * user_data`
- `pj_rbcolor_t color`

7.20.2  Field Documentation

7.20.2.1  `struct pj_rbtree_node * pj_rbtree_node::parent`

Pointers to the node’s parent, and left and right siblings.

7.20.2.2  `const void * pj_rbtree_node::key`

Key associated with the node.

7.20.2.3  `void * pj_rbtree_node::user_data`

User data associated with the node.

7.20.2.4  `pj_rbcolor_t pj_rbtree_node::color`

The R/B Tree node color.

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

- `rbtree.h`
7.21  pj_sockaddr Union Reference

7.21.1  Detailed Description

This union describes a generic socket address.

Data Fields

- pj_addr_hdr addr
- pj_sockaddr_in ipv4
- pj_sockaddr_in6 ipv6

7.21.2  Field Documentation

7.21.2.1  pj_addr_hdr pj_sockaddr::addr

Generic transport address.

7.21.2.2  pj_sockaddr_in pj_sockaddr::ipv4

IPv4 transport address.

7.21.2.3  pj_sockaddr_in6 pj_sockaddr::ipv6

IPv6 transport address.

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

- sock.h
### 7.22 pj_sockaddr_in Struct Reference

#### 7.22.1 Detailed Description

This structure describes Internet socket address. If PJ_SOCKADDR_HAS_LEN is not zero, then sin_zero_len member is added to this struct. As far the application is concerned, the value of this member will always be zero. Internally, PJLIB may modify the value before calling OS socket API, and reset the value back to zero before returning the struct to application.

#### Data Fields

- `pj_uint16_t sin_family`
- `pj_uint16_t sin_port`
- `pj_in_addr sin_addr`
- `char sin_zero [8]`

#### 7.22.2 Field Documentation

##### 7.22.2.1 `pj_uint16_t pj_sockaddr_in::sin_family`

Address family.

##### 7.22.2.2 `pj_uint16_t pj_sockaddr_in::sin_port`

Transport layer port number.

##### 7.22.2.3 `pj_in_addr pj_sockaddr_in::sin_addr`

IP address.

##### 7.22.2.4 `char pj_sockaddr_in::sin_zero[8]`

Padding.

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

- `sock.h`
7.23 pj_sockaddr_in6 Struct Reference

7.23.1 Detailed Description

This structure describes IPv6 socket address. If PJ_SOCKADDR_HAS_LEN is not zero, then sin_zero_len member is added to this struct. As far the application is concerned, the value of this member will always be zero. Internally, PJLIB may modify the value before calling OS socket API, and reset the value back to zero before returning the struct to application.

Data Fields

- pj_uint16_t sin6_family
- pj_uint16_t sin6_port
- pj_uint32_t sin6_flowinfo
- pj_in6_addr sin6_addr
- pj_uint32_t sin6_scope_id

7.23.2 Field Documentation

7.23.2.1 pj_uint16_t pj_sockaddr_in6::sin6_family

Address family

7.23.2.2 pj_uint16_t pj_sockaddr_in6::sin6_port

Transport layer port number.

7.23.2.3 pj_uint32_t pj_sockaddr_in6::sin6_flowinfo

IPv6 flow information

7.23.2.4 pj_in6_addr pj_sockaddr_in6::sin6_addr

IPv6 address.

7.23.2.5 pj_uint32_t pj_sockaddr_in6::sin6_scope_id

IPv6 scope-id

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

- sock.h
7.24  pj_str_t Struct Reference

7.24.1  Detailed Description

This type is used as replacement to legacy C string, and used throughout the library. By convention, the string is NOT null terminated.

Data Fields

- char * ptr
- pj_ssize_t slen

7.24.2  Field Documentation

7.24.2.1  char* pj_str_t::ptr

Buffer pointer, which is by convention NOT null terminated.

7.24.2.2  pj_ssize_t pj_str_t::slen

The length of the string.

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

- types.h
7.25 pj_time_val Struct Reference

7.25.1 Detailed Description

Representation of time value in this library. This type can be used to represent either an interval or a specific time or date.

Data Fields

- long sec
- long msec

7.25.2 Field Documentation

7.25.2.1 long pj_time_val::sec

The seconds part of the time.

7.25.2.2 long pj_time_val::msec

The miliseconds fraction of the time.
The documentation for this struct was generated from the following file:

- types.h
7.26  pj_timer_entry Struct Reference

7.26.1  Detailed Description

This structure represents an entry to the timer.

Data Fields

- void * user_data
- int id
- pj_timer_heap_callback * cb
- pj_timer_id_t timer_id
- pj_time_val timer_value

7.26.2  Field Documentation

7.26.2.1  void* pj_timer_entry::user_data

User data to be associated with this entry. Applications normally will put the instance of object that owns the timer entry in this field.

7.26.2.2  int pj_timer_entry::id

Arbitrary ID assigned by the user/owner of this entry. Applications can use this ID to distinguish multiple timer entries that share the same callback and user_data.

7.26.2.3  pj_timer_heap_callback* pj_timer_entry::cb

Callback to be called when the timer expires.

7.26.2.4  pj_timer_id_t pj_timer_entry::timer_id

Internal unique timer ID, which is assigned by the timer heap. Application should not touch this ID.

7.26.2.5  pj_time_val pj_timer_entry::timer_value

The future time when the timer expires, which the value is updated by timer heap when the timer is scheduled.

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

- timer.h
7.27 pj_timestamp Union Reference

7.27.1 Detailed Description

This structure represents high resolution (64bit) time value. The time values represent time in cycles, which is retrieved by calling `pj_get_timestamp()`.

**Data Fields**

- struct {
  - `pj_uint32_t` hi
  - `pj_uint32_t` lo
} u32

7.27.2 Field Documentation

7.27.2.1 `pj_uint32_t` pj_timestamp::hi

high 32-bit value of the 64-bit value.

7.27.2.2 `pj_uint32_t` pj_timestamp::lo

Low 32-bit value of the 64-bit value.

7.27.2.3 struct { ... } pj_timestamp::u32

The 64-bit value as two 32-bit values.

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

- `types.h`
Chapter 8

PJLIB Reference File Documentation

8.1 addr_resolv.h File Reference

8.1.1 Detailed Description

IP address resolution.

Data Structures

• struct pj_hostent

Defines

• #define h_addr h_addr_list[0]

Functions

• pj_status_t pj_gethostbyname (const pj_str_t *name, pj_hostent *he)
• pj_status_t pj_gethostip (pj_in_addr *ip_addr)
8.2 array.h File Reference

8.2.1 Detailed Description

PJLIB Array helper.

Functions

- void *pj_array_insert (void *array, unsigned elem_size, unsigned count, unsigned pos, const void *value)
- void *pj_array_erase (void *array, unsigned elem_size, unsigned count, unsigned pos)
- pj_status_t *pj_array_find (const void *array, unsigned elem_size, unsigned count, void *result)
8.3 assert.h File Reference

8.3.1 Detailed Description

Assertion macro `pj_assert()`.

Defines

- `#define pj_assert(expr)`
- `#define PJ_ASSERT_RETURN(expr, retval)`
- `#define PJ_ASSERT_ON_FAIL(expr, exec_on_fail)`
8.4 config.h File Reference

8.4.1 Detailed Description

PJLIB Main configuration settings.

Defines

- `#define PJ_DEBUG 1`
- `#define PJ_FUNCTIONS_ARE_INLINED 0`
- `#define PJ_HAS_FLOATING_POINT 1`
- `#define PJ_LOG_MAX_SIZE 2000`
- `#define PJ_LOG_USE_STACK_BUFFER 1`
- `#define PJ_TERM_HAS_COLOR 1`
- `#define PJ_SAFE_POOL 0`
- `#define PJ_POOL_DEBUG 0`
- `#define PJ_THREAD_DEFAULT_STACK_SIZE 8192`
- `#define PJ_HAS_POOL_ALT_API PJ_POOL_DEBUG`
- `#define PJ_MAX_HOSTNAME (128)`
- `#define PJ_IOQUEUE_MAX_HANDLES (256)`
- `#define PJ_IOQUEUE_HAS_SAFE_UNREG 1`
- `#define PJ_IOQUEUE_KEY_FREE_DELAY 500`
- `#define FD_SETSIZE PJ_IOQUEUE_MAX_HANDLES`
- `#define PJ_MAXPATH 260`
- `#define PJ_ENABLE_EXTRA_CHECK 1`
- `#define PJ_HAS_EXCEPTION_NAMES 1`
- `#define PJ_MAX_EXCEPTION_ID 16`
- `#define PJ_EXCEPTION_USE_WIN32_SEH 0`
- `#define PJ_TIMESTAMP_USE_RDTSC 0`
- `#define PJ_NATIVE_ERR_POSITIVE 1`
- `#define PJ_HAS_ERROR_STRING 1`
- `#define PJ_HAS_STRICMP_ALNUM 0`
- `#define PJINLINE(type) PJINLINE_SPECIFIER type`
- `#define PJDECL_NO_RETURN(type) PJNORETURN type`
- `#define PJ IDECL_NO_RETURN(type) PJNORETURN PJINLINE(type)`
- `#define PJ_BEGINDECL`
- `#define PJ_ENDDECL`
- `#define PJ EXPORT_SYMBOL(x)`
- `#define PJDECL_DATA(type) extern type`
- `#define PJDEF_DATA(type) type`
- `#define PJIDECL(type) PJDECL(type)`
- `#define PJIDEF(type) PJDEF(type)`
- `#define PJUNUSED_ARG(arg) (void)arg`
- `#define PJTODO(id) TODO___##id:
- `#define __pj_throw__(x)`

Functions

- `const char * pj_get_version (void)`
- `void pj_dump_config (void)`,

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
Variables

- const char * PJ_VERSION

8.4.2 Define Documentation

8.4.2.1 #define __pj_throw__(x)

Function attributes to inform that the function may throw exception.

Parameters:

  x The exception list, enclosed in parenthesis.

8.4.2.2 #define PJ_BEGIN_DECL

Mark beginning of declaration section in a header file.

8.4.2.3 #define PJ_DECL_DATA(type) extern type

Parameters:

  type The data type. Declare a global data.

8.4.2.4 #define PJ_DECL_NO_RETURN(type) PJ_NORETURN type

Parameters:

  type The return type of the function. Declare a function that will not return.

8.4.2.5 #define PJ_DEF_DATA(type) type

Parameters:

  type The data type. Define a global data.

8.4.2.6 #define PJ_END_DECL

Mark end of declaration section in a header file.

8.4.2.7 #define PJ_EXPORT_SYMBOL(x)

This macro has been deprecated. It will evaluate to nothing.

8.4.2.8 #define PJ_IDECL(type) PJ_DECL(type)

Parameters:

  type The function’s return type. Declare a function that may be expanded as inline.
### 8.4.2.9 \#define PJ_IDEF(type) PJ_DEF(type)

**Parameters:**

*type* The function’s return type. Define a function that may be expanded as inline.

### 8.4.2.10 \#define PJ_INLINE(type) PJ_INLINE_SPECIFIER type

**Parameters:**

*type* The return type of the function. Expand the function as inline.

### 8.4.2.11 \#define PJ_TODO(id) TODO___##id:

**Parameters:**

*id* Any identifier that will be printed as TODO message. PJ_TODO macro will display TODO message as warning during compilation. Example: `PJ_TODO(CLEAN_UP_ERROR)`;

### 8.4.2.12 \#define PJ_UNUSED_ARG(arg) (void)arg

**Parameters:**

*arg* The argument name. PJ_UNUSED_ARG prevents warning about unused argument in a function.

### 8.4.3 Function Documentation

#### 8.4.3.1 void pj_dump_config (void)

Dump configuration to log with verbosity equal to info(3).

#### 8.4.3.2 const char* pj_get_version (void)

Get PJLIB version string.

**Returns:**

*PJ_VERSION* constant.

### 8.4.4 Variable Documentation

#### 8.4.4.1 const char* PJ_VERSION

PJLIB version string constant.

**See also:**

*pj_get_version()*
8.5 ctype.h File Reference

8.5.1 Detailed Description

C type helper macros.

Defines

- #define pj_hex_digits "0123456789abcdef"

Functions

- int pj_isalnum (int c)
- int pj_isalpha (int c)
- int pj_isascii (int c)
- int pj_isdigit (int c)
- int pj_isspace (int c)
- int pj_islower (int c)
- int pj_isupper (int c)
- int pj_isblank (int c)
- int pj_tolower (int c)
- int pj_toupper (int c)
- int pj_isxdigit (int c)
- void pj_val_to_hex_digit (unsigned value, char *p)
- unsigned pj_hex_digit_to_val (unsigned c)
8.6 doxygen.h File Reference

8.6.1 Detailed Description

Doxygen’s mainpage.
8.7 errno.h File Reference

8.7.1 Detailed Description

PJLIB Error Codes.

Defines

- `#define PJ_ERR_MSG_SIZE 80`
- `#define PJ_RETURN_OS_ERROR(os_code)`
- `#define PJ_STATUS_FROM_OS(e)`
- `#define PJ_STATUS_TO_OS(e)`
- `#define PJ_BUILD_ERR(code, msg) { code, msg " (" #code ")" }`
- `#define PJ_EUNKNOWN`
- `#define PJ_EPENDING`
- `#define PJ_TOOMANYCONN`
- `#define PJ_EINVAL`
- `#define PJ_ENAMETOOLONG`
- `#define PJ_ENOTFOUND`
- `#define PJ_ENOMEM`
- `#define PJ_EBUG`
- `#define PJ_ETIMEDOUT`
- `#define PJ_EBUSY`
- `#define PJ_ENOTSUP`
- `#define PJ_EINVLDOP`
- `#define PJ_ECANCELED`
- `#define PJ_EEXIST`
- `#define PJ_EEOF`
- `#define PJ_EBIG`
- `#define PJ_ERRNO_START 20000`
- `#define PJ_ERRNO_SPACE_SIZE 50000`
- `#define PJ_ERRNO_START_STATUS (PJ_ERRNO_START + PJ_ERRNO_SPACE_SIZE)`
- `#define PJ_ERRNO_START_SYS (PJ_ERRNO_START_STATUS + PJ_ERRNO_SPACE_SIZE)`
- `#define PJ_ERRNO_START_USER (PJ_ERRNO_START_SYS + PJ_ERRNO_SPACE_SIZE)`

Typedefs

- `typedef pj_str_t (pj_status_t, char *, pj_size_t)`

Functions

- `pj_status_t pj_get_os_error (void)`
- `void pj_set_os_error (pj_status_t code)`
- `pj_status_t pj_genetos_error (void)`
- `void pj_set_netos_error (pj_status_t code)`
- `pj_str_t pj_strerror (pj_status_t statcode, char *, pj_size_t)`
- `pj_status_t pj_register_strerror (pj_status_t start_code, pj_status_t err_space, pjsip_error_callback f)`
- `void pj_errno_clear_handlers (void)`
8.7.2 Define Documentation

8.7.2.1 #define PJ_ERRNO_SPACE_SIZE 50000

PJ_ERRNO_SPACE_SIZE is the maximum number of errors in one of the error/status range below.

8.7.2.2 #define PJ_ERRNO_START 20000

PJ_ERRNO_START is where PJLIB specific error values start.

8.7.2.3 #define PJ_ERRNO_START_STATUS (PJ_ERRNO_START + PJ_ERRNO_SPACE_SIZE)

PJ_ERRNO_START_STATUS is where PJLIB specific status codes start. Effectively the error in this class would be 70000 - 119000.

8.7.2.4 #define PJ_ERRNO_START_SYS (PJ_ERRNO_START_STATUS + PJ_ERRNO_SPACE_SIZE)

PJ_ERRNO_START_SYS converts platform specific error codes into pj_status_t values. Effectively the error in this class would be 120000 - 169000.

8.7.2.5 #define PJ_ERRNO_START_USER (PJ_ERRNO_START_SYS + PJ_ERRNO_SPACE_SIZE)

PJ_ERRNO_START_USER are reserved for applications that use error codes along with PJLIB codes. Effectively the error in this class would be 170000 - 219000.
8.8 except.h File Reference

8.8.1 Detailed Description

Exception Handling in C.

Data Structures

- struct pj_exception_state_t

Defines

- #define PJ_USE_EXCEPTION
- #define PJ_TRY
- #define PJ_CATCH(id)
- #define PJ_CATCH_ANY
- #define PJ_END
- #define PJ_THROW(exception_id)
- #define PJ_GET_EXCEPTION()

Functions

- pj_status_t pj_exception_id_alloc (const char *name, pj_exception_id_t *id)
- pj_status_t pj_exception_id_free (pj_exception_id_t id)
- const char * pj_exception_id_name (pj_exception_id_t id)
- void pj_throw_exception_ (pj_exception_id_t id) PJ_ATTR_NORETURN
- void pj_push_exception_handler_ (struct pj_exception_state_t *rec)
- void pj_pop_exception_handler_ (void)

8.8.2 Define Documentation

8.8.2.1 #define PJ_CATCH(id)

Catch the specified exception Id.

Parameters:

- id The exception number to catch.

8.8.2.2 #define PJ_CATCH_ANY

Catch any exception number.

8.8.2.3 #define PJ_END

End of exception specification block.
8.8.2.4  \#define PJ_GET_EXCEPTION()
Get current exception.

Returns:
    Current exception code.

8.8.2.5  \#define PJ_THROW(exception_id)
Throw exception.

Parameters:
    exception_id  The exception number.

8.8.2.6  \#define PJ_TRY
Start exception specification block.

8.8.2.7  \#define PJ_USE_EXCEPTION
Declare that the function will use exception.

8.8.3  Function Documentation

8.8.3.1  void pj_pop_exception_handler_ (void)
Pop exception handler.

8.8.3.2  void pj_push_exception_handler_ (struct pj_exception_state_t * rec)
Push exception handler.

8.8.3.3  void pj_throw_exception_ (pj_exception_id_t id)
Throw exception.

Parameters:
    id  Exception Id.
8.9 file_access.h File Reference

8.9.1 Detailed Description

File manipulation and access.

Data Structures

• struct pj_file_stat

Functions

• pj_bool_t pj_file_exists (const char *filename)
• pj_off_t pj_file_size (const char *filename)
• pj_status_t pj_file_delete (const char *filename)
• pj_status_t pj_file_move (const char *oldname, const char *newname)
• pj_status_t pj_file_getstat (const char *filename, pj_file_stat *stat)
8.10  file_io.h File Reference

8.10.1  Detailed Description

Simple file I/O abstraction.

Enumerations

- enum pj_file_access { PJ_O_RDONLY = 0x1101, PJ_O_WRONLY = 0x1102, PJ_O_RDWR = 0x1103, PJ_O_APPEND = 0x1108 }
- enum pj_file_seek_type { PJ_SEEK_SET = 0x1201, PJ_SEEK_CUR = 0x1202, PJ_SEEK_END = 0x1203 }

Functions

- pj_status_t pj_file_open (pj_pool_t *pool, const char *pathname, unsigned flags, pj_oshandle_t *fd)
- pj_status_t pj_file_close (pj_oshandle_t fd)
- pj_status_t pj_file_write (pj_oshandle_t fd, const void *data, pj_ssize_t *size)
- pj_status_t pj_file_read (pj_oshandle_t fd, void *data, pj_ssize_t *size)
- pj_status_t pj_file_setpos (pj_oshandle_t fd, pj_off_t offset, enum pj_file_seek_type whence)
- pj_status_t pj_file_getpos (pj_oshandle_t fd, pj_off_t *pos)
- pj_status_t pj_file_flush (pj_oshandle_t fd)
8.11 guid.h File Reference

8.11.1 Detailed Description

GUID Globally Unique Identifier.

Defines

- #define PJ_GUID_MAX_LENGTH 36

Functions

- PJ_DECL_DATA (const unsigned) PJ_GUID_STRING_LENGTH
- unsigned pj_GUID_STRING_LENGTH()
- pj_str_t * pj_generate_unique_string (pj_str_t *str)
- void pj_create_unique_string (pj_pool_t *pool, pj_str_t *str)
8.12 hash.h File Reference

8.12.1 Detailed Description

Hash Table.

Defines

- \#define PJ_HASH_KEY_STRING ((unsigned)-1)
- \#define PJ_HASH_ENTRY_BUF_SIZE (3*sizeof(void*) + 2*sizeof(pj_uint32_t))

Typedefs

- typedef void *pj_hash_entry_buf [((3*sizeof(void*)+2*sizeof(pj_uint32_t))+sizeof(void*))-1]/sizeof(void*)]

Functions

- pj_uint32_t pj_hash_calc (pj_uint32_t hval, const void *key, unsigned keylen)
- pj_uint32_t pj_hash_calc_tolower (pj_uint32_t hval, char *result, const pj_str_t *key)
- pj_hash_table_t *pj_hash_create (pj_pool_t *pool, unsigned size)
- void *pj_hash_get (pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t *hval)
- void pj_hash_set (pj_pool_t *pool, pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t *hval, void *value)
- void pj_hash_set_np (pj_hash_table_t *ht, const void *key, unsigned keylen, pj_uint32_t *hval, pj_hash_entry_buf entry_buf, void *value)
- unsigned pj_hash_count (pj_hash_table_t *ht)
- pj_hash_iterator_t *pj_hash_first (pj_hash_table_t *ht, pj_hash_iterator_t *it)
- pj_hash_iterator_t *pj_hash_next (pj_hash_table_t *ht, pj_hash_iterator_t *it)
- void *pj_hash_this (pj_hash_table_t *ht, pj_hash_iterator_t *it)
8.13 ioqueue.h File Reference

8.13.1 Detailed Description
I/O Dispatching Mechanism.

Data Structures

- struct pj_ioqueue_op_key_t
- struct pj_ioqueue_callback

Defines

- #define PJ_IOQUEUE_MAX_EVENTS_IN_SINGLE_POLL (16)
- #define PJ_IOQUEUE_ALWAYS_ASYNC ((pj_uint32_t)1 << ((pj_uint32_t)31))

Enumerations

- enum pj_ioqueue_operation_e {
  PJ_IOQUEUE_OP_NONE = 0, PJ_IOQUEUE_OP_READ = 1, PJ_IOQUEUE_OP_RECV = 2,
  PJ_IOQUEUE_OP_RECV_FROM = 4, PJ_IOQUEUE_OP_WRITE = 8, PJ_IOQUEUE_OP_SEND = 16,
  PJ_IOQUEUE_OP_SEND_TO = 32, PJ_IOQUEUE_OP_ACCEPT = 64,
  PJ_IOQUEUE_OP_CONNECT = 128 }

Functions

- const char * pj_ioqueue_name (void)
- pj_status_t pj_ioqueue_create (pj_pool_t *pool, pj_size_t max_fd, pj_ioqueue_t **ioqueue)
- pj_status_t pj_ioqueue_destroy (pj_ioqueue_t *ioque)
- pj_status_t pj_ioqueue_set_lock (pj_ioqueue_t *iioque, pj_lock_t *lock, pj_bool_t auto_delete)
- pj_status_t pj_ioqueue_register_sock (pj_pool_t *pool, pj_ioqueue_t *iioque, pj_sock_t sock, void *user_data, const pj_ioqueue_callback *cb, pj_ioqueue_key_t **key)
- pj_status_t pj_ioqueue_unregister (pj_ioqueue_key_t *key)
- void * pj_ioqueue_get_user_data (pj_ioqueue_key_t *key)
- void * pj_ioqueue_set_user_data (pj_ioqueue_key_t *key, void *user_data, void **old_data)
- void pj_ioqueue_op_key_init (pj_ioqueue_op_key_t *op_key, pj_size_t size)
- pj_bool_t pj_ioqueue_op_key_pending (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key)
- pj_status_t pj_ioqueue_post_completion (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_ssize_t bytes_status)
- pj_status_t pj_ioqueue_accept (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, pj_sock_t *new_sock, pj_sockaddr_t *local, pj_sockaddr_t *remote, int *addrlen)
- pj_status_t pj_ioqueue_connect (pj_ioqueue_key_t *key, const pj_sockaddr_t *addr, int addrlen)
- int pj_ioqueue_poll (pj_ioqueue_t *ioque, const pj_time_val *timeout)
- pj_status_t pj_ioqueue_recv (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, void *buffer, pj_ssize_t *length, pj_uint32_t flags)
- pj_status_t pj_ioqueue_recvfrom (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, void *buffer, pj_ssize_t *length, pj_uint32_t flags, pj_sockaddr_t *addr, int *addrlen)
• `pj_status_t pj_ioqueue_send (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, const void *data, pj_ssize_t *length, pj_uint32_t flags)`

• `pj_status_t pj_ioqueue_sendto (pj_ioqueue_key_t *key, pj_ioqueue_op_key_t *op_key, const void *data, pj_ssize_t *length, pj_uint32_t flags, const pj_sockaddr_t *addr, int addrlen)`
8.14 ip_helper.h File Reference

8.14.1 Detailed Description

IP helper API.

Data Structures

- union pj_ip_route_entry

Functions

- pj_status_t pj_enum_ip_interface (unsigned *count, pj_in_addr ifs[])
- pj_status_t pj_enum_ip_route (unsigned *count, pj_ip_route_entry routes[])
8.15  list.h File Reference

8.15.1  Detailed Description

Linked List data structure.

Data Structures

- struct pj_list

Defines

- #define PJ_DECL_LIST_MEMBER(type)

Functions

- void pj_list_init (pj_list_type *node)
- int pj_list_empty (const pj_list_type *node)
- void pj_list_insert_before (pj_list_type *pos, pj_list_type *node)
- void pj_list_push_back (pj_list_type *list, pj_list_type *node)
- void pj_list_insert_nodes_before (pj_list_type *lst, pj_list_type *nodes)
- void pj_list_insert_after (pj_list_type *pos, pj_list_type *node)
- void pj_list_push_front (pj_list_type *list, pj_list_type *node)
- void pj_list_insert_nodes_after (pj_list_type *lst, pj_list_type *nodes)
- void pj_list_merge_first (pj_list_type *list1, pj_list_type *list2)
- void pj_list_merge_last (pj_list_type *list1, pj_list_type *list2)
- void pj_list_erase (pj_list_type *node)
- pj_list_type * pj_list_find_node (pj_list_type *list, pj_list_type *node)
- pj_list_type * pj_list_search (pj_list_type *list, void *value, int(*comp)(void *value, const pj_list_type *node))
- pj_size_t pj_list_size (pj_list_type *list)
8.16 lock.h File Reference

8.16.1 Detailed Description

Higher abstraction for locking objects.

Functions

- `pj_status_t pj_lock_create_simple_mutex (pj_pool_t *pool, const char *name, pj_lock_t **lock)`
- `pj_status_t pj_lock_create_recursive_mutex (pj_pool_t *pool, const char *name, pj_lock_t **lock)`
- `pj_status_t pj_lock_create_null_mutex (pj_pool_t *pool, const char *name, pj_lock_t **lock)`
- `pj_status_t pj_lock_create_semaphore (pj_pool_t *pool, const char *name, unsigned initial, unsigned max, pj_lock_t **lock)`
- `pj_status_t pj_lock_acquire (pj_lock_t *lock)`
- `pj_status_t pj_lock_tryacquire (pj_lock_t *lock)`
- `pj_status_t pj_lock_release (pj_lock_t *lock)`
- `pj_status_t pj_lock_destroy (pj_lock_t *lock)`
8.17 log.h File Reference

8.17.1 Detailed Description

Logging Utility.

Defines

- #define PJ_LOG(level, arg)
- #define pj_log_wrapper_1(arg) pj_log_1 arg
- #define pj_log_wrapper_2(arg) pj_log_2 arg
- #define pj_log_wrapper_3(arg) pj_log_3 arg
- #define pj_log_wrapper_4(arg) pj_log_4 arg
- #define pj_log_wrapper_5(arg)
- #define pj_log_wrapper_6(arg)

typedefs

- typedef void pj_log_func (int level, const char ∗data, int len)

Enumerations

- enum pj_log_decoration {
    PJ_LOG_HAS_DAY_NAME = 1, PJ_LOG_HAS_YEAR = 2, PJ_LOG_HAS_MONTH = 4, PJ_LOG_HAS_DAY_OF_MON = 8,
    PJ_LOG_HAS_TIME = 16, PJ_LOG_HAS_MICRO_SEC = 32, PJ_LOG_HAS_SENDER = 64, PJ_LOG_HAS_NEWLINE = 128,
    PJ_LOG_HAS_CR = 256 }

Functions

- void pj_log_write (int level, const char ∗buffer, int len)
- void pj_log (const char ∗sender, int level, const char ∗format, va_list marker)
- void pj_log_set_log_func (pj_log_func ∗func)
- pj_log_func ∗pj_log_get_log_func (void)
- void pj_log_set_level (int level)
- int pj_log_get_level (void)
- void pj_log_set_decor (unsigned decor)
- unsigned pj_log_get_decor (void)
- void pj_log_1 (const char ∗src, const char ∗format,...)
- void pj_log_2 (const char ∗src, const char ∗format,...)
- void pj_log_3 (const char ∗src, const char ∗format,...)
- void pj_log_4 (const char ∗src, const char ∗format,...)
8.17.2 Define Documentation

8.17.2.1 #define pj_log_wrapper_1(arg) pj_log_1 arg
Internal function to write log with verbosity 1. Will evaluate to empty expression if PJ_LOG_MAX-_LEVEL is below 1.
Parameters:
   arg  Log expression.

8.17.2.2 #define pj_log_wrapper_2(arg) pj_log_2 arg
Internal function to write log with verbosity 2. Will evaluate to empty expression if PJ_LOG_MAX-_LEVEL is below 2.
Parameters:
   arg  Log expression.

8.17.2.3 #define pj_log_wrapper_3(arg) pj_log_3 arg
Internal function to write log with verbosity 3. Will evaluate to empty expression if PJ_LOG_MAX-_LEVEL is below 3.
Parameters:
   arg  Log expression.

8.17.2.4 #define pj_log_wrapper_4(arg) pj_log_4 arg
Internal function to write log with verbosity 4. Will evaluate to empty expression if PJ_LOG_MAX-_LEVEL is below 4.
Parameters:
   arg  Log expression.

8.17.2.5 #define pj_log_wrapper_5(arg)
Internal function to write log with verbosity 5. Will evaluate to empty expression if PJ_LOG_MAX-_LEVEL is below 5.
Parameters:
   arg  Log expression.

8.17.2.6 #define pj_log_wrapper_6(arg)
Internal function to write log with verbosity 6. Will evaluate to empty expression if PJ_LOG_MAX-_LEVEL is below 6.
Parameters:

\textit{arg} Log expression.

8.17.3 Function Documentation

8.17.3.1 \texttt{void pj_log\_1 (const char * src, const char * format, ...)}

Internal function.

8.17.3.2 \texttt{void pj_log\_2 (const char * src, const char * format, ...)}

Internal function.

8.17.3.3 \texttt{void pj_log\_3 (const char * src, const char * format, ...)}

Internal function.

8.17.3.4 \texttt{void pj_log\_4 (const char * src, const char * format, ...)}

Internal function.
8.18 os.h File Reference

8.18.1 Detailed Description

OS dependent functions.

Defines

- 
- 
- 
- 
- 

Typedefs

- typedef long pj_thread_desc [64]
- typedef pj_rwlock_t pj_rwlock_t

Enumerations

- enum pj_thread_create_flags { PJ_THREAD_SUSPENDED = 1 }
- enum pj_mutex_type_e { PJ_MUTEX_DEFAULT, PJ_MUTEX_SIMPLE, PJ_MUTEX_RECURSE }

Functions

- typedef int (PJ_THREAD_FUNC pj_thread_proc)(void *s)
- pj_uint32_t pj_getpid (void)
- pj_status_t pj_thread_create (pj_pool_t *pool, const char *thread_name, pj_thread_proc *proc, void *arg, pj_size_t stack_size, unsigned flags, pj_thread_t **thread)
- pj_status_t pj_thread_register (const char *thread_name, pj_thread_desc desc, pj_thread_t **thread)
- pj_bool_t pj_thread_is_registered (void)
- const char *pj_thread_get_name (pj_thread_t *thread)
- pj_status_t pj_thread_resume (pj_thread_t *thread)
- pj_thread_t *pj_thread_this (void)
- pj_status_t pj_thread_join (pj_thread_t *thread)
- pj_status_t pj_thread_destroy (pj_thread_t *thread)
- pj_status_t pj_thread_sleep (unsigned msec)
- pj_status_t pj_symbianos_poll (int priority, int ms_timeout)
- pj_status_t pj_thread_local_alloc (long *index)
- void pj_thread_local_free (long index)
- pj_status_t pj_thread_local_set (long index, void *value)
- void *pj_thread_local_get (long index)
- pj_status_t pj_atomic_create (pj_pool_t *pool, pj_atomic_value_t initial, pj_atomic_t **atomic)
- pj_status_t pj_atomic_destroy (pj_atomic_t *atomic_var)
- void pj_atomic_set (pj_atomic_t *atomic_var, pj_atomic_value_t value)
- pj_atomic_value_t pj_atomic_get (pj_atomic_t *atomic_var)
• void pj_atomic_inc (pj_atomic_t *atomic_var)
• pj_atomic_value_t pj_atomic_inc_and_get (pj_atomic_t *atomic_var)
• void pj_atomic_dec (pj_atomic_t *atomic_var)
• pj_atomic_value_t pj_atomic_dec_and_get (pj_atomic_t *atomic_var)
• void pj_atomic_add (pj_atomic_t *atomic_var, pj_atomic_value_t value)
• pj_atomic_value_t pj_atomic_add_and_get (pj_atomic_t *atomic_var, pj_atomic_value_t value)
• pj_status_t pj_mutex_create (pj_pool_t *pool, const char *name, int type, pj_mutex_t **mutex)
• pj_status_t pj_mutex_create_simple (pj_pool_t *pool, const char *name, pj_mutex_t **mutex)
• pj_status_t pj_mutex_create_recursive (pj_pool_t *pool, const char *name, pj_mutex_t **mutex)
• pj_status_t pj_mutex_lock (pj_mutex_t *mutex)
• pj_status_t pj_mutex_unlock (pj_mutex_t *mutex)
• pj_status_t pj_mutex_trylock (pj_mutex_t *mutex)
• pj_status_t pj_mutex_destroy (pj_mutex_t *mutex)
• pj_status_t pj_rwlockcreate (pj_pool_t *pool, const char *name, pj_rwlock_t **mutex)
• pj_status_t pj_rwlock_lock_read (pj_rwlock_t *mutex)
• pj_status_t pj_rwlock_lock_write (pj_rwlock_t *mutex)
• pj_status_t pj_rwlock_unlock_read (pj_rwlock_t *mutex)
• pj_status_t pj_rwlock_unlock_write (pj_rwlock_t *mutex)
• pj_status_t pj_rwlock_destroy (pj_rwlock_t *mutex)
• void pj_enter_critical_section (void)
• void pj_leave_critical_section (void)
• pj_status_t pj_sem_create (pj_pool_t *pool, const char *name, unsigned initial, unsigned max, pj_sem_t **sem)
• pj_status_t pj_sem_wait (pj_sem_t *sem)
• pj_status_t pj_sem_trywait (pj_sem_t *sem)
• pj_status_t pj_sem_post (pj_sem_t *sem)
• pj_status_t pj_sem_destroy (pj_sem_t *sem)
• pj_status_t pj_event_create (pj_pool_t *pool, const char *name, pj_bool_t manual_reset, pj_bool_t initial, pj_event_t **event)
• pj_status_t pj_event_wait (pj_event_t *event)
• pj_status_t pj_event_trywait (pj_event_t *event)
• pj_status_t pj_event_set (pj_event_t *event)
• pj_status_t pj_event_pulse (pj_event_t *event)
• pj_status_t pj_event_reset (pj_event_t *event)
• pj_status_t pj_event_destroy (pj_event_t *event)
• pj_status_t pj_event_set (pj_event_t *event)
• pj_status_t pj_time_decode (const pj_time_val *tv, pj_parsed_time *pt)
• pj_status_t pj_time_encode (const pj_parsed_time *pt, pj_time_val *tv)
• pj_status_t pj_time_local_to_gmt (pj_time_val *tv)
• pj_status_t pj_time_gmt_to_local (pj_time_val *tv)
• pj_status_t pj_get_timestamp (pj_timestamp *ts)
• pj_status_t pj_get_timestamp_freq (pj_timestamp *freq)
• void pj_set_timestamp32 (pj_timestamp *t, pj_uint32_t hi, pj_uint32_t lo)
• int pj_cmp_timestamp (const pj_timestamp *t1, const pj_timestamp *t2)
• void pj_add_timestamp (pj_timestamp *t1, const pj_timestamp *t2)
• void pj_add_timestamp32 (pj_timestamp *t1, pj_uint32_t t2)
• void pj_sub_timestamp (pj_timestamp *t1, const pj_timestamp *t2)
• void pj_sub_timestamp32 (pj_timestamp *t1, pj_uint32_t t2)
• pj_uint32_t pj_timestamp_diff32 (const pj_timestamp *t1, const pj_timestamp *t2)
• pj_time_val pj_elapsed_time (const pj_timestamp *start, const pj_timestamp *stop)
8.18 os.h File Reference

- `pj_uint32_t pj_elapsed_msec (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_usec (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_nanosec (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_uint32_t pj_elapsed_cycle (const pj_timestamp *start, const pj_timestamp *stop)`
- `pj_status_t pj_thread_init (void)`

8.18.2 Function Documentation

8.18.2.1 `pj_status_t pj_thread_init (void)`

Internal PJLIB function to initialize the threading subsystem.

**Returns:**

PJ_SUCCESS or the appropriate error code.
8.19 pool.h File Reference

8.19.1 Detailed Description

Memory Pool.

Data Structures

- struct pj_pool_block
- struct pj_pool_t
- struct pj_pool_factory_policy
- struct pj_pool_factory
- struct pj_caching_pool

Defines

- #define PJ_POOL_SIZE (sizeof(struct pj_pool_t))
- #define PJ_POOL_ALIGNMENT 4
- #define PJ_POOL_ALLOC_T(pool, type) ((type *)pj_pool_alloc(pool, sizeof(type)))
- #define PJ_POOL_ZALLOC_T(pool, type) ((type *)pj_pool_zalloc(pool, sizeof(type)))
- #define PJ_CACHING_POOL_ARRAY_SIZE 16

Typedefs

- typedef void pj_pool_callback (pj_pool_t *pool, pj_size_t size)

Functions

- pj_pool_t * pj_pool_create (pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)
- void pj_pool_release (pj_pool_t *pool)
- const char * pj_pool_getobjname (const pj_pool_t *pool)
- void pj_pool_reset (pj_pool_t *pool)
- pj_size_t pj_pool_get_capacity (pj_pool_t *pool)
- pj_size_t pj_pool_get_used_size (pj_pool_t *pool)
- void * pj_pool_alloc (pj_pool_t *pool, pj_size_t size)
- void * pj_pool_calloc (pj_pool_t *pool, pj_size_t count, pj_size_t elem)
- void * pj_pool_zalloc (pj_pool_t *pool, pj_size_t size)
- void * pj_pool_alloc_from_block (pj_pool_block *block, pj_size_t size)
- void * pj_pool_allocate_find (pj_pool_t *pool, unsigned size)
- PJ_DECL_DATA (int) PJ_NO_MEMORY_EXCEPTION
- int pj_NO_MEMORY_EXCEPTION ()
- PJ_DECL_DATA (pj_pool_factory_policy) pj_pool_factory_default_policy
- const pj_pool_factory_policy * pj_pool_factory_get_default_policy (void)
- pj_pool_t * pj_pool_create_int (pj_pool_factory *factory, const char *name, pj_size_t initial_size, pj_size_t increment_size, pj_pool_callback *callback)
- void pj_pool_init_int (pj_pool_t *pool, const char *name, pj_size_t increment_size, pj_pool_callback *callback)
• void `pj_pool_destroy_int (pj_pool_t *pool)`
• void `pj_pool_factory_dump (pj_pool_factory *pf, pj_bool_t detail)`
• void `pj_caching_pool_init (pj_caching_pool *ch_pool, const pj_pool_factory_policy *policy, pj_size_t max_capacity)`
• void `pj_caching_pool_destroy (pj_caching_pool *ch_pool)`
8.20 rand.h File Reference

8.20.1 Detailed Description

Random Number Generator.

Functions

• void pj_srand (unsigned int seed)
• int pj_rand (void)
8.21 rbtree.h File Reference

8.21.1 Detailed Description

Red/Black Tree.

Data Structures

- struct pj_rbtree_node
- struct pj_rbtree

Defines

- #define PJ_RBTREE_NODE_SIZE (sizeof(pj_rbtree_node))
- #define PJ_RBTREE_SIZE (sizeof(pj_rbtree))

Typedefs

- typedef int pj_rbtree_comp (const void *key1, const void *key2)

Enumerations

- enum pj_rbcolor_t { PJ_RBCOLOR_BLACK, PJ_RBCOLOR_RED }

Functions

- void pj_rbtree_init (pj_rbtree *tree, pj_rbtree_comp *comp)
- pj_rbtree_node * pj_rbtree_first (pj_rbtree *tree)
- pj_rbtree_node * pj_rbtree_last (pj_rbtree *tree)
- pj_rbtree_node * pj_rbtree_next (pj_rbtree *tree, pj_rbtree_node *node)
- pj_rbtree_node * pj_rbtree_prev (pj_rbtree *tree, pj_rbtree_node *node)
- int pj_rbtree_insert (pj_rbtree *tree, pj_rbtree_node *node)
- pj_rbtree_node * pj_rbtree_find (pj_rbtree *tree, const void *key)
- pj_rbtree_node * pj_rbtree_erase (pj_rbtree *tree, pj_rbtree_node *node)
- unsigned pj_rbtree_max_height (pj_rbtree *tree, pj_rbtree_node *node)
- unsigned pj_rbtree_min_height (pj_rbtree *tree, pj_rbtree_node *node)
8.22 sock.h File Reference

8.22.1 Detailed Description

Socket Abstraction.

Data Structures

• struct pj_in_addr
• struct pj_sockaddr_in
• struct pj_in6_addr
• struct pj_sockaddr_in6
• struct pj_addr_hdr
• union pj_sockaddr

Defines

• #define PJ_AF_LOCAL PJ_AF_UNIX;
• #define PJ_INET_ANY ((pj_uint32_t)0)
• #define PJ_INET_NONE ((pj_uint32_t)0xffffffff)
• #define PJ_INET_BROADCAST ((pj_uint32_t)0xffffffff)
• #define PJ_SOMAXCONN 5
• #define PJ_INVALID_SOCKET (-1)
• #define s6_addr in6_u.u6_addr8
• #define s6_addr16 in6_u.u6_addr16
• #define s6_addr32 in6_u.u6_addr32
• #define PJ_INET6ADDR_ANY_INIT [ { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 } } ]
• #define PJ_INET6ADDR_LOOPBACK_INIT [ { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1 } } ]

Enumerations

• enum pj_socket_sd_type {
PJ_SD_RECEIVE = 0, PJ_SHUT_RD = 0, PJ_SD_SEND = 1, PJ_SHUT_WR = 1,
PJ_SD_BOTH = 2, PJ_SHUT_RDWR = 2 }
• `pj_uint16_t pj_SOL_TCP (void)`
• `pj_uint16_t pj_SOL_UDP (void)`
• `pj_uint16_t pj_SOL_IPV6 (void)`
• `int pj_IP_TOS (void)`
• `int pj_IPTOS_LOWDELAY (void)`
• `int pj_IPTOS_THROUGHPUT (void)`
• `int pj_IPTOS_RELIABILITY (void)`
• `int pj_IPTOS_MINCOST (void)`
• `pj_uint16_t pj_SO_TYPE (void)`
• `pj_uint16_t pj_SO_RCVBUF (void)`
• `pj_uint16_t pj_SO_SNDBUF (void)`
• `int pj_MSG_OOB (void)`
• `int pj_MSG_PEEK (void)`
• `int pj_MSG_DONTROUTE (void)`
• `pj_uint16_t pj_ntohs (pj_uint16_t netshort)`
• `pj_uint16_t pj_htons (pj_uint16_t hostshort)`
• `pj_uint32_t pj_ntohl (pj_uint32_t netlong)`
• `pj_uint32_t pj_htonl (pj_uint32_t hostlong)`
• `char * pj_inet_ntoa (pj_in_addr inaddr)`
• `int pj_inet_aton (const pj_str_t *cp, struct pj_in_addr *inp)`
• `pj_in_addr pj_inet_addr (const pj_str_t *cp)`
• `pj_in_addr pj_inet_addr2 (const char *cp)`
• `void pj_sockaddr_in_get_port (const pj_sockaddr_in *addr)`
• `void pj_sockaddr_in_set_port (pj_sockaddr_in *addr, pj_uint16_t hostport)`
• `pj_in_addr pj_sockaddr_in_get_addr (const pj_sockaddr_in *addr)`
• `void pj_sockaddr_in_set_addr (pj_sockaddr_in *addr, pj_uint32_t hostaddr)`
• `pj_status_t pj_sockaddr_in_set_str_addr (pj_sockaddr_in *addr, const pj_str_t *cp)`
• `pj_status_t pj_sockaddr_in_init (pj_sockaddr_in *addr, const pj_str_t *cp, pj_uint16_t port)`
• `pj_status_t pj_sock_socket (int family, int type, int protocol, pj_sock_t *sock)`
• `pj_status_t pj_sock_close (pj_sock_t sockfd)`
• `pj_status_t pj_sock_bind (pj_sock_t sockfd, const pj_sockaddr_t *my_addr, int addrlen)`
• `pj_status_t pj_sock_listen (pj_sock_t sockfd, int backlog)`
• `pj_status_t pj_sock_accept (pj_sock_t serverfd, pj_sock_t *newsock, pj_sockaddr_t *addr, int *addrlen)`
• `pj_status_t pj_sock_connect (pj_sock_t sockfd, const pj_sockaddr_t *serv_addr, int addrlen)`
• `pj_status_t pj_sock_getpeername (pj_sock_t sockfd, pj_sockaddr_t *addr, int *namelen)`
• `pj_status_t pj_sock_getsockname (pj_sock_t sockfd, pj_sockaddr_t *addr, int *namelen)`
• `pj_status_t pj_sock_getsockopt (pj_sock_t sockfd, pj_uint16_t level, pj_uint16_t optname, void *optval, int *optlen)`
• `pj_status_t pj_sock_setsockopt (pj_sock_t sockfd, pj_uint16_t level, pj_uint16_t optname, const void *optval, int optlen)`
• `pj_status_t pj_sock_recv (pj_sock_t sockfd, void *buf, pj_ssize_t *len, unsigned flags)`
• `pj_status_t pj_sock_recvfrom (pj_sock_t sockfd, void *buf, pj_ssize_t *len, unsigned flags, pj_sockaddr_t *from, int *fromlen)`
• `pj_status_t pj_sock_send (pj_sock_t sockfd, const void *buf, pj_ssize_t *len, unsigned flags)`
• `pj_status_t pj_sock_sendto (pj_sock_t sockfd, const void *buf, pj_ssize_t *len, unsigned flags, const pj_sockaddr_t *to, int tolen)`
• `pj_status_t pj_sock_shutdown (pj_sock_t sockfd, int how)`
Variables

- const pj_uint16_t PJ_AF_UNIX
- const pj_uint16_t PJ_AF_INET
- const pj_uint16_t PJ_AF_INET6
- const pj_uint16_t PJ_AF_PACKET
- const pj_uint16_t PJ_AF_IRDA
- const pj_uint16_t PJ_SOCK_STREAM
- const pj_uint16_t PJ_SOCK_DGRAM
- const pj_uint16_t PJ_SOCK_RAW
- const pj_uint16_t PJ_SOCK_RDM
- const pj_uint16_t PJ_SOL_SOCKET
- const pj_uint16_t PJ_SOL_IP
- const pj_uint16_t PJ_SOL_TCP
- const pj_uint16_t PJ_SOL_UDP
- const pj_uint16_t PJ_SOL_IPV6
- const pj_uint16_t PJ_IP_TOS
- const pj_uint16_t PJ_IPTOS_LOWDELAY
- const pj_uint16_t PJ_IPTOS_THROUGHPUT
- const pj_uint16_t PJ_IPTOS_RELIABILITY
- const pj_uint16_t PJ_IPTOS_MINCOST
- const pj_uint16_t PJ_SO_TYPE
- const pj_uint16_t PJ_SO_RCVBUF
- const pj_uint16_t PJ_SO_SNDBUF
- const int PJ_MSG_OOB
- const int PJ_MSG_PEEK
- const int PJ_MSG_DONTROUTE

8.22.2 Define Documentation

8.22.2.1 #define s6_addr in6_u.u6_addr8

Shortcut to access in6_u.u6_addr8.

8.22.2.2 #define s6_addr16 in6_u.u6_addr16

Shortcut to access in6_u.u6_addr16.

8.22.2.3 #define s6_addr32 in6_u.u6_addr32

Shortcut to access in6_u.u6_addr32.
8.23 sock_select.h File Reference

8.23.1 Detailed Description

Socket select().

Data Structures

- struct pj_fd_set_t

Functions

- void PJ_FD_ZERO (pj_fd_set_t *fdsetp)
- pj_size_t PJ_FD_COUNT (const pj_fd_set_t *fdsetp)
- void PJ_FD_SET (pj_sock_t fd, pj_fd_set_t *fdsetp)
- void PJ_FD_CLR (pj_sock_t fd, pj_fd_set_t *fdsetp)
- pj_bool_t PJ_FD_ISSET (pj_sock_t fd, const pj_fd_set_t *fdsetp)
- int pj_sock_select (int n, pj_fd_set_t *readfds, pj_fd_set_t *writefds, pj_fd_set_t *exceptfds, const pj_time_val *timeout)
8.24 string.h File Reference

8.24.1 Detailed Description

PJLIB String Operations.

Defines

• #define strnicmp_alnum pj_ansi_strnicmp
• #define pj_stricmp_alnum pj_stricmp

Functions

• pj_str_t pj_str (char *str)
• const pj_str_t * pj_cstr (pj_str_t *str, const char *s)
• pj_str_t * pj_strset (pj_str_t *str, char *ptr, pj_size_t length)
• pj_str_t * pj_strset2 (pj_str_t *str, char *src)
• pj_str_t * pj_strset3 (pj_str_t *str, char *begin, char *end)
• pj_str_t * pj_strassign (pj_str_t *dst, const pj_str_t *src)
• pj_str_t * pj_strcpy (pj_str_t *dst, const pj_str_t *src)
• pj_str_t * pj_strcpy2 (pj_str_t *dst, const char *src)
• pj_str_t * pj_strncpy (pj_str_t *dst, const pj_str_t *src, pj_ssize_t max)
• pj_str_t * pj_strncpy_with_null (pj_str_t *dst, const pj_str_t *src, pj_ssize_t max)
• pj_str_t * pj_strdup (pj_pool_t *pool, pj_str_t *dst, const pj_str_t *src)
• pj_str_t * pj_strdup_with_null (pj_pool_t *pool, pj_str_t *dst, const pj_str_t *src)
• pj_str_t * pj_strdup2 (pj_pool_t *pool, pj_str_t *dst, const char *src)
• pj_str_t * pj_strdup2_with_null (pj_pool_t *pool, pj_str_t *dst, const char *src)
• pj_str_t *pj_strdup3 (pj_pool_t *pool, const char *src)
• pj_size_t pj_strlen (const pj_str_t *str)
• const char * pj_strbuf (const pj_str_t *str)
• int pj_strcmp (const pj_str_t *str1, const pj_str_t *str2)
• int pj_strcmp2 (const pj_str_t *str1, const char *str2)
• int pj_strncmp (const pj_str_t *str1, const pj_str_t *str2, pj_size_t len)
• int pj_strncmp2 (const pj_str_t *str1, const char *str2, pj_size_t len)
• int pj_stricmp (const pj_str_t *str1, const pj_str_t *str2)
• int pj_stricmp2 (const pj_str_t *str1, const char *str2)
• int pj_strnicmp (const pj_str_t *str1, const pj_str_t *str2, pj_size_t len)
• int pj_strnicmp2 (const pj_str_t *str1, const char *str2)
• void pj_strcat (pj_str_t *dst, const pj_str_t *src)
• void pj_strcat2 (pj_str_t *dst, const char *src)
• char * pj_strchr (const pj_str_t *str, int chr)
• pj_str_t * pj_strltrim (pj_str_t *str)
• pj_str_t * pj_strrtrim (pj_str_t *str)
• pj_str_t * pj_strtrim (pj_str_t *str)
• char * pj_create_random_string (char *str, pj_size_t length)
• unsigned long pj_strtoul (const pj_str_t *str)
• unsigned long pj_strtoul2 (const pj_str_t *str, char *endptr, unsigned base)
• int pj_utoa (unsigned long val, char *buf)
• int pj_utoa_pad (unsigned long val, char *buf, int min_dig, int pad)
• void \texttt{pj_bzero} (void *dst, \texttt{pj\_size\_t} size)
• void * \texttt{pj_memset} (void *dst, int c, \texttt{pj\_size\_t} size)
• void * \texttt{pj_memcpy} (void *dst, const void *src, \texttt{pj\_size\_t} size)
• void * \texttt{pj_memmove} (void *dst, const void *src, \texttt{pj\_size\_t} size)
• int \texttt{pj_memcmp} (const void *buf1, const void *buf2, \texttt{pj\_size\_t} size)
• void * \texttt{pj_memchr} (const void *buf, int c, \texttt{pj\_size\_t} size)
8.25 timer.h File Reference

8.25.1 Detailed Description

Timer Heap.

Data Structures

- struct pj_timer_entry

Typedefs

- typedef int pj_timer_id_t
- typedef void pj_timer_heap_callback (pj_timer_heap_t *timer_heap, struct pj_timer_entry *entry)

Functions

- pj_size_t pj_timer_heap_mem_size (pj_size_t count)
- pj_status_t pj_timer_heap_create (pj_pool_t *pool, pj_size_t count, pj_timer_heap_t **ht)
- void pj_timer_heap_destroy (pj_timer_heap_t *ht)
- void pj_timer_heap_set_lock (pj_timer_heap_t *ht, pj_lock_t *lock, pj_bool_t auto_del)
- unsigned pj_timer_heap_set_max_timed_out_per_poll (pj_timer_heap_t *ht, unsigned count)
- pj_timer_entry * pj_timer_entry_init (pj_timer_entry *entry, int id, void *user_data, pj_timer_heap_callback *cb)
- pj_status_t pj_timer_heap_schedule (pj_timer_heap_t *ht, pj_timer_entry *entry, const pj_time_val *delay)
- int pj_timer_heap_cancel (pj_timer_heap_t *ht, pj_timer_entry *entry)
- pj_size_t pj_timer_heap_count (pj_timer_heap_t *ht)
- pj_status_t pj_timer_heap_earliest_time (pj_timer_heap_t *ht, pj_time_val *timeval)
- unsigned pj_timer_heap_poll (pj_timer_heap_t *ht, pj_time_val *next_delay)
8.26 types.h File Reference

8.26.1 Detailed Description

Declaration of basic types and utility.

Data Structures

- struct pj_str_t
- union pj_timestamp
- struct pj_hash_iterator_t
- struct pj_time_val
- struct pj_parsed_time

Defines

- #define PJ_T(literal_str) literal_str
- #define PJ_SUCCESS 0
- #define PJ_TRUE 1
- #define PJ_FALSE 0
- #define PJ_ARRAY_SIZE(a) (sizeof(a)/sizeof(a[0]))
- #define PJ_MAXINT32 0x7FFFFFFFL
- #define PJ_MAX_OBJ_NAME 32
- #define PJ_TIME_VAL_MSEC(t)
- #define PJ_TIME_VAL_EQ(t1, t2)
- #define PJ_TIME_VAL_GT(t1, t2)
- #define PJ_TIME_VAL_GTE(t1, t2)
- #define PJ_TIME_VAL_LT(t1, t2)
- #define PJ_TIME_VAL_LTE(t1, t2)
- #define PJ_TIME_VAL_ADD(t1, t2)
- #define PJ_TIME_VAL_SUB(t1, t2)

typedefs

- typedef unsigned int pj_uint32_t
- typedef short pj_int16_t
- typedef unsigned short pj_uint16_t
- typedef signed char pj_int8_t
- typedef unsigned char pj_uint8_t
- typedef size_t pj_size_t
- typedef long pj_ssize_t
- typedef int pj_status_t
- typedef int pj_bool_t
- typedef char pj_char_t
- typedef pj_ssize_t pj_off_t
- typedef void pj_list_type
- typedef pj_list pj_list
- typedef pj_hash_table_t pj_hash_table_t
- typedef pj_hash_entry pj_hash_entry
• typedef pj_pool_factory pj_pool_factory
• typedef pj_pool_t pj_pool_t
• typedef pj_caching_pool pj_caching_pool
• typedef pj_str_t pj_str_t
• typedef pj_ioqueue_t pj_ioqueue_t
• typedef pj_ioqueue_key_t pj_ioqueue_key_t
• typedef pj_timer_heap_t pj_timer_heap_t
• typedef pj_timer_entry pj_timer_entry
• typedef pj_atomic_t pj_atomic_t
• typedef PJ_ATOMIC_VALUE_TYPE pj_atomic_value_t
• typedef pj_thread_t pj_thread_t
• typedef pj_lock_t pj_lock_t
• typedef pj_mutex_t pj_mutex_t
• typedef pj_sem_t pj_sem_t
• typedef pj_event_t pj_event_t
• typedef pj_pipe_t pj_pipe_t
• typedef void * pj_oshandle_t
• typedef long pj_sock_t
• typedef void * pj_sockaddr_t
• typedef pj_sockaddr_in pj_sockaddr_in
• typedef unsigned int pj_color_t
• typedef int pj_exception_id_t
• typedef void(*) pj_exit_callback (void)

Enumerations

• enum { PJ_TERM_COLOR_R = 2, PJ_TERM_COLOR_G = 4, PJ_TERM_COLOR_B = 1, PJ_TERM_COLOR_BRIGHT = 8 }

Functions

• pj_status_t pj_init (void)
• void pj_shutdown (void)
• pj_status_t pj_atexit (pj_exit_callback func)
• pj_int16_t pj_swap16 (pj_int16_t val16)
• pj_int32_t pj_swap32 (pj_int32_t val32)
• void pj_time_val_normalize (pj_time_val *t)

Variables

• PJ_BEGIN_DECL typedef int pj_int32_t
8.26.2 Enumeration Type Documentation

8.26.2.1 anonymous enum

Color code combination.

Enumerator:

\begin{itemize}
  \item \texttt{PJ\_TERM\_COLOR\_R} \hspace{1em} Red
  \item \texttt{PJ\_TERM\_COLOR\_G} \hspace{1em} Green
  \item \texttt{PJ\_TERM\_COLOR\_B} \hspace{1em} Blue.
  \item \texttt{PJ\_TERM\_COLOR\_BRIGHT} \hspace{1em} Bright mask.
\end{itemize}
8.27 unicode.h File Reference

8.27.1 Detailed Description

Provides Unicode conversion for Unicode OSes.

Defines

- `#define PJ_DECL_UNICODE_TEMP_BUF(var, size)`
- `#define PJ_STRING_TO_NATIVE(s, buf, max) ((char*)s)`
- `#define PJ_DECL_ANSI_TEMP_BUF(buf, size)`
- `#define PJ_NATIVE_TO_STRING(cs, buf, max) ((char*)(const char*)cs)`

Functions

- `wchar_t * pj_ansi_to_unicode(const char *str, pj_size_t len, wchar_t *wbuf, pj_size_t wbuf_count)`
- `char * pj_unicode_to_ansi(const wchar_t *wstr, pj_size_t len, char *buf, pj_size_t buf_size)`
Chapter 9

PJLIB Reference Page Documentation

9.1 Coding Convention

Before you submit your code/patches to be included with PJLIB, you must make sure that your code is compliant with PJLIB coding convention. **This is very important!** Otherwise we would not accept your code.

9.1.1 Editor Settings

The single most important thing in the whole coding convention is editor settings. It’s more important than the correctness of your code (bugs will only crash the system, but incorrect tab size is mental!).

Kindly set your editor as follows:

- tab size to 8.
- indentation to 4.

With vi, you can do it with:

```bash
:se ts=8
:se sts=4
```

You should replace tab with eight spaces.

9.1.2 Coding Style

Coding style MUST strictly follow K&R style. The rest of coding style must follow current style. You SHOULD be able to observe the style currently used by PJLIB from PJLIB sources, and apply the style to your code. If you're not able to do simple thing like to observe PJLIB coding style from the sources, then logic dictates that your ability to observe more difficult area in PJLIB such as memory allocation strategy, concurrency, etc is questionable.

9.1.3 Commenting Your Code

Public API (e.g. in header files) MUST have doxygen compliant comments.
9.2 Building, and Installing PJLIB

9.2.1 Build and Installation

Note:

The most up-to-date information on building and installing PJLIB should be found in the website, under "Getting Started" document. More over, the new PJLIB build system is now based on autoconf, so some of the information here might not be relevant anymore (although most still are, since the autoconf script still use the old Makefile system as the backend).

9.2.1.1 Visual Studio

The PJLIB Visual Studio workspace supports the building of PJLIB for Win32 target. Although currently only the Visual Studio 6 Workspace is actively maintained, developers with later version of Visual Studio can easily imports VS6 workspace into their IDE.

To start building PJLIB projects with Visual Studio 6 or later, open the workspace file in the corresponding build directory. You have several choices on which dsw file to open:

```
$PJPROJECT/pjlib/build/pjlib.dsw
$PJPROJECT/pjsip/build/pjsip.dsw
..etc
```

The easiest way is to open pjsip_apps.dsw file in $PJPROJECT/pjsip-apps/build directory, and build pjsua project or the samples project. However this will not build the complete projects. For example, the PJLIB test is not included in this workspace. To build the complete projects, you must open and build each dsw file in build directory in each subprojects. For example, to open the complete PJLIB workspace, open pjlib.dsw in $PJPROJECT/pjlib/build directory.

Create config_site.h  The file $PJPROJECT/pjlib/include/pj/config_site.h is supposed to contain configuration that is specific to your site/target. This file is not part of PJLIB, so you must create it yourself. Normally you just need to create a blank file.

The reason why it’s not included in PJLIB is so that you would not accidently overwrite your site configuration.

If you fail to do this, Visual C will complain with error like:

"fatal error C1083: Cannot open include file: ‘pj/config_site.h’: No such file or directory".

Build the Projects  Just hit the build button!

9.2.1.2 Make System

For other targets, PJLIB provides a rather comprehensive build system that uses GNU make (and only GNU make will work). Currently, the build system supports building * PJLIB for these targets:

- i386/Win32/mingw
- i386/Linux
- i386/Linux (kernel)
9.2 Building, and Installing PJLIB

- alpha/linux
- sparc/SunOS
- etc..

Requirements In order to use the make based build system, you MUST have:

- GNU make
  The Makefiles heavily utilize GNU make commands which most likely are not available in other make system.
- bash shell is recommended.
  Specifically, there is a command "echo -n" which may not work in other shells.
  This command is used when generating dependencies (make dep) and it's located in $PJPROJECT/build/rules.mak.
- ar, ranlib from GNU binutils
  In your system has different ar or ranlib (e.g. they may have been installed as gar and granlib), then either you create the relevant symbolic links, or modify $PJPROJECT/build/cc-gcc.mak and rename ar and ranlib to the appropriate names.
- gcc to generate dependency.
  Currently the build system uses "gcc -MM" to generate build dependencies. If gcc is not desired to generate dependency, then either you don't run make dep, or edit $PJPROJECT/build/rules.mak to calculate dependency using your prefered method. (And let me know when you do so that I can update the file. :)

Building the Project Generally, steps required to build the PJLIB are:

$ cd /home/user/pjproject
$ ./configure
$ touch pjlib/include/pj/config_site.h
$ make dep
$ make

The above process will build all static libraries and all applications.

Note:
  the configure script is not a proper autoconf script, but rather a simple shell script to detect current host. This script currently does not support cross-compilation.
  For Linux kernel target, there are additional steps required, which will be explained in section Linux Kernel Target.

Cross Compilation For cross compilation, you will need to edit the build.mak file in $PJPROJECT root directory manually. Please see README-configure file in the root directory for more information.

For Linux kernel target, you are also required to declare the following variables in this file:

- KERNEL_DIR: full path of kernel source tree.
- KERNEL_ARCH: kernel ARCH options (e.g. "ARCH=um"), or leave blank for default.
Apart from these, there are also additional steps required to build Linux kernel target, which will be explained in Linux Kernel Target.

Files in "build" Directory The *.mak files in $PJPROJECT/build directory are used to specify the configuration for the specified compiler, target machine target operating system, and host options. These files will be executed (included) by make during building process, depending on the values specified in $PJPROJECT/build.mak file.

Normally you don’t need to edit these files, except when you’re porting PJLIB to new target.

Below are the description of some files in this directory:

- rules.mak: contains generic rules always included during make.
- cc-gcc.mak: rules when gcc is used for compiler.
- cc-vc.mak: rules when MSVC compiler is used.
- host-mingw.mak: rules for building in mingw host.
- host-unix.mak: rules for building in Unix/Posix host.
- host-win32.mak: rules for building in Win32 command console (only valid when VC is used).
- m-i386.mak: rules when target machine is an i386 processor.
- m-m68k.mak: rules when target machine is an m68k processor.
- os-linux.mak: rules when target OS is Linux.
- os-linux-kernel.mak: rules when PJLIB is to be build as part of Linux kernel.
- os-win32.mak: rules when target OS is Win32.

Create config_site.h The file $PJPROJECT/pjlib/include/pj/config_site.h is supposed to contain configuration that is specific to your site/target. This file is not part of PJLIB, so you must create it yourself.

The reason why it’s not included in PJLIB is so that you would not accidently overwrite your site configuration.

Invoking make Normally, make is invoked in build directory under each project. For example, to build PJLIB, you would invoke make in $PJPROJECT/pjlib/build directory like below:

```
$ cd pjlib/build
$ make
```

Alternatively you may invoke make in $PJPROJECT directory, to build all projects under that directory (e.g. PJLIB, PJSIP, etc.).
9.2 Building, and Installing PJLIB

Linux Kernel Target

Note:

BUILDING APPLICATIONS IN LINUX KERNEL MODE IS A VERY DANGEROUS BUSINESS. YOU MAY CRASH THE WHOLE OF YOUR SYSTEM, CORRUPT YOUR HARDISK, ETC. PJLIB KERNEL MODULES ARE STILL IN EXPERIMENTAL PHASE, DO NOT RUN IT IN PRODUCTION SYSTEMS OR OTHER SYSTEMS WHERE RISK OF LOSS OF DATA IS NOT ACCEPTABLE. YOU HAVE BEEN WARNED.

User Mode Linux (UML) provides excellent way to experiment with Linux kernel without risking the stability of the host system. See http://user-mode-linux.sourceforge.net for details.

I only use UML to experiment with PJLIB kernel modules. I wouldn’t be so foolish to use my host Linux machine to experiment with this.

You have been warned.

For building PJLIB for Linux kernel target, there are additional steps required. In general, the additional tasks are:

- Declare some more variables in build.mak file (this has been explained in Cross Compilation above).

Perform these two small modifications in kernel source tree.

There are two small modification need to be applied to the kernel tree.

1. Edit Makefile in kernel root source tree.

Add the following lines at the end of the Makefile in your $KERNEL_SRC dir:

```
script:
    $(SCRIPT)
```

Note:

Remember to replace spaces with tab in the Makefile.

The modification above is needed to capture kernel’s $CFLAGS and $CFLAGS_MODULE which will be used for PJLIB’s compilation.

2. Add Additional Exports.

We need the kernel to export some more symbols for our use. So we declare the additional symbols to be exported in extra-exports.c file, and add a this file to be compiled into the kernel:

- Copy the file extra-exports.c from pjlib/src/pj directory to $KERNEL_SRC/kernel/ directory.

- Edit Makefile in that directory, and add this line somewhere after the declaration of that variable:

  ```
  obj-y += extra-exports.o
  ```

To illustrate what have been done in your kernel source tree, below is screenshot of my kernel source tree _after_ the modification.

```
[root@vpc-linux linux-2.6.7]# pwd
/usr/src/linux-2.6.7
```
Then you must rebuild the kernel. If you fail to do this, you won’t be able to \texttt{insmod} pjlib.

\textbf{Note:}

You will see a lot of warning messages during pjlib-test compilation. The warning messages complain about unresolved symbols which are defined in pjlib module. You can safely ignore these warnings. However, you can not ignore warnings about non-pjlib unresolved symbols.

\subsection{Makefile Explained}

The Makefile for each project (e.g. PJLIB, PJSIP, etc) should be very similar in the contents. The Makefile is located under \texttt{build} directory in each project subdir.

\textbf{PJLIB Makefile.}  Below is PJLIB’s Makefile:
9.2 Building, and Installing PJLIB
include ../../build.mak
include $(PJDIR)/build/common.mak
RULES_MAK := $(PJDIR)/build/rules.mak

export PJLIB_LIB := ../lib/libpj-$(TARGET_NAME)$(LIBEXT)
###############################################################################
# Gather all flags.
#
export _CFLAGS := $(CC_CFLAGS) $(OS_CFLAGS) $(HOST_CFLAGS) $(M_CFLAGS) \
$(CFLAGS) $(CC_INC)../include
export _CXXFLAGS:= $(_CFLAGS) $(CC_CXXFLAGS) $(OS_CXXFLAGS) $(M_CXXFLAGS) \
$(HOST_CXXFLAGS) $(CXXFLAGS)
export _LDFLAGS := $(subst /,$(HOST_PSEP),$(PJLIB_LIB)) \
$(CC_LDFLAGS) $(OS_LDFLAGS) $(M_LDFLAGS) $(HOST_LDFLAGS) \
$(LDFLAGS)
###############################################################################
# Defines for building PJLIB library
#
export PJLIB_SRCDIR = ../src/pj
export PJLIB_OBJS += $(OS_OBJS) $(M_OBJS) $(CC_OBJS) $(HOST_OBJS) \
array.o config.o ctype.o errno.o except.o fifobuf.o guid.o \
hash.o ip_helper_generic.o list.o lock.o log.o os_time_common.o \
pool.o pool_buf.o pool_caching.o pool_dbg.o rand.o \
rbtree.o sock_common.o string.o timer.o types.o
export PJLIB_CFLAGS += $(_CFLAGS)
###############################################################################
# Defines for building test application
#
export TEST_SRCDIR = ../src/pjlib-test
export TEST_OBJS += atomic.o echo_clt.o errno.o exception.o \
fifobuf.o file.o ioq_perf.o ioq_udp.o ioq_unreg.o \
ioq_tcp.o \
list.o mutex.o os.o pool.o pool_perf.o rand.o rbtree.o \
select.o sleep.o sock.o sock_perf.o \
string.o test.o thread.o timer.o timestamp.o \
udp_echo_srv_sync.o udp_echo_srv_ioqueue.o \
util.o
export TEST_CFLAGS += $(_CFLAGS)
export TEST_LDFLAGS += $(_LDFLAGS)
export TEST_EXE := ../bin/pjlib-test-$(TARGET_NAME)$(HOST_EXE)

export CC_OUT CC AR RANLIB HOST_MV HOST_RM HOST_RMDIR HOST_MKDIR OBJEXT LD LDOUT
###############################################################################
# Main entry
#
# $(TARGET) is defined in os-$(OS_NAME).mak file in current directory.
#
all: $(TARGETS)
doc:
cd .. && doxygen docs/doxygen.cfg
print:
$(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib print_lib
$(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-test print_bin
depend: ../include/pj/config_site.h
$(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib depend
$(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-test depend
echo ’$(TEST_EXE): $(PJLIB_LIB)’ >> .pjlib-test-$(TARGET_NAME).depend

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen

267


.PHONY: dep depend pjlib pjlib-test clean realclean distclean

dep: depend

pjlib: ../include/pj/config_site.h
    $(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib $(PJLIB_LIB)

../include/pj/config_site.h:
    touch ../include/pj/config_site.h

pjlib-test:
    $(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-test $(TEST_EXE)

.PHONY: ../lib/pjlib.ko
../lib/pjlib.ko:
    echo Making $@
    $(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib $@

.PHONY: ../lib/pjlib-test.ko
../lib/pjlib-test.ko:
    $(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-test $@

clean:
    $(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib clean
    $(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-clean

reaclean:
    $(subst @@,$(subst /,$(HOST_PSEP),.pjlib-$(TARGET_NAME).depend),$(HOST_RMR))
    $(subst @@,$(subst /,$(HOST_PSEP),.pjlib-test-$(TARGET_NAME).depend),$(HOST_RMR))
    $(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib realclean
    $(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-test realclean

distclean: reaclean

gcov-report:
    $(MAKE) -f $(RULES_MAK) APP=PJLIB app=pjlib gcov-report
    $(MAKE) -f $(RULES_MAK) APP=TEST app=pjlib-test gcov-report

PJLIB os-linux.mak. Below is file os-linux.mak file in $PJPROJECT/pjlib/build directory, which is OS specific configuration file for Linux target that is specific for PJLIB project. For global OS specific configuration, please see $PJPROJECT/build/os-*.mak.

# OS specific configuration for Linux OS target.
#
# PJLIB_OBJS specified here are object files to be included in PJLIB
# (the library) for this specific operating system. Object files common
# to all operating systems should go in Makefile instead.
# export PJLIB_OBJS += addr_resolv_sock.o file_access_unistd.o \file_io_ansi.o guid_simple.o \log_writer_stdout.o os_core_unix.o \os_error_unix.o os_time_unix.o \os_timestamp_common.o os_timestamp_posix.o \pool_policy_malloc.o sock_bsd.o sock_select.o
ifeq (epoll,$(LINUX_POLL))
export PJLIB_OBJS += ioqueue_epoll.o
else
export PJLIB_OBJS += ioqueue_select.o

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
endif

# TEST_OBJS are operating system specific object files to be included in
# the test application.
# export TEST_OBJS += main.o

# Additional LDFlags for pjlib-test
# export TEST_LDFLAGS += -lm

# TARGETS are make targets in the Makefile, to be executed for this given
# operating system.
# export TARGETS = pjlib pjlib-test
9.3 Porting PJLIB

Note:

Since version 0.5.8, PJLIB build system is now based on autoconf, so most of the time we shouldn’t need to apply the tweakings below to get PJLIB working on a new platform. However, since the autoconf build system still uses the old Makefile build system, the information below may still be useful for reference.

9.3.1 Porting to New CPU Architecture

Below is step-by-step guide to add support for new CPU architecture. This sample is based on porting to Alpha architecture; however steps for porting to other CPU architectures should be pretty similar.

Also note that in this example, the operating system used is Linux. Should you wish to add support for new operating system, then follow the next section Porting to New Operating System Target.

Step-by-step guide to port to new CPU architecture:

• decide the name for the new architecture. In this case, we choose alpha.

• edit file $PJPROJECT/build.mak, and add new section for the new target:

```bash
# Linux alpha, gcc
export MACHINE_NAME := alpha
export OS_NAME := linux
export CC_NAME := gcc
export HOST_NAME := unix
```

• create a new file $PJPROJECT/build/m-alpha.mak. Alternatively create a copy from other file in this directory. The contents of this file will look something like:

```bash
export M_CFLAGS := PJ_M_ALPHA=1
export M_CXXFLAGS :=
export M_LDFLAGS :=
export M_SOURCES :=
```

• create a new file $PJPROJECT/pjlib/include/pj/compat/m_alpha.h. Alternatively create a copy from other header file in this directory. The contents of this file will look something like:

```bash
define PJ_HAS_PENTIUM 0
define PJ_IS_LITTLE_ENDIAN 1
define PJ_IS_BIG_ENDIAN 0
```

• edit pjlib/include/pj/config.h. Add new processor configuration in this header file, like follows:

```c
...  
eif defined (PJ_M_ALPHA) && PJ_M_ALPHA != 0
    # include <pj/compat/m_alpha.h>
...  
```
9.3 Porting PJLIB

- done. Build PJLIB with:

```bash
$ cd $PJPROJECT/pjlib/build
$ make dep
$ make clean
$ make
```

9.3.2 Porting to New Operating System Target

This section will try to give you rough guideline on how to port PJLIB to a new target. As a sample, we give the target a name tag, for example xos (for X OS).

9.3.2.1 Create New Compat Header File

You’ll need to create a new header file `include/pj/compat/os_xos.h`. You can copy as a template other header file and edit it accordingly.

9.3.2.2 Modify config.h

Then modify file `include/pj/config.h` to include this file accordingly (e.g. when macro PJ_XOS is defined):

```c
... #elif defined(PJ_XOS)
    # include <pj/compat/os_xos.h>
#else
#...
```

9.3.2.3 Create New Global Make Config File

Then you’ll need to create global configuration file that is specific for this OS, i.e. os-xos.mak in `$PJPROJECT/build` directory.

At very minimum, the file will normally need to define PJ_XOS=1 in the CFLAGS section:

```bash
# $PJPROJECT/build/os-xos.mak:
# export OS_CFLAGS := $(CC_DEF)PJ_XOS=1
export OS_CXXFLAGS :=
export OS_LDFLAGS :=
export OS_SOURCES :=
```

9.3.2.4 Create New Project’s Make Config File

Then you’ll need to create xos-specific configuration file for PJLIB. This file is also named os-xos.mak, but its located in pjlib/build directory. This file will specify source files that are specific to this OS to be included in the build process.

Below is a sample:
9.3.2.5 Create and Edit Source Files

You’ll normally need to create at least these files:

- **os_core_xos.c**: core OS specific functionality.
- **os_timestamp_xos.c**: how to get timestamp in this OS.

Depending on how things are done in your OS, you may need to create these files:

- **os_error_*.c**: how to manipulate OS error codes. Alternatively you may use existing `os_error_unix.c` if the OS has `errno` and `strerror()` function.
- **ioqueue_*.c**: if the OS has specific method to perform asynchronous I/O. Alternatively you may use existing `ioqueue_select.c` if the OS supports `select()` function call.
- **sock_*.c**: if the OS has specific method to perform socket communication. Alternatively you may use existing `sock_bsd.c` if the OS supports BSD socket API, and edit `include/pj/compat/socket.h` file accordingly.

You will also need to check various files in `include/pj/compat/*`, to see if they’re compatible with your OS.

9.3.2.6 Build The Project

After basic building blocks have been created for the OS, then the easiest way to see which parts need to be fixed is by building the project and see the error messages.

9.3.2.7 Editing Existing Files vs Creating New File

When you encounter compatibility errors in PJLIB during porting, you have three options on how to fix the error:

- edit the existing *.c file, and give it `ifdef` switch for the new OS, or
- edit `include/pj/compat/*.h` instead, or
- create a totally new file.

Basically there is no strict rule on which approach is the best to use, however the following guidelines may be used:

- if the file is expected to be completely different than any existing file, then perhaps you should create a completely new file. For example, file `os_core_xxx.c` will normally be different for each OS flavour.
• if the difference can be localized in `include/compat` header file, and existing `ifdef` switch is there, then preferably you should edit this `include/compat` header file.

• if the existing `*.c` file has `ifdef` switch, then you may add another `elif` switch there. This normally is used for behaviors that are not totally different on each platform.

• other than that above, use your own judgement on whether to edit the file or create new file etc.
9.4 Example: Exception Handling

Below is sample program to demonstrate how to use exception handling.

```c
/* $Id: except.c 974 2007-02-19 01:13:53Z benny@prijono.org $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include <pj/except.h>
#include <pj/rand.h>
#include <stdio.h>
#include <stdlib.h>

static pj_exception_id_t NO_MEMORY, OTHER_EXCEPTION;

static void randomly_throw_exception()
{
    if (pj_rand() % 2)
        PJ_THROW(OTHER_EXCEPTION);
}

static void *my_malloc(size_t size)
{
    void *ptr = malloc(size);
    if (!ptr)
        PJ_THROW(NO_MEMORY);
    return ptr;
}

static int test_exception()
{
    PJ_USE_EXCEPTION;
    
    PJ_TRY {
        void *data = my_malloc(200);
        free(data);
        randomly_throw_exception();
    }
    PJ_CATCH_ANY {
        pj_exception_id_t x_id;
        x_id = PJ_GET_EXCEPTION();
        printf("Caught exception %d (%s)\n",
               x_id, pj_exception_id_name(x_id));
    }
    PJ_END
    return 1;
}

int main()
{
    pj_status_t rc;
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
// Error handling is omitted for clarity.
rc = pj_init();
rc = pj_exception_id_alloc("No Memory", &NO_MEMORY);
rc = pj_exception_id_alloc("Other Exception", &OTHER_EXCEPTION);
return test_exception();
9.5 Example: List Manipulation

Below is a sample program to demonstrate how to manipulate a linked list.

```c
/* $Id: list.c 974 2007-02-19 01:13:53Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 */
/*
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 */
/*
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include <pj/list.h>
#include <pj/assert.h>
#include <pj/log.h>

struct my_node
{
    // This must be the first member declared in the struct!
    PJ_DECL_LIST_MEMBER(struct my_node);
    int value;
};

int main()
{
    struct my_node nodes[10];
    struct my_node list;
    struct my_node *it;
    int i;

    // Initialize the list as empty.
    pj_list_init(&list);
    // Insert nodes.
    for (i=0; i<10; ++i) {
        nodes[i].value = i;
        pj_list_insert_before(&list, &nodes[i]);
    }
    // Iterate list nodes.
    it = list.next;
    while (it != &list) {
        PJ_LOG(3, ("list", "value = %d", it->value));
        it = it->next;
    }
    // Erase all nodes.
    for (i=0; i<10; ++i) {
        pj_list_erase(&nodes[i]);
    }
    // List must be empty by now.
    pj_assert( pj_list_empty(&list) );
    return 0;
```

---

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
71 );
9.6 Example: Log, Hello World

Very simple program to write log.

```c
/* $Id: log.c 974 2007-02-19 01:13:53Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 */
#include <pj/log.h>
int main()
{    pj_status_t rc;
    // Error handling omitted for clarity
    // Must initialize PJLIB first!
    rc = pj_init();
    PJ_LOG(3, ("main.c", "Hello world!");
    return 0;
}
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
9.7 Test: Atomic Variable

This file provides implementation of atomic_test(). It tests the functionality of the atomic variable API.

9.7.1 Scope of the Test

API tested:

- pj_atomic_create()
- pj_atomic_get()
- pj_atomic_inc()
- pj_atomic_dec()
- pj_atomic_set()
- pj_atomic_destroy()

This file is pjlib-test/atomic.c

```c
/* $Id: atomic.c 974 2007-02-19 01:13:53Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include "test.h"
#include <pjlib.h>

#if INCLUDE_ATOMIC_TEST

int atomic_test(void)
{
  pj_pool_t *pool;
  pj_atomic_t *atomic_var;
  pj_status_t rc;

  pool = pj_pool_create(mem, NULL, 4096, 0, NULL);
  if (!pool)
    return -10;

  /* create() */
  rc = pj_atomic_create(pool, 111, &atomic_var);
  if (rc != 0)
  {
    return -20;
  }

  /* get: check the value. */

  return 0;
}
#endif
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
if (pj_atomic_get(atomic_var) != 111)
    return -30;

/* increment. */
pj_atomic_inc(atomic_var);
if (pj_atomic_get(atomic_var) != 112)
    return -40;

/* decrement. */
pj_atomic_dec(atomic_var);
if (pj_atomic_get(atomic_var) != 111)
    return -50;

/* set */
pj_atomic_set(atomic_var, 211);
if (pj_atomic_get(atomic_var) != 211)
    return -60;

/* add */
pj_atomic_add(atomic_var, 10);
if (pj_atomic_get(atomic_var) != 221)
    return -60;

/* check the value again. */
if (pj_atomic_get(atomic_var) != 221)
    return -70;

/* destroy */
rc = pj_atomic_destroy(atomic_var);
if (rc != 0)
    return -80;

pj_pool_release(pool);
return 0;
}

#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled. */
int dummy_atomic_test;
#endif /* INCLUDE_ATOMIC_TEST */
9.8 Test: Exception Handling

This file provides implementation of exception_test(). It tests the functionality of the exception handling API.

Note:
This test use static ID not acquired through proper registration. This is not recommended, since it may create ID collisions.

9.8.1 Scope of the Test

Some scenarios tested:

- no exception situation
- basic TRY/CATCH
- multiple exception handlers
- default handlers

This file is pjlib-test/exception.c

```c
/* $Id: exception.c 974 2007-02-19 01:13:53Z benny1p $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include "test.h"

#if INCLUDE_EXCEPTION_TEST
#include <pjlib.h>
#define ID_1 1
#define ID_2 2

static int throw_id_1(void)
{
    PJ_THROW( ID_1 );
    return -1;
}

static int throw_id_2(void)
{
    PJ_THROW( ID_2 );
}
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
return -1;
}

static int test(void)
{
    int rc = 0;
    PJ_USE_EXCEPTION;
    
    /*
    * No exception situation.
    */
    PJ_TRY {
        rc = rc;
    }
    PJ_CATCH_ANY {
        rc = -3;
    }
    PJ_END;
    if (rc != 0)
        return rc;
    
    /*
    * Basic TRY/CATCH
    */
    PJ_TRY {
        rc = throw_id_1();
        // should not reach here.
        rc = -10;
    }
    PJ_CATCH_ANY {
        int id = PJ_GET_EXCEPTION();
        if (id != ID_1) {
            PJ_LOG(3, ("", ...error: got unexpected exception \d (\%s)",
                        id, pj_exception_id_name(id)));
            if (!rc) rc = -20;
        }
    }
    PJ_END;
    if (rc != 0)
        return rc;
    
    /*
    * Multiple exceptions handlers
    */
    PJ_TRY {
        rc = throw_id_2();
        // should not reach here.
        rc = -25;
    }
    PJ_CATCH_ANY {
        switch (PJ_GET_EXCEPTION()) {
        case ID_1:
            if (!rc) rc = -30; break;
        case ID_2:
            if (!rc) rc = 0; break;
        default:
            if (!rc) rc = -40;
                break;
        }
    }
    PJ_END;
if (rc != 0)
    return rc;

/*
 * Test default handler.
 */
PJ_TRY {
    rc = throw_id_1();
    // should not reach here
    rc = -50;
}
PJ_CATCH_ANY {
    switch (PJ_GET_EXCEPTION()) {
    case ID_1:
        if (!rc) rc = 0;
        break;
    default:
        if (!rc) rc = -60;
        break;
    }
}
PJ_END;
if (rc != 0)
    return rc;
return 0;
}

int exception_test(void)
{
    int i, rc;
    enum { LOOP = 10 };
    for (i=0; i<LOOP; ++i) {
        if ((rc=test()) != 0) {
            PJ_LOG(3,("...failed at i=%d (rc=%d)", i, rc));
            return rc;
        }
    }
    return 0;
}

#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_exception_test;
#endif /* INCLUDE_EXCEPTION_TEST */
9.9 Test: I/O Queue Performance

Test the performance of the I/O queue, using typical producer consumer test. The test should examine the effect of using multiple threads on the performance.

This file is pjlib-test/ioq_perf.c

```
/* $Id: ioq_perf.c 1405 2007-07-20 08:08:30Z bennylp $ */
/*
* Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
#include "test.h"
#include <pjlib.h>
#include <pj/compat/high_precision.h>
#if INCLUDE_IOQUEUE_PERF_TEST
#if _MSC_VER
#pragma warning ( disable: 4204) // non-constant aggregate initializer
#endif
#define THIS_FILE "ioq_perf"
//define TRACE_(expr) PJ_LOG(3,expr)
#define TRACE_(expr)

static pj_bool_t thread_quit_flag;
static pj_status_t last_error;
static unsigned last_error_counter;
/* Descriptor for each producer/consumer pair. */
typedef struct test_item {
  pj_sock_t server_fd,
  client_fd;
  pj_ioqueue_t *ioqueue;
  pj_ioqueue_key_t *server_key, *client_key;
  pj_ioqueue_op_key_t recv_op, send_op;
  int has_pending_send;
  pj_size_t buffer_size;
  char outgoing_buffer[100000];
  char incoming_buffer[100000];
  pj_size_t buffer_sent, buffer_recv;
} test_item;

/* Callback when data has been read.*/
/* Increment item->bytes_recv and ready to read the next data. */
static void on_read_complete(pj_ioqueue_key_t *key,}
pj_ioqueue_op_key_t *op_key,
    pj_ssize_t bytes_read)
{
    test_item *item = (test_item*)pj_ioqueue_get_user_data(key);
    pj_status_t rc;
    int data_is_available = 1;

    TRACE_((THIS_FILE, " read complete, bytes_read=%d", bytes_read));

    do {
        if (thread_quit_flag)
            return;

        if (bytes_read < 0) {
            pj_status_t rc = -bytes_read;
            char errmsg[PJ_ERR_MSG_SIZE];
            if (rc != last_error) {
                //last_error = rc;
                pj_strerror(rc, errmsg, sizeof(errmsg));
                PJ_LOG(3,(THIS_FILE,"...error: read error, bytes_read=%d (%s)",
                    bytes_read, errmsg));
                PJ_LOG(3,(THIS_FILE,
                    ".....additional info: total read=%u, total sent=%u",
                    item->bytes_recv, item->bytes_sent));
            } else {
                last_error_counter++;
            }
            bytes_read = 0;
        } else if (bytes_read == 0) {
            PJ_LOG(3,(THIS_FILE, "...socket has closed!");
        }
        item->bytes_recv += bytes_read;

        /* To assure that the test quits, even if main thread
           doesn’t have time to run. */
        if (item->bytes_recv > item->buffer_size + 10000)
            thread_quit_flag = 1;

        bytes_read = item->buffer_size;
        rc = pj_ioqueue_recv(key, op_key,
            item->incoming_buffer, &bytes_read, 0);
        if (rc == PJ_SUCCESS) {
            data_is_available = 1;
        } else if (rc == PJ_EPENDING) {
            data_is_available = 0;
        } else {
            data_is_available = 0;
            if (rc != last_error) {
                last_error = rc;
                app_perror("...error: read error(1)", rc);
            } else {
                last_error_counter++;
            }
        }
        if (!item->has_pending_send) {
            pj_ssize_t sent = item->buffer_size;
            rc = pj_ioqueue_send(item->client_key, &item->send_op,
                item->outgoing_buffer, &sent, 0);
            if (rc != PJ_SUCCESS && rc != PJ_EPENDING) {
                app_perror("...error: write error", rc);
            }
        }
    }
    }
item->has_pending_send = (rc==PJ_EPENDING);
}
} while (data_is_available);

/* Callback when data has been written. 
* Increment item->bytes_sent and write the next data. */
static void on_write_complete(pj_ioqueue_key_t *key, 
pj_ioqueue_op_key_t *op_key, 
pj_ssize_t bytes_sent)
{
    test_item *item = (test_item*) pj_ioqueue_get_user_data(key);

    //TRACE_((THIS_FILE, " write complete: sent = %d", bytes_sent));

    if (thread_quit_flag)
        return;

    item->has_pending_send = 0;
    item->bytes_sent += bytes_sent;

    if (bytes_sent <= 0) {
        PJ_LOG(3, (THIS_FILE, "...error: sending stopped. bytes_sent=%d", 
            bytes_sent));
    } else {
        pj_status_t rc;

        bytes_sent = item->buffer_size;
        rc = pj_ioqueue_send( item->client_key, op_key, 
            item->outgoing_buffer, &bytes_sent, 0);
        if (rc != PJ_SUCCESS && rc != PJ_EPENDING) {
            app_perror("...error: write error", rc);
        }
        item->has_pending_send = (rc==PJ_EPENDING);
    }
}

struct thread_arg
{
    int id;
    pj_ioqueue_t *ioqueue;
    unsigned counter;
};

/* The worker thread. */
static int worker_thread(void *p)
{
    struct thread_arg *arg = (struct thread_arg*) p;
    const pj_time_val timeout = {0, 100};
    int rc;

    while (!thread_quit_flag) {

        ++arg->counter;
        rc = pj_ioqueue_poll(arg->ioqueue, &timeout);
        //TRACE_((THIS_FILE, " thread: poll returned rc=%d", rc));
        if (rc < 0) {
            char errmsg[PJ_ERR_MSG_SIZE];
            pj_strerror(-rc, errmsg, sizeof(errmsg));
            PJ_LOG(3, (THIS_FILE, "...error in pj_ioqueue_poll() in thread %d " 
                "after %d loop: %s [pj_status_t=%d]", 
                thread_id, arg->counter, errmsg, rc));
        }
    }
}
9.9 Test: I/O Queue Performance

```c
arg->id, arg->counter, errmsg, -rc));
    //return -1;
}
}
return 0;
}

/* Calculate the bandwidth for the specific test configuration.
 * The test is simple:
 * - create sockpair_cnt number of producer-consumer socket pair.
 * - create thread_cnt number of worker threads.
 * - each producer will send buffer_size bytes data as fast and
 *   as soon as it can.
 * - each consumer will read buffer_size bytes of data as fast
 *   as it could.
 * - measure the total bytes received by all consumers during a
 *   period of time.
 */
static int perform_test(int sock_type, const char *type_name,
                        unsigned thread_cnt, unsigned sockpair_cnt,
                        pj_size_t buffer_size,
                        pj_size_t *p_bandwidth)
{
    enum { MSEC_DURATION = 5000 };  
    pj_pool_t *pool;
    test_item *items;
    pj_thread_t **thread;
    pj_ioqueue_t *ioqueue;
    pj_status_t rc;
    pj_ioqueue_callback ioqueue_callback;
    pj_uint32_t total_elapsed_usec, total_received;
    pj_highprec_t bandwidth;
    pj_timestamp start, stop;
    unsigned i;

    TRACE_((THIS_FILE, " starting test..");

    ioqueue_callback.on_read_complete = &on_read_complete;
    ioqueue_callback.on_write_complete = &on_write_complete;

    thread_quit_flag = 0;

    pool = pj_pool_create(mem, NULL, 4096, 4096, NULL);
    if (!pool)
        return -10;
    items = (test_item*) pj_pool_alloc(pool, sockpair_cnt*sizeof(test_item));
    thread = (pj_thread_t**) pj_pool_alloc(pool, thread_cnt*sizeof(pj_thread_t*));

    TRACE_((THIS_FILE, " creating ioqueue..");
    rc = pj_ioqueue_create(pool, sockpair_cnt*2, &ioqueue);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: unable to create ioqueue", rc);
        return -15;
    }

    /* Initialize each producer-consumer pair. */
    for (i=0; i<sockpair_cnt; ++i) {
        pj_size_t bytes;

        items[i].ioqueue = ioqueue;
        items[i].buffer_size = buffer_size;
        items[i].outgoing_buffer = (char*) pj_pool_alloc(pool, buffer_size);
        items[i].incoming_buffer = (char*) pj_pool_alloc(pool, buffer_size);
        items[i].bytes_recv = items[i].bytes_sent = 0;
    }
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
/* randomize outgoing buffer. */
pj_create_random_string(items[i].outgoing_buffer, buffer_size);

/* Create socket pair. */
TRACE_((THIS_FILE, " calling socketpair..");
rc = app_socketpair(pj_AF_INET(), sock_type, 0,
   &items[i].server_fd, &items[i].client_fd);
if (rc != PJ_SUCCESS) {
   app_perror("...error: unable to create socket pair", rc);
   return -20;
}

/* Register server socket to ioqueue. */
TRACE_((THIS_FILE, " register(1).."));
rc = pj_ioqueue_register_sock(pool, ioqueue,
   items[i].server_fd,
   &items[i].ioqueue_callback,
   &items[i].server_key);
if (rc != PJ_SUCCESS) {
   app_perror("...error: registering server socket to ioqueue", rc);
   return -60;
}

/* Register client socket to ioqueue. */
TRACE_((THIS_FILE, " register(2).."));
rc = pj_ioqueue_register_sock(pool, ioqueue,
   items[i].client_fd,
   &items[i].ioqueue_callback,
   &items[i].client_key);
if (rc != PJ_SUCCESS) {
   app_perror("...error: registering server socket to ioqueue", rc);
   return -70;
}

/* Start reading. */
TRACE_((THIS_FILE, " pj_ioqueue_recv..");
bytes = items[i].buffer_size;
rc = pj_ioqueue_recv(items[i].server_key, &items[i].recv_op,
   items[i].incoming_buffer, &bytes, 0);
if (rc != PJ_EPENDING) {
   app_perror("...error: pj_ioqueue_recv", rc);
   return -73;
}

/* Start writing. */
TRACE_((THIS_FILE, " pj_ioqueue_write..");
bytes = items[i].buffer_size;
rc = pj_ioqueue_send(items[i].client_key, &items[i].send_op,
   items[i].outgoing_buffer, &bytes, 0);
if (rc != PJ_SUCCESS && rc != PJ_EPENDING) {
   app_perror("...error: pj_ioqueue_write", rc);
   return -76;
}
items[i].has_pending_send = (rc==PJ_EPENDING);

/* Create the threads. */
for (i=0; i<thread_cnt; ++i) {
   struct thread_arg *arg;
   arg = (struct thread_arg*) pj_pool_zalloc(pool, sizeof(*arg));
   arg->id = i;
   arg->ioqueue = ioqueue;
   arg->counter = 0;
rc = pj_thread_create(pool, NULL,
            &worker_thread,
            arg,
            PJ_THREAD_DEFAULT_STACK_SIZE,
            PJ_THREAD_SUSPENDED, &thread[i] );
if (rc != PJ_SUCCESS) {
    app_error("...error: unable to create thread", rc);
    return -80;
}
/* Mark start time. */
rc = pj_get_timestamp(&start);
if (rc != PJ_SUCCESS)
    return -90;
/* Start the thread. */
TRACE_((THIS_FILE, " resuming all threads.."));
for (i=0; i<thread_cnt; ++i) {
    rc = pj_thread_resume(thread[i]);
    if (rc != 0)
        return -100;
}
/* Wait for MSEC_DURATION seconds. *
 * This should be as simple as pj_thread_sleep(MSEC_DURATION) actually,
 * but unfortunately it doesn’t work when system doesn’t employ
 * timeslicing for threads.
 */
TRACE_((THIS_FILE, " wait for few seconds.."));
do {
    pj_thread_sleep(1);
    /* Mark end time. */
    rc = pj_get_timestamp(&stop);
    if (thread_quit_flag) {
        TRACE_((THIS_FILE, " transfer limit reached.."));
        break;
    }
    if (pj_elapsed_usec(&start,&stop)<MSEC_DURATION * 1000) {
        TRACE_((THIS_FILE, " time limit reached.."));
        break;
    }
} while (1);
/* Terminate all threads. */
TRACE_((THIS_FILE, " terminating all threads.."));
thread_quit_flag = 1;
for (i=0; i<thread_cnt; ++i) {
    TRACE_((THIS_FILE, " join thread %d..", i));
    pj_thread_join(thread[i]);
}
/* Close all sockets. */
TRACE_((THIS_FILE, " closing all sockets.."));
for (i=0; i<sockpair_cnt; ++i) {
    pj_ioqueue_unregister(items[i].server_key);
    pj_ioqueue_unregister(items[i].client_key);
}
/* Destroy threads */
for (i=0; i<thread_cnt; ++i) {
    pj_thread_destroy(thread[i]);
}
/* Destroy ioqueue. */
TRACE_((THIS_FILE, " destroying ioqueue.."));
pj_ioqueue_destroy(ioqueue);

/* Calculate actual time in usec. */
total_elapsed_usec = pj_elapsed_usec(&start, &stop);

/* Calculate total bytes received. */
total_received = 0;
for (i=0; i<sockpair_cnt; ++i) {
    total_received = items[i].bytes_recv;
}

/* bandwidth = total_received*1000/total_elapsed_usec */
bandwidth = total_received;
bandwidth = pj_highprec_mul(bandwidth, 1000);
bandwidth = pj_highprec_div(bandwidth, total_elapsed_usec);
*p_bandwidth = (pj_uint32_t)bandwidth;

PJ_LOG(3,(THIS_FILE, " %.4s %2d %2d %8d KB/s",
    type_name, thread_cnt, sockpair_cnt,
    *p_bandwidth));

/* Done. */
pj_pool_release(pool);
TRACE_((THIS_FILE, " done.."));
return 0;

/* main test entry. */
int ioqueue_perf_test(void)
{ enum { BUF_SIZE = 512 }; int i, rc;
  struct {
      int type;
      const char *type_name;
      int thread_cnt;
      int sockpair_cnt;
  } test_param[] = {
    { pj_SOCK_DGRAM(), "udp", 1, 1 },
    { pj_SOCK_DGRAM(), "udp", 1, 2 },
    { pj_SOCK_DGRAM(), "udp", 1, 4 },
    { pj_SOCK_DGRAM(), "udp", 1, 8 },
    { pj_SOCK_DGRAM(), "udp", 2, 1 },
    { pj_SOCK_DGRAM(), "udp", 2, 2 },
    { pj_SOCK_DGRAM(), "udp", 2, 4 },
    { pj_SOCK_DGRAM(), "udp", 2, 8 },
    { pj_SOCK_DGRAM(), "udp", 4, 1 },
    { pj_SOCK_DGRAM(), "udp", 4, 2 },
    { pj_SOCK_DGRAM(), "udp", 4, 4 },
    { pj_SOCK_DGRAM(), "udp", 4, 8 },
    { pj_SOCK_DGRAM(), "udp", 4, 16 },
    { pj_SOCK_STREAM(), "tcp", 1, 1 },
    { pj_SOCK_STREAM(), "tcp", 1, 2 },
    { pj_SOCK_STREAM(), "tcp", 1, 4 },
    { pj_SOCK_STREAM(), "tcp", 1, 8 },
    { pj_SOCK_STREAM(), "tcp", 2, 1 },
    { pj_SOCK_STREAM(), "tcp", 2, 2 },
    { pj_SOCK_STREAM(), "tcp", 2, 4 },

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
9.9 Test: I/O Queue Performance

{ pj_SOCK_STREAM(), "tcp", 2, 8 },
{ pj_SOCK_STREAM(), "tcp", 4, 1 },
{ pj_SOCK_STREAM(), "tcp", 4, 2 },
{ pj_SOCK_STREAM(), "tcp", 4, 4 },
{ pj_SOCK_STREAM(), "tcp", 4, 8 },
{ pj_SOCK_STREAM(), "tcp", 4, 16 },
/*
{ pj_SOCK_DGRAM(), "udp", 32, 1 },
{ pj_SOCK_DGRAM(), "udp", 32, 1 },
{ pj_SOCK_DGRAM(), "udp", 32, 1 },
{ pj_SOCK_DGRAM(), "udp", 32, 1 },
{ pj_SOCK_DGRAM(), "udp", 1, 32 },
{ pj_SOCK_DGRAM(), "udp", 1, 32 },
{ pj_SOCK_DGRAM(), "udp", 1, 32 },
{ pj_SOCK_DGRAM(), "udp", 1, 32 },
{ pj_SOCK_STREAM(), "tcp", 32, 1 },
{ pj_SOCK_STREAM(), "tcp", 32, 1 },
{ pj_SOCK_STREAM(), "tcp", 32, 1 },
{ pj_SOCK_STREAM(), "tcp", 32, 1 },
{ pj_SOCK_STREAM(), "tcp", 1, 32 },
{ pj_SOCK_STREAM(), "tcp", 1, 32 },
{ pj_SOCK_STREAM(), "tcp", 1, 32 },
{ pj_SOCK_STREAM(), "tcp", 1, 32 },
*/

pj_size_t best_bandwidth;
int best_index = 0;

PJ_LOG(3,(THIS_FILE, " Benchmarking %s ioqueue:", pj_ioqueue_name()));
PJ_LOG(3,(THIS_FILE, "======================================"));
Pj_LOG(3,(THIS_FILE, " Type Threads Skt.Pairs Bandwidth"));
Pj_LOG(3,(THIS_FILE, "======================================"));

best_bandwidth = 0;
for (i=0; i<(int)(sizeof(test_param)/sizeof(test_param[0])); ++i) {
    pj_size_t bandwidth;
    rc = perform_test(test_param[i].type,
    test_param[i].type_name,
    test_param[i].thread_cnt,
    test_param[i].sockpair_cnt,
    BUF_SIZE,
    &bandwidth);
    if (rc != 0)
        return rc;
    if (bandwidth > best_bandwidth)
        best_bandwidth = bandwidth, best_index = i;
    /* Give it a rest before next test, to allow system to close the
     * sockets properly.
     */
    pj_thread_sleep(500);
}
PJ_LOG(3,(THIS_FILE, " Best: Type=%s Threads=%d, Skt.Pairs=%d, Bandwidth=%u KB/s",
    test_param[best_index].type_name,
    test_param[best_index].thread_cnt,
    test_param[best_index].sockpair_cnt,
    best_bandwidth));
PJ_LOG(3,(THIS_FILE, " (Note: packet size=%d, total errors=%u)",
    BUF_SIZE, last_error_counter));
    return 0;
}
#else

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
/* To prevent warning about "translation unit is empty"
   * when this test is disabled.
   */
int dummy_uiq_perf_test;
#endif /* INCLUDE_IOQUEUE_PERF_TEST */
9.10 Test: I/O Queue (TCP)

This file provides implementation to test the functionality of the I/O queue when TCP socket is used. This file is `pjlib-test/ioq_tcp.c`

```c
/* $Id: ioq_tcp.c 1405 2007-07-20 08:08:30Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include "test.h"
#include <pjlib.h>
#if INCLUDE_TCP_IOQUEUE_TEST
#include <pjlib.h>
#endif
#if PJ_HAS_TCP
#define THIS_FILE "test_tcp"
#define NON_EXISTANT_PORT 50123
#define LOOP 100
#define BUF_MIN_SIZE 32
#define BUF_MAX_SIZE 2048
#define SOCK_INACTIVE_MIN (4-2)
#define SOCK_INACTIVE_MAX (PJ_IOQUEUE_MAX_HANDLES - 2)
#define POOL_SIZE (2*BUF_MAX_SIZE + SOCK_INACTIVE_MAX*128 + 2048)
static pj_ssize_t callback_read_size,
static unsigned callback_call_count;
static pj_ioqueue_key_t *callback_read_key,
static pj_ioqueue_op_key_t *callback_read_op;
static void on_ioqueue_read(pj_ioqueue_key_t *key,
static void on_ioqueue_write(pj_ioqueue_key_t *key,
static pj_ioqueue_key_t *op_key,
static pj_ssize_t bytes_read)
{ callback_read_key = key;
callback_read_op = op_key;
callback_read_size = bytes_read;
callback_call_count++; }
static void on_ioqueue_write(pj_ioqueue_key_t *key,
pj_ssize_t bytes_written)
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
{  
callback_write_key = key;  
callback_write_op = op_key;  
callback_write_size = bytes_written;  
callback_call_count++;  
}

static void on_ioqueue_accept(pj_ioqueue_key_t *key,  
   pj_ioqueue_op_key_t *op_key,  
   pj_sock_t sock,  
   int status)  
{
  if (sock == PJ_INVALID_SOCKET) {
    if (status != PJ_SUCCESS) {
      /* Ignore. Could be blocking error */
      app_perror(".....warning: received error in on_ioqueue_accept() callback",  
                 status);  
    } else {  
      callback_accept_status = -61;  
      PJ_LOG(3,("", "..... on_ioqueue_accept() callback was given "  
         "invalid socket and status is %d", status));  
    }
  } else {  
    callback_accept_key = key;  
    callback_accept_op = op_key;  
    callback_accept_status = status;  
    callback_call_count++;  
  }
}

static void on_ioqueue_connect(pj_ioqueue_key_t *key, int status)  
{
  callback_connect_key = key;  
  callback_connect_status = status;  
  callback_call_count++;  
}

static pj_ioqueue_callback test_cb =  
{  
&on_ioqueue_read,  
&on_ioqueue_write,  
&on_ioqueue_accept,  
&on_ioqueue_connect,  
};

static int send_recv_test(pj_ioqueue_t *ioque,  
   pj_ioqueue_key_t *skey,  
   pj_ioqueue_key_t *ckey,  
   void *send_buf,  
   void *recv_buf,  
   pj_ssize_t bufsize,  
   pj_timestamp *t_elapsed)  
{
  pj_status_t status;  
  pj_ssize_t bytes;  
  pj_time_val timeout;  
  pj_timestamp t1, t2;  
  int pending_op = 0;  
  pj_ioqueue_op_key_t read_op, write_op;  

  // Start reading on the server side.  
  bytes = bufsize;  
  status = pj_ioqueue_recv(skey, &read_op, recv_buf, &bytes, 0);  
  if (status != PJ_SUCCESS && status != PJ_EPENDING) {
    app_perror("...pj_ioqueue_recv error", status);  
    return -100;
  }
if (status == PJ_EPENDING)
    ++pending_op;
else {
    /* Does not expect to return error or immediate data. */
    return -115;
}

// Randomize send buffer.
pj_create_random_string((char*)send_buf, bufsize);

// Starts send on the client side.
bytes = bufsize;
status = pj_ioqueue_send(ckey, &write_op, send_buf, &bytes, 0);
if (status != PJ_SUCCESS && bytes != PJ_EPENDING) {
    return -120;
}
if (status == PJ_EPENDING) {
    ++pending_op;
}

// Begin time.
pj_get_timestamp(&t1);

// Reset indicators
callback_read_size = callback_write_size = 0;
callback_read_key = callback_write_key = NULL;
callback_read_op = callback_write_op = NULL;

// Poll the queue until we've got completion event in the server side.
status = 0;
while (pending_op > 0) {
    timeout.sec = 1; timeout.msec = 0;
#ifdef PJ_SYMBIAN
    PJ_UNUSED_ARG(ioque);
    status = pj_symbianos_poll(-1, 1000);
#else
    status = pj_ioqueue_poll(ioque, &timeout);
#endif
    if (status > 0) {
        if (callback_read_size) {
            if (callback_read_size != bufsize)
                return -160;
            if (callback_read_key != skey)
                return -161;
            if (callback_read_op != &read_op)
                return -162;
        }
        if (callback_write_size) {
            if (callback_write_key != ckey)
                return -163;
            if (callback_write_op != &write_op)
                return -164;
        }
        pending_op -= status;
    }
    if (status == 0) {
        PJ_LOG(3,("", "...error: timed out"));
    }
    if (status < 0) {
        return -170;
    }
}

// Pending op is zero.
// Subsequent poll should yield zero too.
timeout.sec = timeout.msec = 0;
#ifdef PJ_SYMBIAN
    status = pj_symbianos_poll(-1, 1);
#else
    status = pj_ioqueue_poll(ioque, &timeout);
#endif
if (status != 0)
    return -173;

// End time.
pj_get_timestamp(&t2);
t_elapsed->u32.lo += (t2.u32.lo - t1.u32.lo);

// Compare recv buffer with send buffer.
if (pj_memcmp(send_buf, recv_buf, bufsize) != 0) {
    return -180;
}

// Success
return 0;

/*
 * Compliance test for success scenario.
 */
static int compliance_test_0(void)
{
    pj_sock_t ssock=-1, csock0=-1, csock1=-1;
    pj_sockaddr_in addr, client_addr, rmt_addr;
    int client_addr_len;
    pj_pool_t *pool = NULL;
    char *send_buf, *recv_buf;
    pj_ioqueue_t *ioque = NULL;
    pj_ioqueue_key_t *skey=NULL, *ckey0=NULL, *ckey1=NULL;
    pj_ioqueue_op_key_t accept_op;
    int bufsize = BUF_MIN_SIZE;
    pj_msize_t status = -1;
    int pending_op = 0;
    pj_timestamp t_elapsed;
    pj_str_t s;
    pj_status_t rc;

    // Create pool.
    pool = pj_pool_create(mem, NULL, POOL_SIZE, 4000, NULL);

    // Allocate buffers for send and receive.
    send_buf = (char*)pj_pool_alloc(pool, bufsize);
    recv_buf = (char*)pj_pool_alloc(pool, bufsize);

    // Create server socket and client socket for connecting
    rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_STREAM(), 0, &ssock);
    if (rc != PJ_SUCCESS) {
        app_perror("...error creating socket", rc);
        status=-1; goto on_error;
    }

    rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_STREAM(), 0, &csock1);
    if (rc != PJ_SUCCESS) {
        app_perror("...error creating socket", rc);
        status=-1; goto on_error;
    }

    // Bind server socket.
    pj_sockaddr_in_init(&addr, 0, 0);
    if ((rc=pj_sock_bind(ssock, &addr, sizeof(addr))) != 0) {
        app_perror("...bind error", rc);
    }
status=-10; goto on_error;
}
// Get server address.
client_addr_len = sizeof(addr);
rc = pj_sock_getsockname(ssock, &addr, &client_addr_len);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_sock_getsockname()", rc);
    status=-15; goto on_error;
}
addr.sin_addr = pj_inet_addr(pj_cstr(&s, "127.0.0.1"));

// Create I/O Queue.
rc = pj_ioqueue_create(pool, PJ_IOQUEUE_MAX_HANDLES, &ioque);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_ioqueue_create()", rc);
    status=-20; goto on_error;
}

// Register server socket and client socket.
rc = pj_ioqueue_register_sock(pool, ioque, ssock, NULL, &test_cb, &skey);
if (rc == PJ_SUCCESS)
    rc = pj_ioqueue_register_sock(pool, ioque, csock1, NULL, &test_cb, &ckey1);
else
    ckey1 = NULL;
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_ioqueue_register_sock()", rc);
    status=-23; goto on_error;
}

// Server socket listen().
if (pj_sock_listen(ssock, 5)) {
    app_perror("...ERROR in pj_sock_listen()", rc);
    status=-25; goto on_error;
}

// Server socket accept().
client_addr_len = sizeof(pj_sockaddr_in);
status = pj_ioqueue_accept(skey, &accept_op, &csock0, &client_addr, &rmt_addr, &client_addr_len);
if (status != PJ_EPENDING) {
    app_perror("...ERROR in pj_ioqueue_accept()", rc);
    status=-30; goto on_error;
}
if (status==PJ_EPENDING) {
    ++pending_op;
}

// Client socket connect().
status = pj_ioqueue_connect(ckey1, &addr, sizeof(addr));
if (status!=PJ_SUCCESS && status != PJ_EPENDING) {
    app_perror("...ERROR in pj_ioqueue_connect()", rc);
    status=-40; goto on_error;
}
if (status==PJ_EPENDING) {
    ++pending_op;
}

// Poll until connected
callback_read_size = callback_write_size = 0;
callback_accept_status = callback_connect_status = -2;
callback_call_count = 0;
callback_read_key = callback_write_key =
callback_accept_key = callback_connect_key = NULL;
callback_accept_op = callback_read_op = callback_write_op = NULL;
while (pending_op) {
    pj_time_val timeout = {1, 0};
#endif PJ_SYMBIAN
    callback_call_count = 0;
    pj_symbianos_poll(-1, 1000);
    status = callback_call_count;
#else
    status = pj_ioqueue_poll(ioque, &timeout);
#endif
    if (status > 0) {
        if (callback_accept_status != -2) {
            if (callback_accept_status != 0) {
                status=-41; goto on_error;
            }
            if (callback_accept_key != skey) {
                status=-42; goto on_error;
            }
            if (callback_accept_op != &accept_op) {
                status=-43; goto on_error;
            }
            callback_accept_status = -2;
        }
        if (callback_connect_status != -2) {
            if (callback_connect_status != 0) {
                status=-50; goto on_error;
            }
            if (callback_connect_key != ckey1) {
                status=-51; goto on_error;
            }
            callback_connect_status = -2;
        }
        if (status > pending_op) {
            PJ_LOG(3,(THIS_FILE,
                "...error: pj_ioqueue_poll() returned %d "
                "(only expecting %d)",
                status, pending_op));
            return -52;
        }
        pending_op -= status;
        if (pending_op == 0) {
            status = 0;
        }
    }
    // There's no pending operation.
    // When we poll the ioqueue, there must not be events.
#ifdef PJ_SYMBIAN
    pj_time_val timeout = {1, 0};
#else
    status = pj_ioqueue_poll(ioque, &timeout);
#endif
    if (status != 0) {
        status=-60; goto on_error;
    }
    // Check accepted socket.
    if (csock0 == PJ_INVALID_SOCKET) {
        status = -69;
    }
app_perror("...accept() error", pj_get_os_error());
goto on_error;
}

// Register newly accepted socket.
rc = pj_ioqueue_register_sock(pool, ioque, csock0, NULL,
   &test_cb, &ckey0);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_ioqueue_register_sock", rc);
    status = -70;
    goto on_error;
}

// Test send and receive.
t_elapsed.u32.lo = 0;
status = send_recv_test(ioque, ckey0, ckey1, send_buf,
    recv_buf, bufsize, &t_elapsed);
if (status != 0) {
    goto on_error;
}

// Success
status = 0;

on_error:
if (skey != NULL)
    pj_ioqueue_unregister(skey);
else if (ssock != PJ_INVALID_SOCKET)
    pj_sock_close(ssock);
if (ckey1 != NULL)
    pj_ioqueue_unregister(ckey1);
else if (csock1 != PJ_INVALID_SOCKET)
    pj_sock_close(csock1);
if (ckey0 != NULL)
    pj_ioqueue_unregister(ckey0);
else if (csock0 != PJ_INVALID_SOCKET)
    pj_sock_close(csock0);
if (ioque != NULL)
    pj_ioqueue_destroy(ioque);
    pj_pool_release(pool);
    return status;

/*
 * Compliance test for failed scenario.
 * In this case, the client connects to a non-existant service.
 */
static int compliance_test_1(void)
{
pj_sock_t csock1 = PJ_INVALID_SOCKET;
pj_sockaddr_in addr;
pj_pool_t *pool = NULL;
pj_ioqueue_t *ioque = NULL;
pj_ioqueue_key_t *ckey1 = NULL;
pj_ssize_t status = -1;
int pending_op = 0;
pj_str_t s;
pj_status_t rc;

// Create pool.
pool = pj_pool_create(mem, NULL, POOL_SIZE, 4000, NULL);
// Create I/O Queue.
rc = pj_ioqueue_create(pool, PJ_IOQUEUE_MAX_HANDLES, &ioque);
if (!ioque) {
    status=-20; goto on_error;
}

// Create client socket
rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_STREAM(), 0, &csock1);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_sock_socket()", rc);
    status=-1; goto on_error;
}

// Register client socket.
rc = pj_ioqueue_register_sock(pool, ioque, csock1, NULL, &test_cb, &ckey1);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_ioqueue_register_sock()", rc);
    status=-23; goto on_error;
}

// Initialize remote address.
pj_sockaddr_in_init(&addr, pj_cstr(&s, "127.0.0.1"), NON_EXISTANT_PORT);

// Client socket connect()
status = pj_ioqueue_connect(ckey1, &addr, sizeof(addr));
if (status==PJ_SUCCESS) {
    // unexpectedly success!
    status = -30;
    goto on_error;
}
if (status != PJ_EPENDING) {
    // success
} else {
    ++pending_op;
}

callback_connect_status = -2;
callback_connect_key = NULL;

// Poll until we've got result
while (pending_op) {
    pj_time_val timeout = {1, 0};
#ifdef PJ_SYMBIAN
    callback_call_count = 0;
    pj_symbianos_poll(-1, 1000);
    status = callback_call_count;
#else
    status = pj_ioqueue_poll(ioque, &timeout);
#endif
    if (status > 0) {
        if (callback_connect_key==ckey1) {
            if (callback_connect_status == 0) {
                // unexpectedly connected!
                status = -50;
                goto on_error;
            }
        }
        if (status > pending_op) {
            PJ_LOG(3,(THIS_FILE, "...error: pj_ioqueue_poll() returned %d "
            "(only expecting %d)", status, pending_op));
            return -552;
        }
    }
}

// other code...

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
pending_op -= status;
if (pending_op == 0) {
    status = 0;
}
}

// There’s no pending operation.
// When we poll the ioqueue, there must not be events.
if (pending_op == 0) {
    pj_time_val timeout = {1, 0};
    #ifdef PJ_Symbian
        status = pj_symbianos_poll(-1, 1000);
    #else
        status = pj_ioqueue_poll(ioque, &timeout);
    #endif
    if (status != 0) {
        status=-60; goto on_error;
    }
}

// Success
status = 0;

on_error:
    if (ckey1 != NULL)
        pj_ioqueue_unregister(ckey1);
    else if (csock1 != PJ_INVALID_SOCKET)
        pj_sock_close(csock1);
    if (ioque != NULL)
        pj_ioqueue_destroy(ioque);
    pj_pool_release(pool);
    return status;

.getServerFunction

static int compliance_test_2(void)
{
    #if defined(PJ_Symbian) && PJ_Symbian!=0
        enum { MAX_PAIR = 1, TEST_LOOP = 2 };
    #else
        enum { MAX_PAIR = 4, TEST_LOOP = 2 };
    #endif

    struct listener
    {
        pj_sock_t sock;
        pj_ioqueue_key_t *key;
        pj_sockaddr_in addr;
        int addr_len;
    } listener;

    struct server
    {
        pj_sock_t sock;
        pj_ioqueue_key_t *key;
        pj_sockaddr_in local_addr;
        pj_sockaddr_in rem_addr;
        int rem_addr_len;
        pj_ioqueue_op_key_t accept_op;
    } server[MAX_PAIR];

    struct client

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen


```c

pj_sock_t sock;
pj_ioqueue_key_t *key;

client[MAX_PAIR];

pj_pool_t *pool = NULL;
char *send_buf, *recv_buf;
pj_ioqueue_t *ioque = NULL;

int i, bufsize = BUF_MIN_SIZE;
pj_ssize_t status;
int test_loop, pending_op = 0;
pj_timestamp t_elapsed;
pj_str_t s;
pj_status_t rc;

listener.sock = PJ_INVALID_SOCKET;
listener.key = NULL;

for (i=0; i<MAX_PAIR; ++i) {
    server[i].sock = PJ_INVALID_SOCKET;
    server[i].key = NULL;
}

for (i=0; i<MAX_PAIR; ++i) {
    client[i].sock = PJ_INVALID_SOCKET;
    client[i].key = NULL;
}

// Create pool.
pool = pj_pool_create(mem, NULL, POOL_SIZE, 4000, NULL);

// Create I/O Queue.
rC = pj_ioqueue_create(pool, PJ_IOQUEUE_MAX_HANDLES, &ioque);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_ioqueue_create()", rc);
    return -10;
}

// Allocate buffers for send and receive.
send_buf = (char*)pj_pool_alloc(pool, bufsize);
recv_buf = (char*)pj_pool_alloc(pool, bufsize);

// Create listener socket
rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_STREAM(), 0, &listener.sock);
if (rc != PJ_SUCCESS) {
    app_perror("...error creating socket", rc);
    status=-20; goto on_error;
}

// Bind listener socket.
pj_sockaddr_in_init(&listener.addr, 0, 0);
if ((rc=pj_sock_bind(listener.sock, &listener.addr, sizeof(listener.addr))) != 0 ) {
    app_perror("...bind error", rc);
    status=-30; goto on_error;
}

// Get listener address.
listener.addr_len = sizeof(listener.addr);
rc = pj_sock_getsockname(listener.sock, &listener.addr, &listener.addr_len);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_sock_getsockname()", rc);
    status=-40; goto on_error;
}

listener.addr.sin_addr = pj_inet_addr(pj_cstr(&s, "127.0.0.1"));
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
// Register listener socket.
rc = pj_ioqueue_register_sock(pool, ioque, listener.sock, NULL, &test_cb, 
&listener.key);
if (rc != PJ_SUCCESS) {
    app_error("...ERROR", rc);
    status=-50; goto on_error;
}

// Listener socket listen().
if (pj_sock_listen(listener.sock, 5)) {
    app_error("...ERROR in pj_sock_listen()", rc);
    status=-60; goto on_error;
}

for (test_loop=0; test_loop < TEST_LOOP; ++test_loop) {
    // Client connect and server accept.
    for (i=0; i<MAX_PAIR; ++i) {
        rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_STREAM(), 0, &client[i].sock);
        if (rc != PJ_SUCCESS) {
            app_error("...error creating socket", rc);
            status=-70; goto on_error;
        }
        rc = pj_ioqueue_register_sock(pool, ioque, client[i].sock, NULL, 
&test_cb, &client[i].key);
        if (rc != PJ_SUCCESS) {
            app_error("...error", rc);
            status=-80; goto on_error;
        }
        // Server socket accept()
        pj_ioqueue_op_key_init(&server[i].accept_op, 
sizeof(server[i].accept_op);
        server[i].rem_addr_len = sizeof(pj_sockaddr_in);
        status = pj_ioqueue_accept(listener.key, &server[i].accept_op, 
&server[i].sock, &server[i].local_addr, 
&server[i].rem_addr, 
&server[i].rem_addr_len);
        if (status!=PJ_SUCCESS && status != PJ_EPENDING) {
            app_error("...ERROR in pj_ioqueue_accept()", rc);
            status=-90; goto on_error;
        }
        if (status==PJ_EPENDING) {
            ++pending_op;
        }
        // Client socket connect()
        status = pj_ioqueue_connect(client[i].key, &listener.addr, 
sizeof(listener.addr));
        if (status!=PJ_SUCCESS && status != PJ_EPENDING) {
            app_error("...ERROR in pj_ioqueue_connect()", rc);
            status=-100; goto on_error;
        }
        if (status==PJ_EPENDING) {
            ++pending_op;
        }
    }
    // Poll until all connected
    while (pending_op) {
        pj_time_val timeout = {1, 0};
    }
}

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
#ifdef PJ_SYMBIAN
    status = pj_symbianos_poll(-1, 1000);
#else
    status = pj_ioqueue_poll(ioque, &timeout);
#endif
if (status > 0) {
    if (status > pending_op) {
        PJ_LOG(3,(THIS_FILE,
            "...error: pj_ioqueue_poll() returned %d "
            "(only expecting %d),
            status, pending_op));
        return -110;
    }
    pending_op -= status;
    if (pending_op == 0) {
        status = 0;
    }
}

// There's no pending operation.  
// When we poll the ioqueue, there must not be events.
if (pending_op == 0) {
    pj_time_val timeout = {1, 0};
    #ifdef PJ_SYMBIAN
        status = pj_symbianos_poll(-1, 1000);
    #else
        status = pj_ioqueue_poll(ioque, &timeout);
    #endif
    if (status != 0) {
        status=-120; goto on_error;
    }
}
for (i=0; i<MAX_PAIR; ++i) {
    // Check server socket.
    if (server[i].sock == PJ_INVALID_SOCKET) {
        status = -130;
        app_perror("...accept() error", pj_get_os_error());
        goto on_error;
    }
    // Check addresses
    if (server[i].local_addr.sin_family != pj_AF_INET() ||
        server[i].local_addr.sin_addr.s_addr == 0 ||
        server[i].local_addr.sin_port == 0)
    {
        app_perror("...ERROR address not set", rc);
        status = -140;
        goto on_error;
    }
    if (server[i].rem_addr.sin_family != pj_AF_INET() ||
        server[i].rem_addr.sin_addr.s_addr == 0 ||
        server[i].rem_addr.sin_port == 0)
    {
        app_perror("...ERROR address not set", rc);
        status = -150;
        goto on_error;
    }
    // Register newly accepted socket.
    rc = pj_ioqueue_register_sock(pool, ioque, server[i].sock, NULL,
        &test_cb, &server[i].key);
}

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_ioqueue_register_sock", rc);
    status = -160;
    goto on_error;
}

// Test send and receive.
t_elapsed.u32.lo = 0;
status = send_recv_test(ioque, server[i].key, client[i].key,
                      send_buf, recv_buf, bufsize, &t_elapsed);
if (status != 0) {
    goto on_error;
}

// Success
status = 0;
for (i=0; i<MAX_PAIR; ++i) {
if (server[i].key != NULL) {
    pj_ioqueue_unregister(server[i].key);
    server[i].key = NULL;
    server[i].sock = PJ_INVALID_SOCKET;
} else if (server[i].sock != PJ_INVALID_SOCKET) {
    pj_sock_close(server[i].sock);
    server[i].sock = PJ_INVALID_SOCKET;
}

if (client[i].key != NULL) {
    pj_ioqueue_unregister(client[i].key);
    client[i].key = NULL;
    client[i].sock = PJ_INVALID_SOCKET;
} else if (client[i].sock != PJ_INVALID_SOCKET) {
    pj_sock_close(client[i].sock);
    client[i].sock = PJ_INVALID_SOCKET;
}
}

status = 0;
on_error:
for (i=0; i<MAX_PAIR; ++i) {
if (server[i].key != NULL) {
    pj_ioqueue_unregister(server[i].key);
    server[i].key = NULL;
    server[i].sock = PJ_INVALID_SOCKET;
} else if (server[i].sock != PJ_INVALID_SOCKET) {
    pj_sock_close(server[i].sock);
    server[i].sock = PJ_INVALID_SOCKET;
}

if (client[i].key != NULL) {
    pj_ioqueue_unregister(client[i].key);
    client[i].key = NULL;
    server[i].sock = PJ_INVALID_SOCKET;
} else if (client[i].sock != PJ_INVALID_SOCKET) {
    pj_sock_close(client[i].sock);
    client[i].sock = PJ_INVALID_SOCKET;
}
}

if (listener.key) {
    pj_ioqueue_unregister(listener.key);
    listener.key = NULL;
} else if (listener.sock != PJ_INVALID_SOCKET) {
    pj_sock_close(listener.sock);
}
if (iQue != NULL)
    pj_ioqueue_destroy(iQue);
pj_pool_release(pool);
return status;

int tcp_ioqueue_test()
{
    int status;

    PJ_LOG(3, (THIS_FILE, "..%s compliance test 0 (success scenario)",
               pj_ioqueue_name()));
    if ((status=compliance_test_0()) != 0) {
        PJ_LOG(1, (THIS_FILE, "....FAILED (status=%d)\n", status));
        return status;
    }

    PJ_LOG(3, (THIS_FILE, "..%s compliance test 1 (failed scenario)",
               pj_ioqueue_name()));
    if ((status=compliance_test_1()) != 0) {
        PJ_LOG(1, (THIS_FILE, "....FAILED (status=%d)\n", status));
        return status;
    }

    PJ_LOG(3, (THIS_FILE, "..%s compliance test 2 (repeated accept)",
               pj_ioqueue_name()));
    if ((status=compliance_test_2()) != 0) {
        PJ_LOG(1, (THIS_FILE, "....FAILED (status=%d)\n", status));
        return status;
    }

    return 0;
}

#endif /* PJ_HAS_TCP */

#else
    /* To prevent warning about "translation unit is empty"
     * when this test is disabled.
     */
    int dummy_uiq_tcp;
#endif /* INCLUDE_TCP_IOQUEUE_TEST */
9.11 Test: I/O Queue (UDP)

This file provides implementation to test the functionality of the I/O queue when UDP socket is used.

This file is `pjlib-test/ioq_udp.c`

```c
/* $Id: ioq_udp.c 1405 2007-07-20 08:08:30Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 */
#include "test.h"
#if INCLUDE_UDP_IOQUEUE_TEST
#include <pjlib.h>
#include <pj/compat/socket.h>
#define THIS_FILE "test_udp"
#define PORT 51233
#define LOOP 2
#define BUF_MIN_SIZE 32
#define BUF_MAX_SIZE 2048
#define SOCK_INACTIVE_MIN (1)
#define SOCK_INACTIVE_MAX (PJ_IOQUEUE_MAX_HANDLES - 2)
#define POOL_SIZE (2*BUF_MAX_SIZE + SOCK_INACTIVE_MAX*128 + 2048)
#undef TRACE_
#define TRACE_(msg) PJ_LOG(3,(THIS_FILE,"....." msg))
#if 0
#define TRACE__(args) PJ_LOG(3,args)
#else
#define TRACE__(args)
#endif
static pj_ssize_t callback_read_size,
callback_write_size,
callback_accept_status,
callback_connect_status;
static pj_ioqueue_key_t *callback_read_key,
*callback_write_key,
*callback_accept_key,
*callback_connect_key;
static pj_ioqueue_op_key_t *callback_read_op,
*callback_write_op,
*callback_accept_op;
static void on_ioqueue_read(pj_ioqueue_key_t *key,
pj_ioqueue_op_key_t *op_key,
pj_ssize_t bytes_read)
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
```c
{  
callback_read_key = key;
callback_read_op = op_key;
callback_read_size = bytes_read;
TRACE__((THIS_FILE, " callback_read_key = %p, bytes=%d",
key, bytes_read));
}

static void on_ioqueue_write(pj_ioqueue_key_t *key,
pj_ioqueue_op_key_t *op_key,
pj_ssize_t bytes_written)
{
callback_write_key = key;
callback_write_op = op_key;
callback_write_size = bytes_written;
}

static void on_ioqueue_accept(pj_ioqueue_key_t *key,
pj_ioqueue_op_key_t *op_key,
pj_sock_t sock, int status)
{
PJ_UNUSED_ARG(sock);
callback_accept_key = key;
callback_accept_op = op_key;
callback_accept_status = status;
}

static void on_ioqueue_connect(pj_ioqueue_key_t *key, int status)
{
callback_connect_key = key;
callback_connect_status = status;
}

static pj_ioqueue_callback test_cb =
{
&on_ioqueue_read,
&on_ioqueue_write,
&on_ioqueue_accept,
&on_ioqueue_connect,
};

#ifdef PJ_WIN32
#define S_ADDR S_un.S_addr
#else
#define S_ADDR s_addr
#endif

/*
compliance_test()
*/

static int compliance_test(void)
{
  pj_sock_t sssock=-1, csock=-1;
pj_sockaddr_in addr, dst_addr;
int addrlen;
pj_pool_t *pool = NULL;
char *send_buf, *recv_buf;
pj_ioqueue_t *ioque = NULL;
pj_ioqueue_key_t *skey = NULL, *ckey = NULL;
pj_ioqueue_op_key_t read_op, write_op;
int bufsize = BUF_MIN_SIZE;
pj_ssize_t bytes, status = -1;
pj_str_t temp;
pj_bool_t send_pending, recv_pending;
pj_status_t rc;
```
9.11 Test: I/O Queue (UDP)

pj_set_os_error(PJ_SUCCESS);

// Create pool.
pool = pj_pool_create(mem, NULL, POOL_SIZE, 4000, NULL);

// Allocate buffers for send and receive.
send_buf = (char*)pj_pool_alloc(pool, bufsize);
recv_buf = (char*)pj_pool_alloc(pool, bufsize);

// Allocate sockets for sending and receiving.
rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &ssock);
if (rc==PJ_SUCCESS)
  rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &csock);
else
  csock = PJ_INVALID_SOCKET;
if (rc != PJ_SUCCESS) {
  app_perror("...ERROR in pj_sock_socket()", rc);
  status=-1; goto on_error;
}

// Bind server socket.
TRACE_("bind socket...");
pj_bzero(&addr, sizeof(addr));
addr.sin_family = pj_AF_INET();
addr.sin_port = pj_htons(PORT);
if (pj_sock_bind(ssock, &addr, sizeof(addr))) {
  status=-10; goto on_error;
}

// Create I/O Queue.
TRACE_("create ioqueue...");
rc = pj_ioqueue_create(pool, PJ_IOQUEUE_MAX_HANDLES, &ioque);
if (rc != PJ_SUCCESS) {
  app_perror("...error(10): ioqueue_register error", rc);
  status=-20; goto on_error;
}

// Register server and client socket.
// We put this after inactivity socket, hopefully this can represent the
// worst waiting time.
TRACE_("registering first sockets...");
r = pj_ioqueue_register_sock(pool, ioque, ssock, NULL,
  &test_cb, &skey);
if (r != PJ_SUCCESS) {
  app_perror("...error(10): ioqueue_register error", r);
  status=-25; goto on_error;
}

TRACE_("registering second sockets...");
r = pj_ioqueue_register_sock(pool, ioque, csock, NULL,
  &test_cb, &ckey);
if (r != PJ_SUCCESS) {
  app_perror("...error(11): ioqueue_register error", r);
  status=-26; goto on_error;
}

// Randomize send_buf.
pj_create_random_string(send_buf, bufsize);

// Register reading from ioqueue.
TRACE_("start recvfrom...");
pj_bzero(&addr, sizeof(addr));
addrlen = sizeof(addr);
bytes = bufsize;
r = pj_ioqueue_recvfrom(skey, &read_op, recv_buf, &bytes, 0,
  &addr, &addrlen);
if (r != PJ_SUCCESS && r != PJ_EPENDING) {
app_perror("...error: pj_ioqueue_recvfrom", rc);
status=-28; goto on_error;
} else if (rc == PJ_EPENDING) {
recv_pending = 1;
PJ_LOG(3, (THIS_FILE,
"......ok: recvfrom returned pending"));
} else {
PJ_LOG(3, (THIS_FILE,
"......error: recvfrom returned immediate ok!");
status=-29; goto on_error;
}

// Set destination address to send the packet.
TRACE_("set destination address...");
temp = pj_str("127.0.0.1");
if ( (rc= pj_sockaddr_in_init(&dst_addr, &temp, PORT) != 0) {
app_perror("...error: unable to resolve 127.0.0.1", rc);
status=-290; goto on_error;
}

// Write must return the number of bytes.
TRACE_("start sendto...");
bytes = bufsize;
rc = pj_ioqueue_sendto(ckey, &write_op, send_buf, &bytes, 0, &dst_addr,
sizeof(dst_addr));
if (rc != PJ_SUCCESS && rc != PJ_EPENDING) {
app_perror("...error: pj_ioqueue_sendto", rc);
status=-30; goto on_error;
} else if (rc == PJ_EPENDING) {
send_pending = 1;
PJ_LOG(3, (THIS_FILE,
"......ok: sendto returned pending");
} else {
send_pending = 0;
PJ_LOG(3, (THIS_FILE,
"......ok: sendto returned immediate success");
}

// reset callback variables.
callback_read_size = callback_write_size = 0;
callback_accept_status = callback_connect_status = -2;
callback_read_key = callback_write_key =
callback_accept_key = callback_connect_key = NULL;
callback_read_op = callback_write_op = NULL;

// Poll if pending.
while (send_pending || recv_pending) {
int rc;
pj_time_val timeout = { 5, 0);
TRACE_("poll...");
#ifdef PJ_SYMBIAN
rc = pj_symbianos_poll(-1, 5000); 
#else
rc = pj_ioqueue_poll(ioque, &timeout);
#endif
if (rc == 0) { 
PJ_LOG(1,(THIS_FILE, "...ERROR: timed out...");
status=-45; goto on_error;
} else if (rc < 0) { 
app_perror("...ERROR in ioqueue_poll()", -rc);
status=-50; goto on_error;
}
if (callback_read_key != NULL) {
if (callback_read_size != bufsize) {
app_perror("...error: unable to resolve 127.0.0.1", rc);
status=-290; goto on_error;
} else if (rc == PJ_EPENDING) {
recv_pending = 1;
PJ_LOG(3, (THIS_FILE,
"......ok: recvfrom returned pending");
} else {
recv_pending = 0;
PJ_LOG(3, (THIS_FILE,
"......ok: recvfrom returned immediate ok!");

// Set destination address to send the packet.
TRACE_("set destination address...");
temp = pj_str("127.0.0.1");
if ( (rc= pj_sockaddr_in_init(&dst_addr, &temp, PORT) != 0) {
app_perror("...error: unable to resolve 127.0.0.1", rc);
status=-290; goto on_error;
}

// Write must return the number of bytes.
TRACE_("start sendto...");
bytes = bufsize;
rc = pj_ioqueue_sendto(ckey, &write_op, send_buf, &bytes, 0, &dst_addr,
sizeof(dst_addr));
if (rc != PJ_SUCCESS && rc != PJ_EPENDING) {
app_perror("...error: pj_ioqueue_sendto", rc);
status=-30; goto on_error;
} else if (rc == PJ_EPENDING) {
send_pending = 1;
PJ_LOG(3, (THIS_FILE,
"......ok: sendto returned pending");
} else {
send_pending = 0;
PJ_LOG(3, (THIS_FILE,
"......ok: sendto returned immediate success");
}

// reset callback variables.
callback_read_size = callback_write_size = 0;
callback_accept_status = callback_connect_status = -2;
callback_read_key = callback_write_key =
callback_accept_key = callback_connect_key = NULL;
callback_read_op = callback_write_op = NULL;

// Poll if pending.
while (send_pending || recv_pending) {
int rc;
pj_time_val timeout = { 5, 0 );
TRACE_("poll...");
#ifdef PJ_SYMBIAN
rc = pj_symbianos_poll(-1, 5000); 
#else
rc = pj_ioqueue_poll(ioque, &timeout);
#endif
if (rc == 0) { 
PJ_LOG(1,(THIS_FILE, "...ERROR: timed out...");
status=-45; goto on_error;
} else if (rc < 0) { 
app_perror("...ERROR in ioqueue_poll()", -rc);
status=-50; goto on_error;
}
if (callback_read_key != NULL) {
if (callback_read_size != bufsize) {
status=-61; goto on_error;
}
if (callback_read_key != skey) {
    status=-65; goto on_error;
}
if (callback_read_op != &read_op) {
    status=-66; goto on_error;
}
if (pj_memcmp(send_buf, recv_buf, bufsize) != 0) {
    status=-67; goto on_error;
}
if (addrlen != sizeof(pj_sockaddr_in)) {
    status=-68; goto on_error;
}
if (addr.sin_family != pj_AF_INET()) {
    status=-69; goto on_error;
}
recv_pending = 0;
}
if (callback_write_key != NULL) {
if (callback_write_size != bufsize) {
    status=-73; goto on_error;
}
if (callback_write_key != ckey) {
    status=-75; goto on_error;
}
if (callback_write_op != &write_op) {
    status=-76; goto on_error;
}
send_pending = 0;
}

// Success
status = 0;

on_error:
    if (skey)
        pj_ioqueue_unregister(skey);
    else if (ssock != -1)
        pj_sock_close(ssock);
    if (ckey)
        pj_ioqueue_unregister(ckey);
    else if (csock != -1)
        pj_sock_close(csock);
    if (ioque != NULL)
        pj_ioqueue_destroy(ioque);
    pj_pool_release(pool);
    return status;

static void on_read_complete(pj_ioqueue_key_t *key, 
pj_ioqueue_op_key_t *op_key, 
pj_ssize_t bytes_read)
{
    unsigned *p_packet_cnt = (unsigned*) pj_ioqueue_get_user_data(key);
    PJ_UNUSED_ARG(op_key);

PJ_UNUSED_ARG(bytes_read);

(*p_packet_cnt)++;
}

/*
 * unregister_test()
 * Check if callback is still called after socket has been unregistered or
 * closed.
 */
static int unregister_test(void)
{
    enum { RPORT = 50000, SPORT = 50001 }
    pj_pool_t *pool;
    pj_ioqueue_t *ioqueue;
    pj_sock_t ssock;
    pj_sock_t rsock;
    int addrlen;
    pj_sockaddr_in addr;
    pj_ioqueue_key_t *key;
    pj_ioqueue_op_key_t opkey;
    pj_ioqueue_callback cb;
    unsigned packet_cnt;
    char sendbuf[10], recvbuf[10];
    pj_ssize_t bytes;
    pj_time_val timeout;
    pj_status_t status;

    pool = pj_pool_create(mem, "test", 4000, 4000, NULL);
    if (!pool) {
        app_perror("Unable to create pool", PJ_ENOMEM);
        return -100;
    }

    status = pj_ioqueue_create(pool, 16, &ioqueue);
    if (status != PJ_SUCCESS) {
        app_perror("Error creating ioqueue", status);
        return -110;
    }

    /* Create sender socket */
    status = app_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, SPORT, &ssock);
    if (status != PJ_SUCCESS) {
        app_perror("Error initializing socket", status);
        return -120;
    }

    /* Create receiver socket. */
    status = app_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, RPORT, &rsock);
    if (status != PJ_SUCCESS) {
        app_perror("Error initializing socket", status);
        return -130;
    }

    /* Register rsock to ioqueue. */
    pj_bzero(&cb, sizeof(cb));
    cb.on_read_complete = &on_read_complete;
    packet_cnt = 0;
    status = pj_ioqueue_register_sock(pool, ioqueue, rsock, &packet_cnt,
        &cb, &key);
    if (status != PJ_SUCCESS) {
        app_perror("Error registering to ioqueue", status);
        return -140;
    }

    /* Init operation key. */
    pj_ioqueue_op_key_init(&opkey, sizeof(opkey));
}
/* Start reading. */
bytes = sizeof(recvbuf);
status = pj_ioqueue_recv( key, &opkey, recvbuf, &bytes, 0);
if (status != PJ_EPENDING) {
    app_perror("Expecting PJ_EPENDING, but got this", status);
    return -150;
}

/* Init destination address. */
addrlen = sizeof(addr);
status = pj_sock_getsockname(rsock, &addr, &addrlen);
if (status != PJ_SUCCESS) {
    app_perror("getsockname error", status);
    return -160;
}

/* Override address with 127.0.0.1, since getsockname will return *
 * zero in the address field. */
addr.sin_addr = pj_inet_addr2("127.0.0.1");

/* Init buffer to send */
pj_ansi_strcpy(sendbuf, "Hello0123");

/* Send one packet. */
bytes = sizeof(sendbuf);
status = pj_sock_sendto(ssock, sendbuf, &bytes, 0,
    &addr, sizeof(addr));
if (status != PJ_SUCCESS) {
    app_perror("sendto error", status);
    return -170;
}

/* Check if packet is received. */
timeout.sec = 1; timeout.msec = 0;
#ifdef PJ_SYMBIAN
    pj_symbianos_poll(-1, 1000);
#else
    pj_ioqueue_poll(ioqueue, &timeout);
#endif
if (packet_cnt != 1) {
    return -180;
}

/* Just to make sure things are settled. */
pj_thread_sleep(100);

/* Start reading again. */
bytes = sizeof(recvbuf);
status = pj_ioqueue_recv( key, &opkey, recvbuf, &bytes, 0);
if (status != PJ_EPENDING) {
    app_perror("Expecting PJ_EPENDING, but got this", status);
    return -190;
}

/* Reset packet counter */
packet_cnt = 0;

/* Send one packet. */
bytes = sizeof(sendbuf);
status = pj_sock_sendto(ssock, sendbuf, &bytes, 0,
    &addr, sizeof(addr));
if (status != PJ_SUCCESS) {
app_perror("sendto error", status);
return -200;
}

/* Now unregister and close socket. */
pj_ioqueue_unregister(key);

/* Poll iqueue. */
#ifndef PJ_SYMBIAN
pj_symbianos_poll(-1, 1000);
#else
timeout.sec = 1; timeout.msec = 0;
pj_ioqueue_poll(ioqueue, &timeout);
#endif

/* Must NOT receive any packets after socket is closed! */
if (packet_cnt > 0) {
    PJ_LOG(3, (THIS_FILE, "....errror: not expecting to receive packet "
        "after socket has been closed");
    return -210;
}

/* Success */
pj_sock_close(ssock);
pj_ioqueue_destroy(ioqueue);
pj_pool_release(pool);
return 0;


*/
/* Testing with many handles. */
/* This will just test registering PJ_IOQUEUE_MAX_HANDLES count */
/* of sockets to the iqueue. */
*/
static int many_handles_test(void)
{
    enum { MAX = PJ_IOQUEUE_MAX_HANDLES };
    pj_pool_t *pool;
    pj_ioqueue_t *ioqueue;
    pj_sock_t *sock;
    pj_ioqueue_key_t **key;
    pj_status_t rc;
    int count, i; /* must be signed */
    PJ_LOG(3, (THIS_FILE, "...testing with so many handles");)

    pool = pj_pool_create(mem, NULL, 4000, 4000, NULL);
    if (!pool)
        return PJ_ENOMEM;

    key = (pj_ioqueue_key_t**) pj_pool_alloc(pool, MAX*sizeof(pj_ioqueue_key_t*));
    sock = (pj_sock_t*) pj_pool_alloc(pool, MAX*sizeof(pj_sock_t));

    /* Create IQueue */
    rc = pj_ioqueue_create(pool, MAX, &ioqueue);
    if (rc != PJ_SUCCESS || ioqueue == NULL) {
        app_perror("...error in pj_ioqueue_create", rc);
        return -10;
    }

    /* Register as many sockets. */
    for (count=0; count<MAX; ++count) {
        sock[count] = PJ_INVALID_SOCKET;
        }
9.11 Test: I/O Queue (UDP)

```c
rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &sock[count]);
if (rc != PJ_SUCCESS || sock[count] == PJ_INVALID_SOCKET) {
PJ_LOG(3,(THIS_FILE, "....unable to create %d-th socket, rc=%d", count, rc));
    break;
}
key[count] = NULL;
rc = pj_ioqueue_register_sock(pool, ioqueue, sock[count],
    NULL, &test_cb, &key[count]);
if (rc != PJ_SUCCESS || key[count] == NULL) {
PJ_LOG(3,(THIS_FILE, "....unable to register %d-th socket, rc=%d", count, rc));
    return -30;
}

/* Test complete. */
/* Now deregister and close all handles. */

/* NOTE for RTEMS:
 * It seems that the order of close(sock) is pretty important here.
 * If we close the sockets with the same order as when they were created,
 * RTEMS doesn't seem to reuse the sockets, thus next socket created
 * will have descriptor higher than the last socket created.
 * If we close the sockets in the reverse order, then the descriptor will
 * get reused.
 * This used to cause problem with select ioqueue, since the ioqueue
 * always gives FD_SETSIZE for the first select() argument. This ioqueue
 * behavior can be changed with setting PJ_SELECT_NEEDS_NFDS macro.
 */
for (i=count-1; i>=0; --i) {
    rc = pj_ioqueue_unregister(key[i]);
    if (rc != PJ_SUCCESS) {
        app_perror("...error in pj_ioqueue_unregister", rc);
    }
}
rc = pj_ioqueue_destroy(ioqueue);
if (rc != PJ_SUCCESS) {
    app_perror("...error in pj_ioqueue_destroy", rc);
}
pj_pool_release(pool);
PJ_LOG(3,(THIS_FILE,"....many_handles_test() ok");

return 0;
}

/* Multi-operation test. */
/*
 * Benchmarking IOQueue
 */
static int bench_test(int bufsize, int inactive_sock_count)
{
    pj_sock_t ssock=-1, csock=-1;
    pj_sockaddr_in addr;
    pj_pool_t *pool = NULL;
    pj_sock_t *inactive_sock=NULL;
    pj_ioqueue_op_key_t *inactive_read_op;
    char *send_buf, *recv_buf;
    pj_ioqueue_t *ioque = NULL;
    pj_ioqueue_key_t *skey, *ckey, *keys[SOCK_INACTIVE_MAX+2];
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
pj_timestamp t1, t2, t_elapsed;
int rc=0, i; /* i must be signed */
pj_str_t temp;
char errbuf[PJ_ERR_MSG_SIZE];

TRACE__((THIS_FILE, " bench test \d", inactive_sock_count));

// Create pool.
pool = pj_pool_create(mem, NULL, POOL_SIZE, 4000, NULL);

// Allocate buffers for send and receive.
send_buf = (char*)pj_pool_alloc(pool, bufsize);
recv_buf = (char*)pj_pool_alloc(pool, bufsize);

// Allocate sockets for sending and receiving.
rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &ssock);
if (rc == PJ_SUCCESS) {
    rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &csock);
} else
    csock = PJ_INVALID_SOCKET;
if (rc != PJ_SUCCESS) {
    app_perror("...error: pj_sock_socket()", rc);
    goto on_error;
}

// Bind server socket.
pj_bzero(&addr, sizeof(addr));
addr.sin_family = pj_AF_INET();
addr.sin_port = pj_htons(PORT);
if (pj_sock_bind(ssock, &addr, sizeof(addr)))
    goto on_error;
pj_assert(inactive_sock_count+2 <= PJ_IOQUEUE_MAX_HANDLES);

// Create I/O Queue.
rc = pj_ioqueue_create(pool, PJ_IOQUEUE_MAX_HANDLES, &ioque);
if (rc != PJ_SUCCESS) {
    app_perror("...error: pj_ioqueue_create()", rc);
    goto on_error;
}

// Allocate inactive sockets, and bind them to some arbitrary address.
// Then register them to the I/O queue, and start a read operation.
inactive_sock = (pj_sock_t*)pj_pool_alloc(pool,
inactive_sock_count*sizeof(pj_sock_t));
inactive_read_op = (pj_ioqueue_op_key_t*)pj_pool_alloc(pool,
inactive_sock_count*sizeof(pj_ioqueue_op_key_t));
pj_bzero(&addr, sizeof(addr));
addr.sin_family = pj_AF_INET();
for (i=0; i<inactive_sock_count; ++i) {
    pj_size_t bytes;

    rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &inactive_sock[i]);
    if (rc != PJ_SUCCESS || inactive_sock[i] < 0) {
        app_perror("...error: pj_sock_socket()", rc);
        goto on_error;
    }
    if ((rc=pj_sock_bind(inactive_sock[i], &addr, sizeof(addr))) != 0) {
        pj_sock_close(inactive_sock[i]);
inactive_sock[i] = PJ_INVALID_SOCKET;
        app_perror("...error: pj_sock_bind()", rc);
        goto on_error;
    }
    rc = pj_ioqueue_register_sock(pool, ioque, inactive_sock[i],
        PJ_INVALID_SOCKET, &test_cb, &keys[i]);
    if (rc != PJ_SUCCESS) {
        pj_sock_close(inactive_sock[i]);
        goto on_error;
    }
}
inactive_sock[i] = PJ_INVALID_SOCKET;
app_perror("...error(1): pj_ioqueue_register_sock()", rc);
PJ_LOG(3,(THIS_FILE, "...i=%d", i));
goto on_error;
}
bytes = bufsize;
rc = pj_ioqueue_recv(keys[i], &inactive_read_op[i], recv_buf, &bytes, 0);
if (rc != PJ_EPENDING) {
pj_sock_close(inactive_sock[i]);
inactive_sock[i] = PJ_INVALID_SOCKET;
app_perror("...error: pj_ioqueue_read()", rc);
goto on_error;
}
}

// Register server and client socket.
// We put this after inactivity socket, hopefully this can represent the
// worst waiting time.
rc = pj_ioqueue_register_sock(pool, ioque, ssock, NULL,
&test_cb, &skey);
if (rc != PJ_SUCCESS) {
app_perror("...error(2): pj_ioqueue_register_sock()", rc);
goto on_error;
}
rc = pj_ioqueue_register_sock(pool, ioque, csock, NULL,
&test_cb, &ckey);
if (rc != PJ_SUCCESS) {
app_perror("...error(3): pj_ioqueue_register_sock()", rc);
goto on_error;
}

// Set destination address to send the packet.
pj_sockaddr_in_init(&addr, pj_cstr(&temp, "127.0.0.1"), PORT);

// Test loop.
t_elapsed.u64 = 0;
for (i=0; i<LOOP; ++i) {
pj ssize_t bytes;
pj_ioqueue_op_key_t read_op, write_op;

// Randomize send buffer.
pj_create_random_string(send_buf, bufsize);

// Start reading on the server side.
bytes = bufsize;
rc = pj_ioqueue_recv(skey, &read_op, recv_buf, &bytes, 0);
if (rc != PJ_EPENDING) {
app_perror("...error: pj_ioqueue_read()", rc);
break;
}

// Starts send on the client side.
bytes = bufsize;
rc = pj_ioqueue_sendto(ckey, &write_op, send_buf, &bytes, 0,
&addr, sizeof(addr));
if (rc != PJ_SUCCESS && rc != PJ_EPENDING) {
app_perror("...error: pj_ioqueue_write()", rc);
break;
}
if (rc == PJ_SUCCESS) {
if (bytes < 0) {
app_perror("...error: pj_ioqueue_sendto()", -bytes);
break;
}
// Begin time.
pj_get_timestamp(&t1);

// Poll the queue until we’ve got completion event in the server side.
callback_read_key = NULL;
callback_read_size = 0;
TRACE__((THIS_FILE, " waiting for key = %p", skey));
do {
    pj_time_val timeout = { 1, 0 };
    #ifdef PJ_SYMBIAN
        rc = pj_symbianos_poll(-1, 1000);
    #else
        rc = pj_ioqueue_poll(ioque, &timeout);  
    #endif 
    TRACE__((THIS_FILE, " poll rc=%d", rc));
} while (rc >= 0 && callback_read_key != skey);

// End time.
pj_get_timestamp(&t2);
t_elapsed.u64 += (t2.u64 - t1.u64);

if (rc < 0) {
    app_perror(" error: pj_ioqueue_poll", -rc);
    break;
}

// Compare recv buffer with send buffer.
if (callback_read_size != bufsize
    || pj_memcmp(send_buf, recv_buf, bufsize))
    {  
        rc = -10;
        PJ_LOG(3,(THIS_FILE, " error: size/buffer mismatch");
        break;
    }

// Poll until all events are exhausted, before we start the next loop.
do {
    #ifdef PJ_SYMBIAN
        rc = pj_symbianos_poll(-1, 100);
    #else
        rc = pj_ioqueue_poll(ioque, &timeout);
    #endif
   }while (rc>0);

rc = 0;

// Print results
if (rc == 0) {
    pj_timestamp tzero;
pj_uint32_t usec_delay;
    tzero.u32.hi = tzero.u32.lo = 0;
    usec_delay = pj_elapsed_usec( &tzero, &t_elapsed);
    PJ_LOG(3, (THIS_FILE, "...%10d %15d % 9d",
        bufsize, inactive_sock_count, usec_delay));
} else {
    PJ_LOG(2, (THIS_FILE, ".ERROR rc=%d (buf:%d, fds:%d)",
        rc, bufsize, inactive_sock_count+2));
}

// Cleaning up.
for (i=inactive_sock_count-1; i>=0; --i) {
    pj_ioqueue_unregister(keys[i]);
9.11 Test: I/O Queue (UDP)

```
pj_ioqueue_unregister(skey);
pj_ioqueue_unregister(ckey);
pj_ioqueue_destroy(ioque);
pj_pool_release(pool);
return rc;

on_error:
PJ_LOG(1, (THIS_FILE, "...ERROR: %s",
    pj_strerror(pj_get_netos_error(), errbuf, sizeof(errbuf))));
if (ssock)    pj_sock_close(ssock);
if (csock)    pj_sock_close(csock);
for (i=0; i<inactive_sock_count && inactive_sock &&
    inactive_sock[i]!=PJ_INVALID_SOCKET; ++i) {
    pj_sock_close(inactive_sock[i]);
}    if (ioque != NULL)    pj_ioqueue_destroy(ioque);
pj_pool_release(pool);
return -1;

int udp_ioqueue_test()
{
    int status;
    int bufsize, sock_count;
    //goto pass1;
    PJ_LOG(3, (THIS_FILE, "...compliance test (%s)", pj_ioqueue_name()));
    if ((status=compliance_test()) != 0) {
        return status;
    }
    PJ_LOG(3, (THIS_FILE, "....compliance test ok"));

    PJ_LOG(3, (THIS_FILE, "...unregister test (%s)", pj_ioqueue_name()));
    if ((status=unregister_test()) != 0) {
        return status;
    }
    PJ_LOG(3, (THIS_FILE, "....unregister test ok"));

    if ((status=many_handles_test()) != 0) {
        return status;
    }
    //return 0;
    PJ_LOG(4, (THIS_FILE, "...benchmarking different buffer size:"));
    PJ_LOG(4, (THIS_FILE, "... note: buf=bytes sent, fds=# of fds, "
        "elapsed=in timer ticks"));
    //pass1:
    PJ_LOG(3, (THIS_FILE, "...Benchmarking poll times for %s:", pj_ioqueue_name()));
    PJ_LOG(3, (THIS_FILE, "...====================================="));
    PJ_LOG(3, (THIS_FILE, "...Buf.size #inactive-socks Time/poll"));
    PJ_LOG(3, (THIS_FILE, "... (bytes) (nanosec)"));
    PJ_LOG(3, (THIS_FILE, "...====================================="));
    //goto pass2;
```
for (bufsize=BUF_MIN_SIZE; bufsize <= BUF_MAX_SIZE; bufsize *= 2) {
    if ((status=bench_test(bufsize, SOCK_INACTIVE_MIN)) != 0)
        return status;
}

//pass2:
bufsize = 512;
for (sock_count=SOCK_INACTIVE_MIN+2;
    sock_count<=SOCK_INACTIVE_MAX+2;
    sock_count *= 2)
{
    //PJ_LOG(3,(THIS_FILE, "...testing with %d fds", sock_count));
    if ((status=bench_test(bufsize, sock_count-2)) != 0)
        return status;
}
return 0;

#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_uiq_udp;
#endif /* INCLUDE_UDP_IOQUEUE_TEST */
9.12 Test: Linked List

This file provides implementation of list_test(). It tests the functionality of the linked-list API.

9.12.1 Scope of the Test

API tested:

- pj_list_init()
- pj_list_insert_before()
- pj_list_insert_after()
- pj_list_merge_last()
- pj_list_empty()
- pj_list_insert_nodes_before()
- pj_list_erase()
- pj_list_find_node()
- pj_list_search()

This file is pjlib-test/list.c

```
/* $Id: list.c 974 2007-02-19 01:13:53Z bennyip $ */
/*
* Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
#include "test.h"
#if INCLUDE_LIST_TEST
#include <pjlib.h>

typedef struct list_node
{
    PJ_DECL_LIST_MEMBER(struct list_node);
    int value;
} list_node;

static int compare_node(void *value, const pj_list_type *nd)
{
    list_node *node = (list_node*)nd;
    return ((long)value == node->value) ? 0 : -1;
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
```c
#define PJ_SIGNED_ARRAY_SIZE(a) ((int)PJ_ARRAY_SIZE(a))

int list_test()
{
    list_node nodes[4]; // must be even number of nodes
    list_node list;
    list_node list2;
    list_node *p;
    int i; // don't change to unsigned!

    // Test insert_before().
    //
    list.value = (unsigned)-1;
    pj_list_init(&list);
    for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
        nodes[i].value = i;
        pj_list_insert_before(&list, &nodes[i]);
    }
    // check.
    for (i=0, p=list.next; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i, p=p->next) {
        pj_assert(p->value == i);
        if (p->value != i) {
            return -1;
        }
    }

    // Test insert_after()
    //
    pj_list_init(&list);
    for (i=PJ_SIGNED_ARRAY_SIZE(nodes)-1; i>=0; --i) {
        pj_list_insert_after(&list, &nodes[i]);
    }
    // check.
    for (i=0, p=list.next; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i, p=p->next) {
        pj_assert(p->value == i);
        if (p->value != i) {
            return -1;
        }
    }

    // Test merge_last()
    //
    pj_list_init(&list);
    pj_list_init(&list2);
    for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes)/2; ++i) {
        pj_list_insert_before(&list, &nodes[i]);
    }
    for (i=PJ_SIGNED_ARRAY_SIZE(nodes)/2; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
        pj_list_insert_before(&list2, &nodes[i]);
    }
    // merge
    pj_list_merge_last(&list, &list2);
    // check.
    for (i=0, p=list.next; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i, p=p->next) {
        pj_assert(p->value == i);
        if (p->value != i) {
            return -1;
        }
    }
    // check list is empty
    pj_assert( pj_list_empty(&list2) );
}
```

if (!pj_list_empty(&list2)) {
    return -1;
}

// Check merge_first()
//
pj_list_init(&list);
pj_list_init(&list2);
for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes)/2; ++i) {
    pj_list_insert_before(&list, &nodes[i]);
}
for (i=PJ_SIGNED_ARRAY_SIZE(nodes)/2; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
    pj_list_insert_before(&list2, &nodes[i]);
}
// merge
pj_list_merge_first(&list2, &list);
// check (list2).
for (i=0, p=list2.next; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i, p=p->next) {
    pj_assert(p->value == i);
    if (p->value != i) {
        return -1;
    }
}
// check list is empty
pj_assert( pj_list_empty(&list) );
if (!pj_list_empty(&list)) {
    return -1;
}

// Test insert_nodes_before()
//
// init list
pj_list_init(&list);
for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes)/2; ++i) {
    pj_list_insert_before(&list, &nodes[i]);
}
// chain remaining nodes
pj_list_init(&nodes[PJ_SIGNED_ARRAY_SIZE(nodes)/2]);
for (i=PJ_SIGNED_ARRAY_SIZE(nodes)/2+1; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
    pj_list_insert_before(&nodes[PJ_SIGNED_ARRAY_SIZE(nodes)/2], &nodes[i]);
}
// insert nodes
pj_list_insert_nodes_before(&list, &nodes[PJ_SIGNED_ARRAY_SIZE(nodes)/2]);
// check
for (i=0, p=list.next; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i, p=p->next) {
    pj_assert(p->value == i);
    if (p->value != i) {
        return -1;
    }
}

// erase test.
pj_list_init(&list);
for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
    int j;
    nodes[i].value = i;
    pj_list_insert_before(&list, &nodes[i]);
}
for (i=PJ_SIGNED_ARRAY_SIZE(nodes)-1; i>=0; --i) {
    pj_list_erase(&nodes[i]);
    for (j=0, p=list.next; j<i; ++j, p=p->next) {
        pj_assert(p->value == j);
        if (p->value != j) {
            return -1;
        }
    }
}
// find and search
pj_list_init(&list);
for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
    nodes[i].value = i;
    pj_list_insert_before(&list, &nodes[i]);
}
for (i=0; i<PJ_SIGNED_ARRAY_SIZE(nodes); ++i) {
    p = (list_node*) pj_list_find_node(&list, &nodes[i]);
    pj_assert( p == &nodes[i] );
    if (p != &nodes[i]) {
        return -1;
    }
    p = (list_node*) pj_list_search(&list, (void*)(long)i, &compare_node);
    pj_assert( p == &nodes[i] );
    if (p != &nodes[i]) {
        return -1;
    }
}
return 0;

#endif /* INCLUDE_LIST_TEST */

/* To prevent warning about "translation unit is empty"
   * when this test is disabled.
   */
int dummy_list_test;
#endif /* INCLUDE_LIST_TEST */
This file provides implementation of pool_test(). It tests the functionality of the memory pool.

This file is pjlib-test/pool.c

```c
/* $Id: pool.c 974 2007-02-19 01:13:53Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include <pj/pool.h>
#include <pj/pool_buf.h>
#include <pj/rand.h>
#include <pj/log.h>
#include <pj/except.h>
#include "test.h"

#if INCLUDE_POOL_TEST
#define SIZE 4096

/* Normally we should throw exception when memory alloc fails.
   Here we do nothing so that the flow will go back to original caller,
   which will test the result using NULL comparison. Normally caller will
   catch the exception instead of checking for NULLs.
*/
static void null_callback(pj_pool_t *pool, pj_size_t size)
{
    PJ_UNUSED_ARG(pool);
    PJ_UNUSED_ARG(size);
}
#define GET_FREE(p) [(pj_pool_t *pool, pj_size_t size)]
{
    PJ_UNUSED_ARG(pool);
    PJ_UNUSED_ARG(size);
}

#if INCLUDE_POOL_TEST
#define GET_FREE(p) [(pj_pool_get_capacity(p) - pj_pool_get_used_size(p))]

/* Test that the capacity and used size reported by the pool is correct.
*/
static int capacity_test(void)
{
    pj_pool_t *pool = pj_pool_create(mem, NULL, SIZE, 0, &null_callback);
    pj_size_t freesize;
    PJ_LOG(3,("test", "...capacity_test()");
    if (!pool)
        return -200;
    freesize = GET_FREE(pool);
    if (pj_pool_alloc(pool, freesize) == NULL) {
        PJ_LOG(3,("test", "...error: wrong freesize %u reported", freesize));
        pj_pool_release(pool);
        return -210;
    }
    PJ_LOG(3,("test", "...success: got own %u", freesize));
    pj_pool_release(pool);
    return 0;
}
#endif
```
/* Test function to drain the pool’s space. */
static int drain_test(pj_size_t size, pj_size_t increment)
{
    pj_pool_t *pool = pj_pool_create(mem, NULL, size, increment,
        &null_callback);
    pj_size_t freesize;
    void *p;
    int status = 0;
    PJ_LOG(3,("test", "...drain_test(%d,%d)", size, increment));
    if (!pool)
        return -10;
    /* Get free size */
    freesize = GET_FREE(pool);
    if (freesize < 1) {
        status=-15;
        goto on_error;
    }
    /* Drain the pool until there’s nothing left. */
    while (freesize > 0) {
        int size;
        if (freesize > 255)
            size = ({pj_rand() & 0x000000FF} + 4) & ~0x03L;
        else
            size = freesize;
        p = pj_pool_alloc(pool, size);
        if (!p) {
            status=-20; goto on_error;
        }
        freesize -= size;
    }
    /* Check that capacity is zero. */
    if (GET_FREE(pool) != 0) {
        PJ_LOG(3,("test", "....error: returned free=%u (expecting 0)",
            GET_FREE(pool)));
        status=-30; goto on_error;
    }
    /* Try to allocate once more */
    p = pj_pool_alloc(pool, 257);
    if (!p) {
        status=-40; goto on_error;
    }
    /* Check that capacity is NOT zero. */
    if (GET_FREE(pool) == 0) {
        status=-50; goto on_error;
    }
    on_error:
    pj_pool_release(pool);
    return status;
9.13 Test: Pool

} /* Test the buffer based pool */
static int pool_buf_test(void)
{
    enum { STATIC_BUF_SIZE = 40 }
        /* 16 is the internal struct in pool_buf */
    static char buf[ STATIC_BUF_SIZE + sizeof(pj_pool_t) +
        sizeof(pj_pool_block) + 16];
    pj_pool_t *pool;
    void *p;
    PJ_USE_EXCEPTION;
    PJ_LOG(3, ("test", ".pool_buf test");

    pool = pj_pool_create_on_buf("no name", buf, sizeof(buf));
    if (!pool)
        return -70;

    /* Drain the pool */
    PJ_TRY {
        if ((p = pj_pool_alloc(pool, STATIC_BUF_SIZE/2)) == NULL)
            return -75;
        if ((p = pj_pool_alloc(pool, STATIC_BUF_SIZE/2)) == NULL)
            return -76;
    }
    PJ_CATCH_ANY {
        return -77;
    }
    PJ_END;

    /* On the next alloc, exception should be thrown */
    PJ_TRY {
        p = pj_pool_alloc(pool, STATIC_BUF_SIZE);
        if (p != NULL) {
            /* This is unexpected, the alloc should fail */
            return -78;
        }
    }
    PJ_CATCH_ANY {
        /* This is the expected result */
    }
    PJ_END;

    /* Done */
    return 0;
}

int pool_test(void)
{
    enum { LOOP = 2 }
    int loop;
    int rc;

    rc = capacity_test();
    if (rc) return rc;

    for (loop = 0; loop < LOOP; ++loop) {
        /* Test that the pool should grow automatically. */
        rc = drain_test(SIZE, SIZE);
        if (rc != 0) return rc;

        /* Test situation where pool is not allowed to grow.
         * We expect the test to return correct error.
         */
    }
}

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
rc = drain_test(SIZE, 0);
    if (rc != -40) return rc;
}

rc = pool_buf_test();
if (rc != 0)
    return rc;

return 0;
}

#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_pool_test;
#endif /* INCLUDE_POOL_TEST */
9.14 Test: Socket Select()

This file provides implementation of select_test(). It tests the functionality of the pj_sock_select() API. This file is pjlib-test/select.c

```c
#include "test.h"
#if INCLUDE_SELECT_TEST
#include <pj/sock.h>
#include <pj/sock_select.h>
#include <pj/log.h>
#include <pj/string.h>
#include <pj/assert.h>
#include <pj/os.h>
#include <pj/errno.h>
enum {
    READ_FDS,
    WRITE_FDS,
    EXCEPT_FDS
};
#define UDP_PORT 51232
#define THIS_FILE "select_test"

/*
   * do_select()
   * Perform pj_sock_select() and find out which sockets
   * are signalled.
   */
static int do_select( pj_sock_t sock1, pj_sock_t sock2,
                      int setcount[])
{
    pj_fd_set_t fds[3];
    pj_time_val timeout;
    int i, n;
    for (i=0; i<3; ++i) {
        PJ_FD_ZERO(&fds[i]);
        PJ_FD_SET(sock1, &fds[i]);
        PJ_FD_SET(sock2, &fds[i]);
        setcount[i] = 0;
    }
    timeout.sec = 1;
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
timeout.msec = 0;

n = pj_sock_select(FD_SETSIZE, &fds[0], &fds[1], &fds[2],
    &timeout);
if (n < 0)
    return n;
if (n == 0)
    return 0;
for (i=0; i<3; ++i) {
    if (PJ_FD_ISSET(sock1, &fds[i]))
        setcount[i]++;
    if (PJ_FD_ISSET(sock2, &fds[i]))
        setcount[i]++;
}

return n;

/*********************************************************/

int select_test()
{
    pj_sock_t udp1=PJ_INVALID_SOCKET, udp2=PJ_INVALID_SOCKET;
    pj_sockaddr_in udp_addr;
    int status;
    int setcount[3];
    pj_str_t s;
    const char data[] = "hello";
    const int datalen = 5;
    pj_ssize_t sent, received;
    char buf[10];
    pj_status_t rc;

    PJ_LOG(3, (THIS_FILE, "...Testing simple UDP select()"));

    // Create two UDP sockets.
    rc = pj_sock_socket( pj_AF_INET(), pj_SOCK_DGRAM(), 0, &udp1);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: unable to create socket", rc);
        status=-10; goto on_return;
    }
    rc = pj_sock_socket( pj_AF_INET(), pj_SOCK_DGRAM(), 0, &udp2);
    if (udp2 == PJ_INVALID_SOCKET) {
        app_perror("...error: unable to create socket", rc);
        status=-20; goto on_return;
    }

    // Bind one of the UDP socket.
    pj_bzero(&udp_addr, sizeof(udp_addr));
    udp_addr.sin_family = pj_AF_INET();
    udp_addr.sin_port = UDP_PORT;
    udp_addr.sin_addr = pj_inet_addr(pj_cstr(&s, "127.0.0.1"));
    if (pj_sock_bind(udp2, &udp_addr, sizeof(udp_addr)) ) {
        status=-30; goto on_return;
    }

    // Send data.
    sent = datalen;
    rc = pj_sock_sendto(udp1, data, &sent, 0, &udp_addr, sizeof(udp_addr));
    if (rc != PJ_SUCCESS || sent != datalen) {
        app_perror("...error: sendto() error", rc);
        status=-40; goto on_return;
    }
}

9.14 Test: Socket Select()

// Check that socket is marked as reable.
// Note that select() may also report that sockets are writable.
status = do_select(udp1, udp2, setcount);
if (status < 0) {
    char errbuf[128];
    pj_strerror(pj_get_netos_error(), errbuf, sizeof(errbuf));
    PJ_LOG(1, (THIS_FILE, "...error: %s", errbuf));
    status = -50; goto on_return;
}
if (status == 0) {
    status = -60; goto on_return;
}
if (setcount[READ_FDS] != 1) {
    status = -70; goto on_return;
}
if (setcount[WRITE_FDS] != 0) {
    if (setcount[WRITE_FDS] == 2) {
        PJ_LOG(3, (THIS_FILE, "...info: system reports writable sockets"));
    } else {
        status = -80; goto on_return;
    }
} else {
    PJ_LOG(3, (THIS_FILE, "...info: system doesn’t report writable sockets"));
}
if (setcount[EXCEPT_FDS] != 0) {
    status = -90; goto on_return;
}

// Read the socket to clear readable sockets.
received = sizeof(buf);
rc = pj_sock_recv(udp2, buf, &received, 0);
if (rc != PJ_SUCCESS || received != 5) {
    status = -100; goto on_return;
}
status = 0;

// Test timeout on the read part.
// This won’t necessarily return zero, as select() may report that
// sockets are writable.
status = do_select(udp1, udp2, setcount);
if (status != 0 && status != setcount[WRITE_FDS]) {
    PJ_LOG(3, (THIS_FILE, "...error: expecting timeout but got %d sks set", status));
    PJ_LOG(3, (THIS_FILE, " rdset: %d, wrset: %d, exset: %d", setcount[0], setcount[1], setcount[2]));
    status = -110; goto on_return;
}
if (setcount[READ_FDS] != 0) {
    PJ_LOG(3, (THIS_FILE, "...error: readable socket not expected"));
    status = -120; goto on_return;
}
status = 0;

on_return:
if (udp1 != PJ_INVALID_SOCKET)
    pj_sock_close(udp1);
if (udp2 != PJ_INVALID_SOCKET)
    pj_sock_close(udp2);
return status;
#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_select_test;
#endif /* INCLUDE_SELECT_TEST */
9.15 Test: Sleep, Time, and Timestamp

This file provides implementation of sleep_test().

9.15.1 Scope of the Test

This tests:

- whether pj_thread_sleep() works.
- whether pj_gettimeofday() works.
- whether pj_get_timestamp() and friends works.

API tested:

- pj_thread_sleep()
- pj_gettimeofday()
- PJ_TIME_VAL_SUB()
- PJ_TIME_VAL_LTE()
- pj_get_timestamp()
- pj_get_timestamp_freq() (implicitly)
- pj_elapsed_time()
- pj_elapsed_usec()

This file is pjlib-test/sleep.c

/* $Id: sleep.c 974 2007-02-19 01:13:53Z benny1p $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include "test.h"

#if INCLUDE_SLEEP_TEST
#include <pjlib.h>
#define THIS_FILE "sleep_test"

static int simple_sleep_test(void)
enum { COUNT = 5 };
int i;
pj_status_t rc;

PJ_LOG(3,(THIS_FILE, "...will write messages every 1 second:");

for (i=0; i<COUNT; ++i) {
    rc = pj_thread_sleep(1000);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: pj_thread_sleep()", rc);
        return -10;
    }
    PJ_LOG(3,(THIS_FILE, "...wake up.");
}

return 0;
}

static int sleep_duration_test(void)
{
    enum { MIS = 20, DURATION = 1000, DURATION2 = 500 }; 
    pj_status_t rc;
    PJ_LOG(3,(THIS_FILE, "..running sleep duration test");

    /* Test pj_thread_sleep() and pj_gettimeofday() */
    { 
        pj_timeval start, stop;
        pj_uint32_t msec;

        /* Mark start of test. */
        rc = pj_gettimeofday(&start);
        if (rc != PJ_SUCCESS) {
            app_perror("...error: pj_gettimeofday()", rc);
            return -10;
        }

        /* Sleep */
        rc = pj_thread_sleep(DURATION);
        if (rc != PJ_SUCCESS) {
            app_perror("...error: pj_thread_sleep()", rc);
            return -20;
        }

        /* Mark end of test. */
        rc = pj_gettimeofday(&stop);

        /* Calculate duration (store in stop). */
        PJ_TIME_VAL_SUB(stop, start);
        /* Convert to msec. */
        msec = PJ_TIME_VAL_MSEC(stop);
        /* Check if it’s within range. */
        if (msec < DURATION * (100-MIS)/100 ||
            msec > DURATION * (100+MIS)/100)
        { 
            PJ_LOG(3,(THIS_FILE,
                      "...error: slept for %d ms instead of %d ms 
                      "(outside %d%% err window)",
                      msec, DURATION, MIS));
            return -30;
        }
    }
}

/* Test pj_thread_sleep() and pj_get_timestamp() and friends */
{
    pj_time_val t1, t2;
    pj_timestamp start, stop;
    pj_uint32_t msec;

    pj_thread_sleep(0);
    /* Mark start of test. */
    rc = pj_get_timestamp(&start);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: pj_get_timestamp()", rc);
        return -60;
    }
    /* ...also with gettimeofday() */
    pj_gettimeofday(&t1);
    /* Sleep */
    rc = pj_thread_sleep(DURATION2);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: pj_thread_sleep()", rc);
        return -70;
    }
    /* Mark end of test. */
    pj_get_timestamp(&stop);
    /* ...also with gettimeofday() */
    pj_gettimeofday(&t2);
    /* Compare t1 and t2. */
    if (PJ_TIME_VAL_LT(t2, t1)) {
        PJ_LOG(3,(THIS_FILE, "...error: t2 is less than t1!!");
        return -75;
    }
    /* Get elapsed time in msec */
    msec = pj_elapsed_msec(&start, &stop);
    /* Check if it's within range. */
    if (msec < DURATION2 * (100-MIS)/100 ||
        msec > DURATION2 * (100+MIS)/100)
    {
        PJ_LOG(3,(THIS_FILE,
            "...error: slept for %d ms instead of %d ms ",
            "(outside %d%% err window)",
            msec, DURATION2, MIS));
        PJ_TIME_VAL_SUB(t2, t1);
        PJ_LOG(3,(THIS_FILE,
            "...info: gettimeofday() reported duration is ",
            "%d msec",
            PJ_TIME_VAL_MSEC(t2)));
        return -76;
    }
    /* All done. */
    return 0;
}

int sleep_test()
{
    int rc;
    rc = simple_sleep_test();
}
if (rc != PJ_SUCCESS)
    return rc;

crc = sleep_duration_test();
if (rc != PJ_SUCCESS)
    return rc;

return 0;

#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
dummy_sleep_test;
#endif /* INCLUDE_SLEEP_TEST */
9.16 Test: Socket

This file provides implementation of sock_test(). It tests the various aspects of the socket API.

9.16.1 Scope of the Test

The scope of the test:

- verify the validity of the address structs.
- verify that address manipulation API works.
- simple socket creation and destruction.
- simple socket send/recv and sendto/recvfrom.
- UDP connect()
- send/recv big data.
- all for both UDP and TCP.

The APIs tested in this test:

- pj_inet_aton()
- pj_inet_ntoa()
- pj_gethostname()
- pj_sock_socket()
- pj_sock_close()
- pj_sock_send()
- pj_sock_sendto()
- pj_sock_recv()
- pj_sock_recvfrom()
- pj_sock_bind()
- pj_sock_connect()
- pj_sock_listen()
- pj_sock_accept()
- pj_gethostbyname()

This file is pjlib-test/sock.c
/* $Id: sock.c 1405 2007-07-20 08:08:30Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
#include <pjlib.h>
#include "test.h"

#if INCLUDE_SOCK_TEST
#define UDP_PORT 51234
#define TCP_PORT (UDP_PORT+10)
#define BIG_DATA_LEN 8192
#define ADDRESS "127.0.0.1"
static char bigdata[BIG_DATA_LEN];
static char bigbuffer[BIG_DATA_LEN];

static int format_test(void)
{
    pj_str_t s = pj_str(ADDRESS);
    unsigned char *p;
    pj_in_addr addr;
    char zero[64];
    pj_sockaddr_in addr2;
    const pj_str_t *hostname;
    const unsigned char A[] = {127, 0, 0, 1};
    PJ_LOG(3,("test", ...format_test()"));
    /* pj_inet_aton() */
    if (pj_inet_aton(&s, &addr) != 1)
        return -10;
    /* Check the result. */
    p = (unsigned char*)&addr;
        PJ_LOG(3,("test", " error: mismatched address. p0=%d, p1=%d, 
                   p2=%d, p3=%d", p[0] & 0xFF, p[1] & 0xFF,
        return -15;
    }
    /* pj_inet_ntoa() */
    p = (unsigned char*)pj_inet_ntoa(addr);
    if (!p)
        return -20;
    if (pj_strcmp2(&s, (char*)p) != 0)
        return -30;
    /* Test that pj_sockaddr_in_init() initialize the whole structure,
       including sin_zero. */
    return 0;
}
#endif
/*
pj_sockaddr_in_init(&addr2, 0, 1000);
pj_bzero(zero, sizeof(zero));
if (pj_memcmp(addr2.sin_zero, zero, sizeof(addr2.sin_zero)) != 0)
    return -35;
*/

/* pj_gethostname() */
hostname = pj_gethostname();
if (!hostname || !hostname->ptr || !hostname->slen)
    return -40;

PJ_LOG(3,("test", "....hostname is %.*s",
    (int)hostname->slen, hostname->ptr));

/* pj_gethostaddr() */
return 0;
}

static int simple_sock_test(void)
{
    int types[2];
pj_sock_t sock;
    int i;
pj_status_t rc = PJ_SUCCESS;
    types[0] = pj_SOCK_STREAM();
    types[1] = pj_SOCK_DGRAM();

    PJ_LOG(3,("test", "....simple_sock_test()"));

    for (i=0; i<(int)(sizeof(types)/sizeof(types[0])); ++i) {
        rc = pj_sock_socket(pj_AF_INET(), types[i], 0, &sock);
        if (rc != PJ_SUCCESS) {
            app_perror("...error: unable to create socket", rc);
            break;
        } else {
            rc = pj_sock_close(sock);
            if (rc != 0) {
                app_perror("...error: close socket", rc);
                break;
            }
        }
    }

    return rc;
}

static int send_recv_test(int sock_type,
    pj_sock_t ss, pj_sock_t cs,
    pj_sockaddr_in *dstaddr, pj_sockaddr_in *srcaddr,
    int addrlen)
{
    enum { DATA_LEN = 16 };
    char senddata[DATA_LEN+4], recvdatalen=DATA_LEN+4;
    pj_size_t sent, received, total_received;
    pj_status_t rc;

    TRACE_(("test", "....create_random_string()"));

    senddata[DATA_LEN-1] = '\0';

    /*
     * Test send/recv small data.
     */

TRACE_("(test", ".sendto()"));)
if (dstaddr) {
    sent = DATA_LEN;
    rc = pj_sock_sendto(cs, senddata, &sent, 0, dstaddr, addrlen);
    if (rc != PJ_SUCCESS || sent != DATA_LEN) {
        app_perror("...sendto error", rc);
        rc = -140; goto on_error;
    }
} else {
    sent = DATA_LEN;
    rc = pj_sock_send(cs, senddata, &sent, 0);
    if (rc != PJ_SUCCESS || sent != DATA_LEN) {
        app_perror("...send error", rc);
        rc = -145; goto on_error;
    }
}
TRACE_("(test", ".recv()"));
if (srcaddr) {
    pj_sockaddr_in addr;
    int srclen = sizeof(addr);
    pj_bzero(&addr, sizeof(addr));
    received = DATA_LEN;
    rc = pj_sock_recvfrom(ss, recvdata, &received, 0, &addr, &srclen);
    if (rc != PJ_SUCCESS || received != DATA_LEN) {
        app_perror("...recvfrom error", rc);
        rc = -150; goto on_error;
    }
    if (srclen != addrlen)
        return -151;
    if (pj_memcmp(&addr, srcaddr, srclen) != 0) {
        char srcaddr_str[32], addr_str[32];
        strcpy(srcaddr_str, pj_inet_ntoa(srcaddr->sin_addr));
        strcpy(addr_str, pj_inet_ntoa(addr.sin_addr));
        PJ_LOG(3,("test", ","...error: src address mismatch (original=%s, 
                      "recvfrom addr=%s)", 
                      srcaddr_str, addr_str));
        return -152;
    }
} else {
    /* Repeat recv() until all data is received. 
     * This applies only for non-UDP of course, since for UDP 
     * we would expect all data to be received in one packet. 
     */
    total_received = 0;
    do {
        received = DATA_LEN-total_received;
        rc = pj_sock_recv(ss, recvdata+total_received, &received, 0);
        if (rc != PJ_SUCCESS) {
            app_perror("...recv error", rc);
            rc = -155; goto on_error;
        }
        if (received <= 0) {
            PJ_LOG(3,("", ","...error: socket has closed! (received=%d)",
                       received));
            rc = -156; goto on_error;
        }
        if (received != DATA_LEN-total_received) {
            if (sock_type != pj_SOCK_STREAM()) {
                PJ_LOG(3,("", ","...error: expecting %u bytes, got %u bytes",
                           DATA_LEN-total_received, received));
                rc = -157; goto on_error;
            }
        }
    }
}
9.16 Test: Socket

```c
total_received += received;
} while (total_received < DATA_LEN);
}

TRACE_("("test", "....memcmp()")");
if (pj_memcmp(senddata, recvdata, DATA_LEN) != 0) {
  PJ_LOG(3,("","...error: received data mismatch 
  "(got:’%s’ expecting:’%s’", 
  recvdata, senddata));
  rc = -160; goto on_error;
}

TRACE_("("test", "....sendto()")");
if (dstaddr) {
  sent = BIG_DATA_LEN;
  rc = pj_sock_sendto(cs, bigdata, &sent, 0, dstaddr, addrlen);
  if (rc != PJ_SUCCESS || sent != BIG_DATA_LEN) {
    app_perror("...sendto error", rc);
    rc = -161; goto on_error;
  }
} else {
  sent = BIG_DATA_LEN;
  rc = pj_sock_send(cs, bigdata, &sent, 0);
  if (rc != PJ_SUCCESS || sent != BIG_DATA_LEN) {
    app_perror("...send error", rc);
    rc = -165; goto on_error;
  }
}

TRACE_("("test", "....recv()")");

*/
* Test send/recv big data.
*
TRACE_("("test", "....sendto()")");
if (dstaddr) {
  sent = BIG_DATA_LEN;
  rc = pj_sock_sendto(cs, bigdata, &sent, 0, dstaddr, addrlen);
  if (rc != PJ_SUCCESS || sent != BIG_DATA_LEN) {
    app_perror("...sendto error", rc);
    rc = -161; goto on_error;
  }
} else {
  sent = BIG_DATA_LEN;
  rc = pj_sock_send(cs, bigdata, &sent, 0);
  if (rc != PJ_SUCCESS || sent != BIG_DATA_LEN) {
    app_perror("...send error", rc);
    rc = -165; goto on_error;
  }
}

TRACE_("("test", "....recv()")");

/* Repeat recv() until all data is received.
 * This applies only for non-UDP of course, since for UDP
 * we would expect all data to be received in one packet.
 */
total_received = 0;
do {
  received = BIG_DATA_LEN-total_received;
  rc = pj_sock_recv(ss, bigbuffer+total_received, &received, 0);
  if (rc != PJ_SUCCESS) {
    app_perror("...recv error", rc);
    rc = -170; goto on_error;
  }
  if (received <= 0) {
    PJ_LOG(3,("","...error: socket has closed! (received=%d)", 
            received));
    rc = -173; goto on_error;
  }
  if (received != BIG_DATA_LEN-total_received) {
    if (sock_type != pj_SOCK_STREAM()) {
      PJ_LOG(3,("","...error: expecting %u bytes, got %u bytes", 
                BIG_DATA_LEN-total_received, received));
      rc = -176; goto on_error;
    }
  }
  total_received += received;
} while (total_received < BIG_DATA_LEN);

TRACE_("("test", "....memcmp()")");
if (pj_memcmp(bigdata, bigbuffer, BIG_DATA_LEN) != 0) {
  PJ_LOG(3,("","...error: received data has been altered!")
  rc = -180; goto on_error;
}
```
342 PJLIB Reference Page Documentation

```c
rc = 0;

on_error:
    return rc;
}

static int udp_test(void)
{
    pj_sock_t cs = PJ_INVALID_SOCKET, ss = PJ_INVALID_SOCKET;
    pj_sockaddr_in dstaddr, srcaddr;
    pj_str_t s;
    pj_status_t rc = 0, retval;

    PJ_LOG(3,("test", ". . . udp_test()");

    rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &ss);
    if (rc != 0) {
        app_perror("error: unable to create socket", rc);
        return -100;
    }

    rc = pj_sock_socket(pj_AF_INET(), pj_SOCK_DGRAM(), 0, &cs);
    if (rc != 0)
        return -110;

    /* Bind server socket. */
    pj_bzero(&dstaddr, sizeof(dstaddr));
    dstaddr.sin_family = pj_AF_INET();
    dstaddr.sin_port = pj_htons(UDP_PORT);
    dstaddr.sin_addr = pj_inet_addr(pj_cstr(&s, ADDRESS));
    if ((rc=pj_sock_bind(ss, &dstaddr, sizeof(dstaddr))) != 0) {
        app_perror("bind error udp: / ADDRESS, rc)
        rc = -120; goto on_error;
    }

    /* Bind client socket. */
    pj_bzero(&srcaddr, sizeof(srcaddr));
    srcaddr.sin_family = pj_AF_INET();
    srcaddr.sin_port = pj_htons(UDP_PORT-1);
    srcaddr.sin_addr = pj_inet_addr(pj_cstr(&s, ADDRESS));
    if ((rc=pj_sock_bind(cs, &srcaddr, sizeof(srcaddr))) != 0) {
        app_perror("bind error", rc);
        rc = -121; goto on_error;
    }

    /* Test send/recv, with sendto */
    rc = send_recv_test(pj_SOCK_DGRAM(), ss, cs, &dstaddr, NULL,
                        sizeof(dstaddr));
    if (rc != 0)
        goto on_error;

    /* Test send/recv, with sendto and recvfrom */
    rc = send_recv_test(pj_SOCK_DGRAM(), ss, cs, &dstaddr,
                        &srcaddr, sizeof(dstaddr));
    if (rc != 0)
        goto on_error;

    /* Disable this test on Symbian since UDP connect()/send() failed
     * with S60 3rd edition (including MR2).
     * See http://www.pjsip.org/trac/ticket/264
     */
    #if !defined(PJ_SYMBIAN) || PJ_SYMBIAN==0
    /* connect() the sockets. */
    rc = pj_sock_connect(cs, &dstaddr, sizeof(dstaddr));
    if (rc != 0) {
```
9.16 Test: Socket

```c
app_perror("...connect() error", rc);
rc = -122; goto on_error;
}

/* Test send/recv with send() */
rc = send_recv_test(pj_SOCK_DGRAM(), ss, cs, NULL, NULL, 0);
if (rc != 0)
goto on_error;

/* Test send/recv with send() and recvfrom */
rc = send_recv_test(pj_SOCK_DGRAM(), ss, cs, NULL, &srcaddr, sizeof(srcaddr));
if (rc != 0)
goto on_error;
#endif

on_error:
retval = rc;
if (cs != PJ_INVALID_SOCKET) {
    rc = pj_sock_close(cs);
    if (rc != PJ_SUCCESS) {
        app_perror("...error in closing socket", rc);
        return -1000;
    }
}
if (ss != PJ_INVALID_SOCKET) {
    rc = pj_sock_close(ss);
    if (rc != PJ_SUCCESS) {
        app_perror("...error in closing socket", rc);
        return -1010;
    }
}
return retval;

static int tcp_test(void)
{
    pj_socket_t cs, ss;
    pj_status_t rc = 0, retval;
    PJ_LOG(3,("test", "...tcp_test()"));

    rc = app_socketpair(pj_AF_INET(), pj_SOCK_STREAM(), 0, &ss, &cs);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: app_socketpair()", rc);
        return -2000;
    }

    /* Test send/recv with send() and recv() */
    retval = send_recv_test(pj_SOCK_STREAM(), ss, cs, NULL, NULL, 0);
    rc = pj_sock_close(cs);
    if (rc != PJ_SUCCESS) {
        app_perror("...error in closing socket", rc);
        return -2000;
    }

    rc = pj_sock_close(ss);
    if (rc != PJ_SUCCESS) {
        app_perror("...error in closing socket", rc);
        return -2010;
    }

    return retval;
}
```
static int ioctl_test(void) {
    return 0;
}

static int gethostbyname_test(void) {
    pj_str_t host;
    pj_hostent he;
    pj_status_t status;

    /* Testing pj_gethostbyname() with invalid host */
    host = pj_str("an-invalid-host-name");
    status = pj_gethostbyname(&host, &he);

    /* Must return failure! */
    if (status == PJ_SUCCESS) 
        return -20100;
    else 
        return 0;
}

#if 0
#include "../pj/os_symbian.h"

static int connect_test() {
    RSocketServ rSockServ;
    RSocket rSock;
    TNetAddr inetAddr;
    TRequestStatus reqStatus;
    char buffer[16];
    TPtrC8 data = (const TUint8*)buffer, (TInt)sizeof(buffer));
    int rc;

    rc = rSockServ.Connect();
    if (rc != KErrNone) 
        return rc;

    rc = rSock.Open(rSockServ, KAfInet, KSockDatagram, KProtocolInetUdp);
    if (rc != KErrNone) {
        rSockServ.Close();
        return rc;
    }

    inetAddr.Init(KAfInet);
    inetAddr.Input(_L("127.0.0.1"));
    inetAddr.SetPort(80);

    rSock.Connect(inetAddr, reqStatus);
    User::WaitForRequest(reqStatus);

    if (reqStatus != KErrNone) {
        rSock.Close();
        rSockServ.Close();
        return rc;
    }

    rSock.Send(data, 0, reqStatus);
    User::WaitForRequest(reqStatus);

    if (reqStatus != KErrNone) {
        rSock.Close();
        rSockServ.Close();
        return rc;
    }
}
9.16 Test: Socket

```c
rSock.Close();
rSockServ.Close();
return KErrNone;
#endif

int sock_test()
{
    int rc;
    pj_create_random_string(bigdata, BIG_DATA_LEN);

    // Enable this to demonstrate the error with S60 3rd Edition MR2
    #if 0
        rc = connect_test();
        if (rc != 0)
            return rc;
    #endif

    rc = format_test();
    if (rc != 0)
        return rc;

    rc = gethostbyname_test();
    if (rc != 0)
        return rc;

    rc = simple_sock_test();
    if (rc != 0)
        return rc;

    rc = ioctl_test();
    if (rc != 0)
        return rc;

    rc = udp_test();
    if (rc != 0)
        return rc;

    rc = tcp_test();
    if (rc != 0)
        return rc;

    return 0;
}

#else
    /* To prevent warning about "translation unit is empty"
     * when this test is disabled.
     */
    int dummy_sock_test;
#endif /* INCLUDE_SOCK_TEST */
9.17 Test: Socket Performance

Test the performance of the socket communication. This will perform simple producer-consumer type of test, where we calculate how long does it take to send certain number of packets from producer to consumer.

This file is `pjlib-test/sock_perf.c`

```c
/* $Id: sock_perf.c 1405 2007-07-20 08:08:30Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */
#include "test.h"
#include <pjlib.h>
#include <pj/compat/high_precision.h>
#if INCLUDE_SOCK_PERF_TEST

/* sock_producer_consumer()

Simple producer-consumer benchmarking. Send loop number of
buf_size size packets as fast as possible.
*/
static int sock_producer_consumer(int sock_type,
unsigned buf_size,
unsigned loop,
unsigned *p_bandwidth)
{
    pj_sock_t consumer, producer;
    pj_pool_t *pool;
    char *outgoing_buffer, *incoming_buffer;
    pj_timestamp start, stop;
    unsigned i;
    pj_highprec_t elapsed, bandwidth;
    pj_size_t total_received;
    pj_status_t rc;
    /* Create pool. */
    pool = pj_pool_create(mem, NULL, 4096, 4096, NULL);
    if (!pool)
        return -10;
    /* Create producer-consumer pair. */
    rc = app_socketpair(pj_AF_INET(), sock_type, 0, &consumer, &producer);
    if (rc != PJ_SUCCESS)
        app_perror("...error: create socket pair", rc);
        return -20;
    } /* Create buffers. */
    outgoing_buffer = (char*) pj_pool_alloc(pool, buf_size);
    } /* sock_producer_consumer() */
#endif
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
incoming_buffer = (char*) pj_pool_alloc(pool, buf_size);

/* Start loop. */
pj_get_timestamp(&start);
total_received = 0;
for (i=0; i<loop; ++i) {
    pj_ssize_t sent, part_received, received;
pj_time_val delay;
    sent = buf_size;
    rc = pj_sock_send(producer, outgoing_buffer, &sent, 0);
    if (rc != PJ_SUCCESS || sent != (pj_ssize_t)buf_size) {
        app_perror("...error: send()", rc);
        return -61;
    }
    /* Repeat recv() until all data is part_received.
     * This applies only for non-UDP of course, since for UDP
     * we would expect all data to be part_received in one packet.
     */
    received = 0;
do {
        part_received = buf_size-received;
        rc = pj_sock_recv(consumer, incoming_buffer+received,
                          &part_received, 0);
        if (rc != PJ_SUCCESS) {
            app_perror("...recv error", rc);
            return -70;
        }
        if (part_received <= 0) {
            PJ_LOG(3,("", "...error: socket has closed (part_received=%d)!", part_received));
            return -73;
        }
        if ((pj_size_t)part_received != buf_size-received) {
            if (sock_type != pj_SOCK_STREAM()) {
                PJ_LOG(3,("", "...error: expecting %u bytes, got %u bytes", buf_size-received, part_received));
                return -76;
            }
        }
        received += part_received;
    }while ((pj_size_t)received < buf_size);
total_received += received;
/* Stop test if it’s been runnign for more than 10 secs. */
pj_get_timestamp(&stop);
delay = pj_elapsed_time(&start, &stop);
if (delay.sec > 10)
    break;
}
/* Stop timer. */
pj_get_timestamp(&stop);

elapsed = pj_elapsed_usec(&start, &stop);
/* bandwidth = total_received * 1000 / elapsed */
bandwidth = total_received;
pj_highprec_mul(bandwidth, 1000);
pj_highprec_div(bandwidth, elapsed);
*p_bandwidth = (pj_uint32_t)bandwidth;
/* Close sockets. */
pj_sock_close(consumer);
pj_sock_close(producer);

/* Done */
pj_pool_release(pool);

return 0;
}

int sock_perf_test(void)
{
    enum { LOOP = 64 * 1024 };
    int rc;
    unsigned bandwidth;

    PJ_LOG(3, ("...
    "benchmarking socket 
    "(2 sockets, packet=512, single threaded):"));

    /* Disable this test on Symbian since UDP connect()/send() failed
     * with 360 3rd edition (including MR2).
     * See http://www.pjsip.org/trac/ticket/264
     */
    #if !defined(PJ_SYMBIAN) || PJ_SYMBIAN==0
    /* Benchmarking UDP */
    rc = sock_producer_consumer(pj_SOCK_DGRAM(), 512, LOOP, &bandwidth);
    if (rc != 0) return rc;
    PJ_LOG(3, ("...bandwidth UDP = %d KB/s", bandwidth));
    #endif

    /* Benchmarking TCP */
    rc = sock_producer_consumer(pj_SOCK_STREAM(), 512, LOOP, &bandwidth);
    if (rc != 0) return rc;
    PJ_LOG(3, ("...bandwidth TCP = %d KB/s", bandwidth));

    return rc;
}

#else
    /* To prevent warning about "translation unit is empty"
     * when this test is disabled.
     */
    int dummy_sock_perf_test;
#endif /* INCLUDE_SOCK_PERF_TEST */
9.18 Test: String

This file provides implementation of string_test(). It tests the functionality of the string API.

9.18.1 Scope of the Test

API tested:

- pj_str()
- pj_strcmp()
- pj_strcmp2()
- pj_stricmp()
- pj_strlen()
- pj_strncmp()
- pj_strnicmp()
- pj_strchr()
- pj_strdup()
- pj_strdup2()
- pj_strcpy()
- pj_strcat()
- pj_strtrim()
- pj_utoa()
- pj_strtoul()
- pj_strtoul2()
- pj_create_random_string()
- ... and mode..

This file is pjlib-test/string.c

/* $Id: string.c 1397 2007-06-28 00:50:10Z benny1p $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 */
#include <pj/string.h>
#include <pj/pool.h>
#include <pj/log.h>
#include <pj/os.h>
#include "test.h"

#define THIS_FILE "string.c"

#if INCLUDE_STRING_TEST
#ifdef _MSC_VER
#pragma warning(disable: 4204)
#endif
#define HELLO_WORLD "Hello World"
#define JUST_HELLO "Hello"
#define UL_VALUE 3456789012UL

#if 1
/* See if both integers have the same sign */
PJ_INLINE(int) cmp(const char *expr, int i, int j)
{
    int r = !(i>0 && j>0) || (i<0 && j<0) || (i==0 && j==0);
    if (r) {
        PJ_LOG(3,(THIS_FILE," error: %s: expecting %d, got %d", expr, j, i));
    }
    return r;
}
#else
/* For strict comparison, must be equal */
PJ_INLINE(int) cmp(const char *expr, int i, int j)
{
    PJ_UNUSED_ARG(expr);
    return i!=j;
}
#endif
#define C(expr, res) cmp(#expr, expr, res)

static int stricmp_test(void)
{
    /* This specifically tests and benchmark pj_stricmp(), pj_stricmp_alnum().
     * In addition, it also tests pj_stricmp2(), pj_strnicmp(), and
     * pj_strnicmp2().
     */
    #define STRTEST(res,res2,S1,S2,code) \
    do { \n        s1.ptr=S1; s1.slen=(S1)?len:0; \n        s2.ptr=S2; s2.slen=(S2)?len:0; \n        pj_get_timestamp(&t1); \n        if (C(pj_stricmp(&s1,&s2),res)) return code; \n        pj_get_timestamp(&t2); \n        pj_sub_timestamp(&t2, &t1); \n        pj_get_timestamp(&t1); \n        if (C(pj_stricmp_alnum(&s1,&s2),res)) return code-1; \n        pj_get_timestamp(&t2); \n        pj_sub_timestamp(&t2, &t1); \n        pj_get_timestamp(&t1); \n        if (C(pj_stricmp2(&s1,S2),res2)) return code*10; \n        pj_get_timestamp(&t2); \n        pj_sub_timestamp(&t2, &t1); \n        pj_get_timestamp(&t1); \n        if (C(pj_strnicmp(&s1,&s2,len),res)) return code*100; \n        pj_get_timestamp(&t2); \n        pj_sub_timestamp(&t2, &t1); \n        if (C(pj_strnicmp2(&s1,S2,len),res)) return code*1000; \n    } while (0)
char *buf;
pj_str_t s1, s2;
pj_timestamp t1, t2, e1, e2, zero;
pj_uint32_t c1, c2;
int len;
e1.u32.hi = e1.u32.lo = e2.u32.hi = e2.u32.lo = 0;
pj_thread_sleep(0);
#define SNULL 0

/* Compare empty strings. */
len=0;
STRTEST( 0, 0, "", "", -500);
STRTEST( 0, 0, SNULL, SNULL, -502);
STRTEST( 0, 0, "", SNULL, -504);
STRTEST( 0, 0, SNULL, SNULL, -506);
STRTEST( 0, -1, "hello", "world", -508);

/* equal, length=1
* use buffer to simulate non-aligned string.
*/
buf = "a""A";
len=1;
STRTEST( 0, -1, "a", buf+0, -510);
STRTEST( 0, 0, "a", buf+1, -512);
STRTEST(-1, -1, "O", "P", -514);
STRTEST(-1, -1, SNULL, "a", -516);
STRTEST( 1, 1, "a", SNULL, -518);

/* equal, length=2
* use buffer to simulate non-aligned string.
*/
buf = "aa""Aa""aA""AA";
len=2;
STRTEST( 0, -1, "aa", buf+0, -520);
STRTEST( 0, -1, "aa", buf+2, -522);
STRTEST( 0, -1, "aa", buf+4, -524);
STRTEST( 0, 0, "aa", buf+6, -524);

/* equal, length=3
* use buffer to simulate non-aligned string.
*/
buf = "aaa""Aaa""aAa""aaA""aAA""AAA";
len=3;
STRTEST( 0, -1, "aaa", buf+0, -530);
STRTEST( 0, -1, "aaa", buf+3, -532);
STRTEST( 0, -1, "aaa", buf+6, -534);
STRTEST( 0, -1, "aaa", buf+9, -536);
STRTEST( 0, -1, "aaa", buf+12, -538);
STRTEST( 0, -1, "aaa", buf+15, -540);
STRTEST( 0, -1, "aaa", buf+18, -542);
STRTEST( 0, 0, "aaa", buf+21, -544);

/* equal, length=4 */
len=4;
STRTEST( 0, 0, "aaaa", "aaaa", -540);
STRTEST( 0, 0, "aaaa", "Aaaa", -542);
STRTEST( 0, 0, "aaaa", "aAaa", -544);
STRTEST( 0, 0, "aaaa", "aaAa", -546);
STRTEST( 0, 0, "aaaa", "aaaA", -548);
STRTEST( 0, 0, "aaaa", "AAaa", -550);
STRTEST( 0, 0, "aaaa", "aAaa", -552);
STRTEST( 0, 0, "aaaa", "aaaA", -554);
STRTEST( 0, 0, "aaaa", "Aaaa", -556);
STRTEST( 0, 0, "aaaa", "aaAa", -558);
STRTEST( 0, 0, "aaaa","AaaA",-560);
STRTEST( 0, 0, "aaaa","AAAa",-562);
STRTEST( 0, 0, "aaaa","aAAA",-564);
STRTEST( 0, 0, "aaaa","AAaA",-566);
STRTEST( 0, 0, "aaaa","AaAA",-568);
STRTEST( 0, 0, "aaaa","AAAA",-570);

    /* equal, length=5 */
    buf = "aaaa""Aaaa""AaaA""AAAAA";
    len=5;
    STRTEST( 0, 0, "aaaaa",buf+0,-580);
    STRTEST( 0, 0, "aaaaa",buf+5,-582);
    STRTEST( 0, 0, "aaaaa",buf+10,-584);
    STRTEST( 0, 0, "aaaaa",buf+15,-586);

    /* not equal, length=1 */
    len=1;
    STRTEST( -1, -1, "a", "b", -600);

    /* not equal, length=2 */
    buf = "ab""ba";
    len=2;
    STRTEST( -1, -1, "aa", buf+0, -610);
    STRTEST( -1, -1, "aa", buf+2, -612);

    /* not equal, length=3 */
    buf = "aab""aba""baa";
    len=3;
    STRTEST( -1, -1, "aaa", buf+0, -620);
    STRTEST( -1, -1, "aaa", buf+3, -622);
    STRTEST( -1, -1, "aaa", buf+6, -624);

    /* not equal, length=4 */
    buf = "aaab""aab""aba""baa";
    len=4;
    STRTEST( -1, -1, "aaaa", buf+0, -630);
    STRTEST( -1, -1, "aaaa", buf+4, -632);
    STRTEST( -1, -1, "aaaa", buf+8, -634);
    STRTEST( -1, -1, "aaaa", buf+12, -636);

    /* not equal, length=5 */
    buf="aaab""aab""aba""aba""baa";
    len=5;
    STRTEST( -1, -1, "aaaaa", buf+0, -640);
    STRTEST( -1, -1, "aaaaa", buf+5, -642);
    STRTEST( -1, -1, "aaaaa", buf+10, -644);
    STRTEST( -1, -1, "aaaaa", buf+15, -646);
    STRTEST( -1, -1, "aaaaa", buf+20, -648);

zero.u32.hi = zero.u32.lo = 0;
c1 = pj_elapsed_cycle(&zero, &e1);
c2 = pj_elapsed_cycle(&zero, &e2);

if (c1 < c2) {
PJ_LOG(3,"" , " info: pj_stricmp_alnum is slower than pj_stricmp!");
    //return -700;
}

    /* Avoid division by zero */
    if (c2 == 0) c2=1;

    PJ_LOG(3, ("" , " time: stricmp=%u, stricmp_alnum=%u (speedup=%d.%02dx)",
        c1, c2,
        (c1 + 100 / c2) / 100,
        (c1 + 100 / c2) % 100));
    return 0;
#undef STRTEST
9.18 Test: String

/* This tests pj_strcmp(), pj_strcmp2(), pj_strncmp(), pj_strncmp2() */
static int strcmp_test(void)
{
#define STR_TEST(res,S1,S2,code)\
    do {\
        s1.ptr=S1; s1.slen=S1?len:0; \\n        s2.ptr=S2; s2.slen=S2?len:0; \\n        if (C(pj_strcmp(&s1,&s2),res)) return code; \\n        if (C(pj_strcmp2(&s1,S2),res)) return code-1; \\n        if (C(pj_strncmp(&s1,&s2,len),res)) return code-2; \\n        if (C(pj_strncmp2(&s1,S2,len),res)) return code-3; \\n    } while (0)

pj_str_t s1, s2;
int len;

/* Test with length == 0 */
len=0;
STR_TEST(0, "", "", -400);
STR_TEST(0, SNULL, "", -405);
STR_TEST(0, "", SNULL, -410);
STR_TEST(0, SNULL, SNULL, -415);
STR_TEST(0, "hello", "", -420);
STR_TEST(0, "hello", SNULL, -425);

/* Test with length != 0 */
len = 2;
STR_TEST(0, "12", "12", -430);
STR_TEST(1, "12", "1", -435);
STR_TEST(-1, "1", "12", -440);
STR_TEST(-1, SNULL, "12", -445);
STR_TEST(1, "12", SNULL, -450);

return 0;
#undef STR_TEST
}

int string_test(void)
{
    const pj_str_t hello_world = { HELLO_WORLD, strlen(HELLO_WORLD) };
    const pj_str_t just_hello = { JUST_HELLO, strlen(JUST_HELLO) };
    pj_str_t s1, s2, s3, s4, s5;
    enum { RCOUNT = 10, RLEN = 16 };
    pj_str_t random[RCOUNT];
    pj_pool_t *pool;
    int i;

    pool = pj_pool_create(mem, SNULL, 4096, 0, SNULL);
    if (!pool) return -5;

    /* pj_str(), pj_strcmp(), pj_stricmp(), pj_strlen(),
       pj_strncmp(), pj_strchr() */
    s1 = pj_str(HELLO_WORLD);
    if (pj_strcmp(&s1, &hello_world) != 0) return -10;
    if (pj_stricmp(&s1, &hello_world) != 0) return -20;
    if (pj_strcmp(&s1, &just_hello) <= 0) return -30;
    if (pj_stricmp(&s1, &just_hello) <= 0) return -40;
    if (pj_strlen(&s1) != strlen(HELLO_WORLD))

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
return -50;
if (pj_strcmp(&s1, &hello_world) != 0)
return -60;
if (pj_strcmp2(&s1, &just_hello) != 0)
return -70;
if (pj_strcmp2(&s1, HELLO_WORLD JUST_HELLO) != 0)
return -80;

/*
 * pj_strdup()
 */
if (!pj_strdup(pool, &s2, &s1))
return -100;
if (pj_strcmp(&s1, &s2) != 0)
return -110;

/*
 * pj_strcpy(), pj_strcat()
 */
s3.ptr = (char*) pj_pool_alloc(pool, 256);
if (!s3.ptr)
return -200;
pj_strcpy(&s3, &s2);
pj_strcat(&s3, &just_hello);
if (pj_strcmp2(&s3, HELLO_WORLD JUST_HELLO) != 0)
return -210;

/*
 * pj_strdup2(), pj_strtrim().
 */
pj_strdup2(pool, &s4, " HELLO_WORLD 	 ");
pj_strtrim(&s4);
if (pj_strcmp2(&s4, HELLO_WORLD) != 0)
return -250;

/*
 * pj_utoa()
 */
s5.ptr = (char*) pj_pool_alloc(pool, 16);
if (!s5.ptr)
return -270;
s5.slen = pj_utoa(UL_VALUE, s5.ptr);

/*
 * pj_strtoul()
 */
if (pj_strtoul(&s5) != UL_VALUE)
return -280;

/*
 * pj_strtoul2()
 */
s5 = pj_str("123456");
pj_strtoul2(&s5, SNULL, 10); /* Crash test */
if (pj_strtoul2(&s5, &s4, 10) != 123456UL)
return -290;
if (s4.slen != 0)
return -291;
if (pj_strtoul2(&s5, &s4, 16) != 0x123456UL)
return -292;
s5 = pj_str("0123ABCD");
if (pj_strtoul2(&s5, &s4, 10) != 123)
return -293;
if (s4.slen != 4) return -294;
if (s4.ptr == SNULL || *s4.ptr != 'A') return -295;
if (pj_strtoul2(&s5, &s4, 16) != 0x123ABCDUL) return -296;
if (s4.slen != 0) return -297;

/*
 * pj_create_random_string()
 * Check that no duplicate strings are returned.
 */
for (i=0; i<RCOUNT; ++i) {
    int j;
    random[i].ptr = (char*) pj_pool_alloc(pool, RLEN);
    if (!random[i].ptr) return -320;
    random[i].slen = RLEN;
    pj_create_random_string(random[i].ptr, RLEN);
    for (j=0; j<i; ++j) {
        if (pj_strcmp(&random[i], &random[j])==0) return -330;
    }
}
/* Done. */
pj_pool_release(pool);

/* Case sensitive comparison test. */
i = strcmp_test();
if (i != 0) return i;
/* Caseless comparison test. */
i = stricmp_test();
if (i != 0) return i;
return 0;
}
#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_string_test;
#endif /* INCLUDE_STRING_TEST */
9.19 Test: Thread Test

This file contains thread_test() definition.

9.19.1 Scope of Test

This tests:

- whether PJ_THREAD_SUSPENDED flag works.
- whether multithreading works.
- whether thread timeslicing works, and threads have equal time-slice proportion.

APIs tested:

- pj_thread_create()
- pj_thread_register()
- pj_thread_this()
- pj_thread_get_name()
- pj_thread_destroy()
- pj_thread_resume()
- pj_thread_sleep()
- pj_thread_join()
- pj_thread_destroy()

This file is pjlib-test/thread.c

/* $Id: thread.c 974 2007-02-19 01:13:53Z bennylp $ */
/*
 * Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <pjlib.h>

#define THIS_FILE "thread_test"
static volatile int quit_flag=0;

#if 0
#define TRACE__(args) PJ_LOG(3,args)
#else
#define TRACE__(args)
#endif

/*
 * The thread’s entry point.
 * Each of the thread mainly will just execute the loop which
 * increments a variable.
 */
static void* thread_proc(pj_uint32_t *pcounter)
{
    /* Test that pj_thread_register() works. */
    pj_thread_desc desc;
    pj_thread_t *this_thread;
    unsigned id;
    pj_status_t rc;
    id = *pcounter;
    TRACE__((THIS_FILE, " thread %d running..", id));
    pj_bzero(desc, sizeof(desc));
    rc = pj_thread_register("thread", desc, &this_thread);
    if (rc != PJ_SUCCESS) {
        app_perror("...error in pj_thread_register", rc);
        return NULL;
    }
    /* Test that pj_thread_this() works */
    this_thread = pj_thread_this();
    if (this_thread == NULL) {
        PJ_LOG(3,(THIS_FILE, "...error: pj_thread_this() returns NULL!");
        return NULL;
    }
    /* Test that pj_thread_get_name() works */
    if (pj_thread_get_name(this_thread) == NULL) {
        PJ_LOG(3,(THIS_FILE, "...error: pj_thread_get_name() returns NULL!");
        return NULL;
    }
    /* Main loop */
    for (;;) {quit_flag;}
    (*pcounter)++;
    //Must sleep if platform doesn’t do time-slicing.
    //pj_thread_sleep(0);
    TRACE__((THIS_FILE, " thread %d quitting..", id));
    return NULL;
}

/*
 * simple_thread()
 */
static int simple_thread(const char *title, unsigned flags)
{
    pj_pool_t *pool;
    pj_thread_t *thread;
    pj_status_t rc;
    ...
```c
pj_uint32_t counter = 0;
PJ_LOG(3,(THIS_FILE, "..%s", title));
pool = pj_pool_create(mem, NULL, 4000, 4000, NULL);
if (!pool)
    return -1000;
quit_flag = 0;
TRACE__((THIS_FILE, " Creating thread 0..");
rc = pj_thread_create(pool, "thread", (pj_thread_proc*)&thread_proc,
    &counter,
    PJ_THREAD_DEFAULT_STACK_SIZE,
    flags,
    &thread);
if (rc != PJ_SUCCESS) {
    app_perror("...error: unable to create thread", rc);
    return -1010;
}
TRACE__((THIS_FILE, " Main thread waiting..");
pj_thread_sleep(1500);
TRACE__((THIS_FILE, " Main thread resuming..");
if (flags & PJ_THREAD_SUSPENDED) {
    /* Check that counter is still zero */
    if (counter != 0) {
        PJ_LOG(3,(THIS_FILE, "...error: thread is not suspended");
        return -1015;
    }
    rc = pj_thread_resume(thread);
    if (rc != PJ_SUCCESS) {
        app_perror("...error: resume thread error", rc);
        return -1020;
    }
}
PJ_LOG(3,(THIS_FILE, "...waiting for thread to quit..");
pj_thread_sleep(1500);
quit_flag = 1;
pj_thread_join(thread);
pj_pool_release(pool);
if (counter == 0) {
    PJ_LOG(3,(THIS_FILE, "...error: thread is not running");
    return -1025;
}
PJ_LOG(3,(THIS_FILE, "..%s success", title));
return PJ_SUCCESS;
}

/*
 * timeslice_test()
 */
static int timeslice_test(void)
{
    enum { NUM_THREADS = 4 };
    pj_pool_t *pool;
    ...
pj_uint32_t counter[NUM_THREADS], lowest, highest, diff;
pj_thread_t *thread[NUM_THREADS];
unsigned i;
pj_status_t rc;
quit_flag = 0;

pool = pj_pool_create(mem, NULL, 4000, 4000, NULL);
if (!pool)
    return -10;

PJ_LOG(3,(THIS_FILE, ".timeslice testing with %d threads", NUM_THREADS));

/* Create all threads in suspended mode. */
for (i=0; i<NUM_THREADS; ++i) {
    counter[i] = i;
    rc = pj_thread_create(pool, "thread", (pj_thread_proc*)&thread_proc,
                       &counter[i],
                       PJ_THREAD_DEFAULT_STACK_SIZE,
                       PJ_THREAD_SUSPENDED,
                       &thread[i]);
    if (rc!=PJ_SUCCESS) {
        app_perror("...ERROR in pj_thread_create()", rc);
        return -20;
    }
}

/* Sleep for 1 second.
 * The purpose of this is to test whether all threads are suspended.
 */
TRACE__((THIS_FILE, " Main thread waiting..

pj_thread_sleep(1000);
TRACE__((THIS_FILE, " Main thread resuming..

/* Check that all counters are still zero. */
for (i=0; i<NUM_THREADS; ++i) {
    if (counter[i] > i) {
        PJ_LOG(3,(THIS_FILE, "....ERROR! Thread %d-th is not suspended!",
                 i));
        return -30;
    }
}

/* Now resume all threads. */
for (i=0; i<NUM_THREADS; ++i) {
    TRACE__((THIS_FILE, " Resuming thread %d [%p].", i, thread[i]));
    rc = pj_thread_resume(thread[i]);
    if (rc != PJ_SUCCESS) {
        app_perror("...ERROR in pj_thread_resume()", rc);
        return -40;
    }
}

/* Main thread sleeps for some time to allow threads to run.
 * The longer we sleep, the more accurate the calculation will be,
 * but it’ll make user waits for longer for the test to finish.
 */
TRACE__((THIS_FILE, " Main thread waiting (5s)...

pj_thread_sleep(5000);
TRACE__((THIS_FILE, " Main thread resuming..

/* Signal all threads to quit. */
quit_flag = 1;

/* Wait until all threads quit, then destroy. */
for (i=0; i<NUM_THREADS; ++i) {
    TRACE__((THIS_FILE, " Main thread joining thread %d [%p].",

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen


```c
i, thread[i]);
rc = pj_thread_join(thread[i]);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_thread_join()", rc);
    return -50;
}
TRACE__(THIS_FILE, " Destroying thread %d [%p]...", i, thread[i]);
rc = pj_thread_destroy(thread[i]);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR in pj_thread_destroy()", rc);
    return -60;
}
}

TRACE__(THIS_FILE, " Main thread calculating time slices...");

/* Now examine the value of the counters.
 * Check that all threads had equal proportion of processing.
 */
lowest = 0xFFFFFFFF;
highest = 0;
for (i=0; i<NUM_THREADS; ++i) {
    if (counter[i] < lowest)
        lowest = counter[i];
    if (counter[i] > highest)
        highest = counter[i];
}

/* Check that all threads are running. */
if (lowest < 2) {
    PJ_LOG(3,(THIS_FILE, "...ERROR: not all threads were running!");
    return -70;
}

/* The difference between lowest and highest should be lower than 50%.
 */
diff = (highest-lowest)*100 / ((highest+lowest)/2);
if (diff >= 50) {
    PJ_LOG(3,(THIS_FILE, "...ERROR: thread didn’t have equal timeslice!");
    PJ_LOG(3,(THIS_FILE, "......lowest counter=%u, highest counter=%u, diff=%u%%", lowest, highest, diff));
    return -80;
} else {
    PJ_LOG(3,(THIS_FILE, "...info: timeslice diff between lowest & highest=%u%%", diff));
}
pj_pool_release(pool);
return 0;
}

int thread_test(void)
{
    int rc;

    rc = simple_thread("simple thread test", 0);
    if (rc != PJ_SUCCESS)
        return rc;

    rc = simple_thread("suspended thread test", PJ_THREAD_SUSPENDED);
    if (rc != PJ_SUCCESS)
        return rc;

    rc = timeslice_test();
    return rc;
}
```

---

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
if (rc != PJ_SUCCESS) {
    return rc;

    return rc;
}

#else
/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_thread_test;
#endif /* INCLUDE_THREAD_TEST */
9.20 Test: Timer

This file provides implementation of timer_test(). It tests the functionality of the timer heap.

This file is pjlib-test/timer.c

```c
#include "test.h"
#if INCLUDE_TIMER_TEST
#include <pjlib.h>
#define LOOP 16
#define MIN_COUNT 250
#define MAX_COUNT (LOOP * MIN_COUNT)
#define MIN_DELAY 2
#define D (MAX_COUNT / 32000)
#define DELAY (D < MIN_DELAY ? MIN_DELAY : D)
#define THIS_FILE "timer_test"

static void timer_callback(pj_timer_heap_t *ht, pj_timer_entry *e)
{
    PJ_UNUSED_ARG(ht);
    PJ_UNUSED_ARG(e);
}

static int test_timer_heap(void)
{
    int i, j;
    pj_timer_entry *entry;
    pj_pool_t *pool;
    pj_timer_heap_t *timer;
    pj_time_val delay;
    pj_status_t rc; int err=0;
    unsigned size, count;

    size = pj_timer_heap_mem_size(MAX_COUNT)+MAX_COUNT*sizeof(pj_timer_entry);
    pool = pj_pool_create( mem, NULL, size, 4000, NULL);
    if (!pool) { PJ_LOG(3, ("test", ".error: unable to create pool of %u bytes",
                          size));
                       return -10;
    }

    entry = (pj_timer_entry*)pj_pool_malloc(pool, MAX_COUNT, sizeof(*entry));
    if (!entry)
        return -20;
```

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
for (i=0; i<MAX_COUNT; ++i) {
    entry[i].cb = &timer_callback;
}
rc = pj_timer_heap_create(pool, MAX_COUNT, &timer);
if (rc != PJ_SUCCESS) {
    app_perror("...error: unable to create timer heap", rc);
    return -30;
}

count = MIN_COUNT;
for (i=0; i<LOOP; ++i) {
    int early = 0;
    int done=0;
    int cancelled=0;
    int rc;
    pj_timestamp t1, t2, t_sched, t_cancel, t_poll;
    pj_time_val now, expire;
    pj_gettimeofday(&now);
    pj_srand(now.sec);
    t_sched.u32.lo = t_cancel.u32.lo = t_poll.u32.lo = 0;
    // Register timers
    for (j=0; j<(int)count; ++j) {
        delay.sec = pj_rand() % DELAY;
        delay.msec = pj_rand() % 1000;
        // Schedule timer
        pj_get_timestamp(&t1);
        rc = pj_timer_heap_schedule(timer, &entry[j], &delay);
        if (rc != 0)
            return -40;
        pj_get_timestamp(&t2);
        t_sched.u32.lo += (t2.u32.lo - t1.u32.lo);
        // Poll timers.
        pj_get_timestamp(&t1);
        rc = pj_timer_heap_poll(timer, NULL);
        pj_get_timestamp(&t2);
        if (rc > 0) {
            t_poll.u32.lo += (t2.u32.lo - t1.u32.lo);
            early += rc;
        }
    }
    // Set the time where all timers should finish
    pj_gettimeofday(&expire);
    delay.sec = DELAY;
    delay.msec = 0;
    PJ_TIME_VAL_ADD(expire, delay);
    // Wait until all timers finish, cancel some of them.
    do {
        int index = pj_rand() % count;
        pj_get_timestamp(&t1);
        rc = pj_timer_heap_cancel(timer, &entry[index]);
        pj_get_timestamp(&t2);
        if (rc > 0) {
            cancelled += rc;
            t_cancel.u32.lo += (t2.u32.lo - t1.u32.lo);
        }
    } while (rc > 0);
    pj_gettimeofday(&now);
    pj_get_timestamp(&t1);
    rc = pj_timer_heap_poll(timer, NULL);
pj_get_timestamp(&t2);
if (rc > 0) {
    done += rc;
    t_poll.u32.lo += (t2.u32.lo - t1.u32.lo);
}

} while (PJ_TIME_VAL_LTE(now, expire) && pj_timer_heap_count(timer) > 0);

if (pj_timer_heap_count(timer)) {
    PJ_LOG(3, (THIS_FILE, "ERROR: %d timers left",
               pj_timer_heap_count(timer)));
    ++err;
}

t_sched.u32.lo /= count;
t_cancel.u32.lo /= count;
t_poll.u32.lo /= count;
PJ_LOG(4, (THIS_FILE,
          "...ok (count:%d, early:%d, cancelled:%d, "
          "sched:%d, cancel:%d poll:%d)",
          count, early, cancelled, t_sched.u32.lo, t_cancel.u32.lo,
          t_poll.u32.lo));

count = count * 2;
if (count > MAX_COUNT)
    break;

pj_pool_release(pool);
return err;

int timer_test()
{
    return test_timer_heap();
}

/* To prevent warning about "translation unit is empty"
 * when this test is disabled.
 */
int dummy_timer_test;
#endif /* INCLUDE_TIMER_TEST */
9.21 Test: Timestamp

This file provides implementation of timestamp_test()

9.21.1 Scope of the Test

This tests whether timestamp API works.

API tested:

- `pj_get_timestamp_freq()`
- `pj_get_timestamp()`
- `pj_elapsed_usec()`
- `PJ_LOG()`

This file is `pjlib-test/timestamp.c`

```c
#include "test.h"
#include <pj/os.h>
#include <pj/log.h>
#include <pj/rand.h>

#if INCLUDE_TIMESTAMP_TEST
#define THIS_FILE "timestamp"

static int timestamp_accuracy()
{
    pj_timestamp freq, t1, t2;
    pj_time_val tv1, tv2, tvtmp;
    pj_uint32_t msec, tics;
    pj_int64_t diff;
    PJ_LOG(3, (THIS_FILE, "...testing frequency accuracy (pls wait)");
    pj_get_timestamp_freq(&freq);
    /* Get the start time */
    pj_gettimeofday(&tv1);
    do {
        pj_gettimeofday(&tv1);
        pj_gettimeofday(&tv2);
        pj_gettimeofday(&tv3);
        msec = tv2.tv_usec - tv1.tv_usec;
        tics = tv2.tv_nsec - tv1.tv_nsec;
        diff = msec*1000 + tics/1000;
        PJ_LOG(3, (THIS_FILE, "frequency accuracy = ", diff, " usec");
    } while (diff < 1000000); // wait 1 second
    PJ_LOG(3, (THIS_FILE, "frequency accuracy = ", diff, " usec");
    return 0;
}
#endif
```

/* $Id: timestamp.c 974 2007-02-19 01:13:53Z bennylp $ */
/* Copyright (C)2003-2007 Benny Prijono <benny@prijono.org>
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA */
#include "test.h"
#include <pj/os.h>
#include <pj/log.h>
#include <pj/rand.h>

#if INCLUDE_TIMESTAMP_TEST
#define THIS_FILE "timestamp"

static int timestamp_accuracy()
{
    pj_timestamp freq, t1, t2;
    pj_time_val tv1, tv2, tvtmp;
    pj_uint32_t msec, tics;
    pj_int64_t diff;
    PJ_LOG(3, (THIS_FILE, "...testing frequency accuracy (pls wait")));
    pj_get_timestamp_freq(&freq);
    /* Get the start time */
    pj_gettimeofday(&tv1);
    do {
        pj_gettimeofday(&tv1);
        pj_gettimeofday(&tv2);
        pj_gettimeofday(&tv3);
        msec = tv2.tv_usec - tv1.tv_usec;
        tics = tv2.tv_nsec - tv1.tv_nsec;
        diff = msec*1000 + tics/1000;
        PJ_LOG(3, (THIS_FILE, "frequency accuracy = ", diff, " usec");
    } while (diff < 1000000); // wait 1 second
    PJ_LOG(3, (THIS_FILE, "frequency accuracy = ", diff, " usec");
    return 0;
}
#endif
while (PJ_TIME_VAL_EQ(tvtmp, tv1));
} while (PJ_TIME_VAL_EQ(tvtmp, tv1));
/* Sleep for 5 seconds */
pj_thread_sleep(10000);
/* Get end time */
pj_gettimeofday(&tvtmp);
} do {
pj_gettimeofday(&tv2);
} while (PJ_TIME_VAL_EQ(tvtmp, tv2));
pj_get_timestamp(&t2);
/* Get the elapsed time */
PJ_TIME_VAL_SUB(tv2, tv1);
msec = PJ_TIME_VAL_MSEC(tv2);
/* Check that the frequency match the elapsed time */
tics = (unsigned)(t2.u64 - t1.u64);
diff = tics - (msec * freq.u64 / 1000);
if (diff < 0)
  diff = -diff;
/* Only allow 1 msec mismatch */
if (diff > (pj_int64_t)(freq.u64 / 1000)) {
PJ_LOG(3,(THIS_FILE, "....error: timestamp drifted by %d usec after "
            "%d msec",
            (pj_uint32_t)(diff * 1000000 / freq.u64),
            msec));
  return -2000;
} else if (diff > (pj_int64_t)(freq.u64 / 1000000)) {
PJ_LOG(3,(THIS_FILE, "....warning: timestamp drifted by %d usec after "
            "%d msec",
            (pj_uint32_t)(diff * 1000000 / freq.u64),
            msec));
} else {
PJ_LOG(3,(THIS_FILE, "....good. Timestamp is accurate down to"
            " nearest usec.");
}
return 0;
}

int timestamp_test(void)
{
  enum { CONSECUTIVE_LOOP = 100 };
  volatile unsigned i;
  pj_timestamp freq, t1, t2;
  pj_time_val tv1, tv2;
  unsigned elapsed;
  pj_status_t rc;
  PJ_LOG(3,(THIS_FILE, "...Testing timestamp (high res time)"));

  /* Get and display timestamp frequency. */
  if ((rc=pj_get_timestamp_freq(&freq)) != PJ_SUCCESS) {
    app_perror("...ERROR: get timestamp freq", rc);
    return -1000;
  }
  PJ_LOG(3,(THIS_FILE, "....frequency: hiword=%lu loword=%lu",
            freq.u32_hi, freq.u32_lo));
  PJ_LOG(3,(THIS_FILE, "...checking if time can run backwards (pls wait)...");
}
/* Check if consecutive readings should yield timestamp value
 * that is bigger than previous value.
 * First we get the first timestamp.
 */
rc = pj_get_timestamp(&t1);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR: pj_get_timestamp", rc);
    return -1001;
}
rc = pj_gettimeofday(&tv1);
if (rc != PJ_SUCCESS) {
    app_perror("...ERROR: pj_gettimeofday", rc);
    return -1002;
}
for (i=0; i<CONSECUTIVE_LOOP; ++i) {
    pj_thread_sleep(pj_rand() % 100);
    rc = pj_get_timestamp(&t2);
    if (rc != PJ_SUCCESS) {
        app_perror("...ERROR: pj_get_timestamp", rc);
        return -1003;
    }
    rc = pj_gettimeofday(&tv2);
    if (rc != PJ_SUCCESS) {
        app_perror("...ERROR: pj_gettimeofday", rc);
        return -1004;
    }
    /* compare t2 with t1, expecting t2 >= t1. */
    if (t2.u32.hi < t1.u32.hi ||
        (t2.u32.hi == t1.u32.hi && t2.u32.lo < t1.u32.lo))
        PJ_LOG(3,(THIS_FILE, "...ERROR: timestamp run backwards!");
        return -1005;
    /* compare tv2 with tv1, expecting tv2 >= tv1. */
    if (PJ_TIME_VAL_LT(tv2, tv1)) {
        PJ_LOG(3,(THIS_FILE, "...ERROR: time run backwards!");
        return -1006;
    }
}
/* Simple test to time some loop.
 */
PJ_LOG(3,(THIS_FILE, "....testing simple 1000000 loop");

/* Mark start time. */
if ((rc=pj_get_timestamp(&t1)) != PJ_SUCCESS) {
    app_perror("....error: can't get timestamp", rc);
    return -1010;
}
/* Loop.. */
for (i=0; i<1000000; ++i) {
    /* Try to do something so that smart compilers wont
     * remove this silly loop.
     */
    null_func();
}
pj_thread_sleep(0);
/* Mark end time. */
pj_get_timestamp(&t2);

/* Get elapsed time in usec. */
elapsed = pj_elapsed_usec(&t1, &t2);
PJ_LOG(3,(THIS_FILE, "....elapsed: %u usec", (unsigned)elapsed));

/* See if elapsed time is "reasonable". */
* This should be good even on 50Mhz embedded powerpc.
*
if (elapsed < 1 || elapsed > 1000000) {
    PJ_LOG(3,(THIS_FILE, "....error: elapsed time outside window (%u, "
               "t1.u32 hi=%u, t1.u32 lo=%u, 
               "t2.u32 hi=%u, t2.u32 lo=%u)*,
               elapsed,
               t1.u32 hi, t1.u32 lo, t2.u32 hi, t2.u32 lo));
    return -1030;
}

/* Testing time/timestamp accuracy */
r = timestamp_accuracy();
if (rc != 0)
    return rc;

return 0;

#else
    /* To prevent warning about "translation unit is empty"
     * when this test is disabled. */
    int dummy_timestamp_test;
#endif /* INCLUDE_TIMESTAMP_TEST */
Index

__pj_throw__
    config.h, 223
_timer_id
    pj_timer_entry, 216
_timer_value
    pj_timer_entry, 216
addr
    pj_sockaddr, 211
addr_resolv.h, 219
Array helper., 19
array.h, 220
assert.h, 221
Assertion Macro, 21
atime
    pj_file_stat, 192
Atomic Variables, 89

Basic Data Types and Library Functionality., 173
block_alloc
    pj_pool_factory_policy, 206
block_free
    pj_pool_factory_policy, 206
block_list
    pj_pool_t, 208
buf
    pj_pool_block, 203
Build Configuration, 22

Caching Pool Factory, 121
callback
    pj_pool_factory_policy, 206
    pj_pool_t, 208
capacity
    pj_caching_pool, 188
    pj_pool_t, 207
cb
    pj_timer_entry, 216
color
    pj_rbtree_node, 210
comp
    pj_rbtree, 209
config.h, 222
    __pj_throw__, 223
    PJ_BEGIN_DECL, 223
PJ_DECL_DATA, 223
PJ_DECL_NO_RETURN, 223
PJ_DEF_DATA, 223
pj_dump_config, 224
PJ_END_DECL, 223
PJ_EXPORT_SYMBOL, 223
pj_get_version, 224
PJ_IDECL, 223
PJ_IDEF, 223
PJ INLINE, 224
PJ_TODO, 224
PJ_UNUSED_ARG, 224
PJ_VERSION, 224
create_pool
    pj_pool_factory, 204
Critical sections., 98
ctime
    pj_file_stat, 192
ctype - Character Type, 27
ctype.h, 225
cur
    pj_pool_block, 203
data
    pj_fd_set_t, 191
Data Structure., 48
day
    pj_parsed_time, 201
doxxygen.h, 226
dst_addr
    pj_ip_route_entry, 199
dump_status
    pj_pool_factory, 205
end
    pj_pool_block, 203
entry
    pj_hash_iterator_t, 193
ermno.h, 227
    PJ_ERRNO_SPACE_SIZE, 228
    PJ_ERRNO_START, 228
    PJ_ERRNO_START_STATUS, 228
    PJ_ERRNO_START_SYS, 228
    PJ_ERRNO_START_USER, 228
Error Codes, 31
Event Object., 101
except.h, 229
   PJ_CATCH, 229
   PJ_CATCH_ANY, 229
   PJ_END, 229
   PJ_GET_EXCEPTION, 229
   pj_pop_exception_handler, 230
   pj_push_exception_handler, 230
   PJ_THROW, 230
   pj_throw_exception, 230
   PJ_TRY, 230
   PJ_USE_EXCEPTION, 230

Exception Handling, 38

factory
   pj_caching_pool, 188
   pj_pool_t, 207
factory_data
   pj_pool_t, 207
Fast Memory Pool, 109
FD_SETSIZE
   pj_config, 22
File Access, 42
File I/O, 44
file_access.h, 231
file_io.h, 232
flags
   pj_pool_factory_policy, 206
free_list
   pj_caching_pool, 189
Globally Unique Identifier, 49
guid.h, 233

h_addr
   pj_addr_resolve, 18
h_addr_list
   pj_hostent, 194
h_addrtype
   pj_hostent, 194
h_aliases
   pj_hostent, 194
h_length
   pj_hostent, 194
h_name
   pj_hostent, 194
Hash Table, 51
hash.h, 234
hi
   pj_timestamp, 217
High Resolution Timestamp, 104
hour
   pj_parsed_time, 201
id
   pj_timer_entry, 216
if_addr
   pj_ip_route_entry, 199
in6_u
   pj_in6_addr, 195
increment_size
   pj_pool_t, 207
index
   pj_hash_iterator_t, 193
Input/Output, 55
int
   PJ_THREAD, 83
internal_
   pj_iq_queue_op_key_t, 198
ioqueue.h, 235
IOQueue: I/O Event Dispatching with Proactor Pattern, 56
IP Interface and Routing Helper, 67
ip_helper.h, 237
ipv4
   pj_ip_route_entry, 199
   pj_sockaddr, 211
ipv6
   pj_sockaddr, 211
key
   pj_rbtree_node, 210
Linked List, 68
list.h, 238
lo
   pj_timestamp, 217
lock
   pj_caching_pool, 189
Lock Objects, 73
lock.h, 239
log.h, 240
   pj_log_1, 242
   pj_log_2, 242
   pj_log_3, 242
   pj_log_4, 242
   pj_log_wrapper_1, 241
   pj_log_wrapper_2, 241
   pj_log_wrapper_3, 241
   pj_log_wrapper_4, 241
   pj_log_wrapper_5, 241
   pj_log_wrapper_6, 241
Logging Facility, 77
mask
   pj_ip_route_entry, 199
max_capacity
   pj_caching_pool, 188
Memory Pool Object, 113
INDEX

min
  pj_parsed_time, 201
Miscellaneous, 76
mon
  pj_parsed_time, 201
msec
  pj_parsed_time, 201
  pj_time_val, 215
mtime
  pj_file_stat, 192
Mutexes., 92

Network Address Resolution, 17
next
  pj_list, 200
null
  pj_rbtree, 209
  null_node
    pj_rbtree, 209
obj_name
  pj_pool_t, 207
on_accept_complete
  PJ_IOQUEUE, 66
on_block_alloc
  pj_pool_factory, 205
on_block_free
  pj_pool_factory, 205
on_connect_complete
  PJ_IOQUEUE, 66
on_read_complete
  pj_ioqueue_callback, 197
on_write_complete
  PJ_IOQUEUE, 66
Operating System Dependent Functionality., 81
os.h, 243
  pj_thread_init, 245
parent
  pj_rbtree_node, 210
peak_used_size
  pj_caching_pool, 188
PJ Library, 172
pj_add_timestamp
  PJ_TIMESTAMP, 104
pj_add_timestamp32
  PJ_TIMESTAMP, 104
pj_addr_hdr, 187
  sa_family, 187
pj_addr_resolve
  h_addr, 18
  pj_gethostbyname, 18
  pj_gethostip, 18
PJ_AF_INET
pj_AF_INET
  PJ.SOCK, 145
pj_AF_INET6
  PJ.SOCK, 145
PJ_AF_INET6
  PJ.SOCK, 134
PJ_AF_IRDA
  PJ.SOCK, 145
pj_AF_IRDA
  PJ.SOCK, 134
PJ_AF_LOCAL
  PJ.SOCK, 132
PJ_AF_PACKET
  PJ.SOCK, 145
pj_AF_PACKET
  PJ.SOCK, 134
PJ_AF_UNIX
  PJ.SOCK, 146
pj_AF_UNIX
  PJ.SOCK, 134
pj_ansi_to_unicode
  PJ_UNICODE, 181
PJ_ARRAY
  pj_array_erase, 19
  pj_array_find, 19
  pj_array_insert, 19
pj_array_erase
  PJ_ARRAY, 19
pj_array_find
  PJ_ARRAY, 19
pj_array_insert
  PJ_ARRAY, 19
PJ_ARRAY_SIZE
  PJ_BASIC, 174
pj_assert
  pj_assert, 21
    PJ_ASSERT_ON_FAIL, 21
    PJ_ASSERT_RETURN, 21
PJ_ASSERT_ON_FAIL
  pj_assert, 21
PJ_ASSERT_RETURN
  pj_assert, 21
pj_atexit
  PJ_BASIC, 178
PJ_ATOMIC
  pj_atomic_add, 89
  pj_atomic_add_and_get, 89
  pj_atomic_create, 89
  pj_atomic_dec, 90
  pj_atomic_dec_and_get, 90
  pj_atomic_destroy, 90
  pj_atomic_get, 90
  pj_atomic_inc, 91
PJ_CATCH
    except.h, 229
PJ_CATCH_ANY
    except.h, 229
pj_char_t
    PJ_BASIC, 175
PJ_CHECK_STACK
    PJ_THREAD, 83
pj_cmp_timestamp
    PJ_TIMESTAMP, 105
pj_color_t
    PJ_BASIC, 175
pj_config
    FD_SETSIZE, 22
    PJ_DEBUG, 22
    PJ_ENABLE_EXTRA_CHECK, 22
    PJ_EXCEPTION_USE_WIN32_SEH, 23
    PJ_FUNCTIONS_ARE_INLINED, 23
    PJ_HAS_ERROR_STRING, 23
    PJ_HAS_EXCEPTION_NAMES, 23
    PJ_HAS_FLOATING_POINT, 23
    PJ_HAS_POOL_ALT_API, 23
    PJ_IQUEUE_HAS_SAFE_UNREG, 24
    PJ_IQUEUE_KEY_FREE_DELAY, 24
    PJ_IQUEUE_MAX_HANDLES, 24
    PJ_LOG_MAX_SIZE, 24
    PJ_LOG_USE_STACK_BUFFER, 24
    PJ_MAX_EXCEPTION_ID, 25
    PJ_MAX_HOSTNAME, 25
    PJ_MAXPATH, 25
    PJ_NATIVE_ERR_POSITIVE, 25
    PJ_POOL_DEBUG, 25
    PJ_SAFE_POOL, 25
    PJ_TERM_HAS_COLOR, 25
    PJ_TIMESTAMP_USE_RDTSC, 26
pj_create_random_string
    PJ_PSTR, 155
pj_create_unique_string
    PJ_GUID, 49
PJ_CRIT_SEC
    pj_enter_critical_section, 98
    pj_leave_critical_section, 98
pj_cstr
    PJ_PSTR, 155
pj_ctype
    pj_hex_digit_to_val, 27
    pj_hex_digits, 27
    pj_isalnum, 27
    pj_isalpha, 28
    pj_isascii, 28
    pj_isblank, 28
    pj_isdigit, 28
    pj_islower, 29
    pj_isspace, 29
    pj_isupper, 29
    pj_isxdigit, 29
    pj_tolower, 30
    pj_toupper, 30
    pj_val_to_hex_digit, 30
PJ_DEBUG
    pj_config, 22
PJ_DECL_ANSI_TEMP_BUF
    PJ_UNICODE, 180
PJ_DECL_DATA
    config.h, 223
    PJ_GUID, 49
    PJ_POOL_FACTORY, 119
PJ_DECL_LIST_MEMBER
    PJ_LIST, 68
    pj_pool_block, 203
    pj_pool_t, 207
PJDECL_NO_RETURN
    config.h, 223
PJ_DECL_UNICODE_TEMP_BUF
    PJ_UNICODE, 180
PJ_DEF_DATA
    config.h, 223
    pj_dump_config
    config.h, 224
PJ_EBUG
    pj_errno, 35
PJ_EBUSY
    pj_errno, 35
PJ_ECANCELLED
    pj_errno, 35
PJ_EEOF
    pj_errno, 35
PJ_EEXISTS
    pj_errno, 36
PJ EINVAL
    pj_errno, 36
PJ EINVALARG
    pj_errno, 36
pj_elapsed_cycle
    PJ_TIMESTAMP, 105
pj_elapsed_msec
    PJ_TIMESTAMP, 105
pj_elapsed_nanosec
    PJ_TIMESTAMP, 105
pj_elapsed_usec
    PJ_TIMESTAMP, 106
pj_elapsed_cycle
    PJ_TIMESTAMP, 106
pj_elapsed_time
    PJ_TIMESTAMP, 106
PJ_ENABLE_EXTRA_CHECK
    pj_config, 22
PJ_ENDIANOOLONG
pj_errnum, 36
PJ_END
except.h, 229
PJ_ENDDECL
config.h, 223
PJ_ENOMEM
pj_errnum, 36
PJ_ENOTFOUND
pj_errnum, 36
PJ_ENOTSUP
pj_errnum, 36
pj_enter_critical_section
pj_errno, 32
PJ_CRITICAL_SEC, 98
pj_enum_ip_interface
pj_ip_helper, 67
pj_enum_ip_route
pj_ip_helper, 67
PJ_EPENDING
pj_errnum, 36
PJ_ERRNO_SPACE_SIZE
pj_errno, 32
PJ_BUILD_ERR, 35
PJ_BUG, 35
PJ_BUSY, 35
PJ_ECANCELLED, 35
PJ_EOF, 35
PJ_EEXIST, 36
PJEINVAL, 36
PJ_INVALIDOP, 36
PJ_ENAMETOOLONG, 36
PJ EINVAL, 36
PJ_ENOTFOUND, 36
PJ_ENOTSUP, 36
PJ_EPENDING, 36
PJ_ERESOLVE, 36
PJETIMEOUT, 36
PJ_ETOOBIG, 36
PJ_ETOOMANY, 37
PJ_ETOOMANYCONN, 37
PJ_ETOOSMAL, 37
PJ_EUNKNOWN, 37
PJETIMEOUT
pj_errno, 36
PJ_ETOOBIG
pj_errno, 36
PJ_ETOOMANY
pj_errno, 36
PJ_ETOOMANYCONN
pj_errno, 36
PJ_ETOOSMAL
pj_errno, 36
PJ_EUNKNOWN
pj_errno, 36
PJ_EVENT
pj_event_create, 101
pj_event_destroy, 101
pj_event_pulse, 101
pj_event_reset, 102
pj_event_set, 102
pj_event_trywait, 102
pj_event_wait, 102
pj_errno, 32
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_CONT
PJ_EVENT,
PJ_EXCEPT, 41
pj_exception_id_name
  PJ_EXCEPT, 41
pj_exception_id_t
  PJ_BASIC, 175
pj_exception_state_t, 190
  prev, 190
  state, 190
PJ_EXCEPTION_USE_WIN32_SEH
  pj_config, 23
pj_exit_callback
  PJ_BASIC, 175
PJ_EXPORT_SYMBOL
  config.h, 223
PJ_FALSE
  PJ_BASIC, 174
PJ_FD_CLR
  PJ.SOCK_SELECT, 150
PJ_FD_COUNT
  PJ.SOCK_SELECT, 150
PJ_FD_ISSET
  PJ.SOCK_SELECT, 151
PJ_FD_SET
  PJ.SOCK_SELECT, 151
  pj_fd_set_t, 191
  data, 191
PJ_FD_ZERO
  PJ.SOCK_SELECT, 151
PJ_FILE_ACCESS
  pj_file_delete, 42
  pj_file_exists, 42
  pj_file_getstat, 42
  pj_file_move, 42
  pj_file_size, 43
pj_file_access
  PJ_FILE_IO, 44
pj_file_close
  PJ_FILE_IO, 45
pj_file_delete
  PJ_FILE_ACCESS, 42
pj_file_exists
  PJ_FILE_ACCESS, 42
pj_file_flush
  PJ_FILE_IO, 45
pj_file_getpos
  PJ_FILE_IO, 45
pj_file_getstat
  PJ_FILE_ACCESS, 42
pj_file_open
  PJ_FILE_IO, 45
pj_file_read
  PJ_FILE_IO, 46
pj_file_seek_type
  PJ_FILE_IO, 44
pj_file_setpos
  PJ_FILE_IO, 46
pj_file_size
  PJ_FILE_ACCESS, 43
pj_file_stat, 192
  atime, 192
  ctime, 192
  mtime, 192
  size, 192
pj_file_write
  PJ_FILE_IO, 46
PJ_FUNCTIONS_ARE_INLINED
  pj_config, 23
pj_generate_unique_string
  PJ_GUID, 49
PJ_GET_EXCEPTION
  except.h, 229
pj_get_netos_error
  pj_errno, 33
pj_get_os_error
  pj_errno, 33
pj_get_timestamp
  PJ_TIMESTAMP, 107
pj_get_timestamp_freq
  PJ_TIMESTAMP, 107
pj_get_version
  config.h, 224
pj_gethostaddr
  PJ.SOCK, 134
pj_gethostbyname
  pj_addr_resolve, 18
pj_gethostip
  pj_addr_resolve, 18
pj_gethostname
PJ_SOCK, 134
pj_getpid
PJ_THREAD, 83
pj_gettimeofday
PJ_TIME, 184
PJ_GUID
  pj_create_unique_string, 49
  PJDECL_DATA, 49
  pj_generate_unique_string, 49
  PJ_GUID_MAX_LENGTH, 49
  pj_GUID_STRING_LENGTH, 50
PJ_GUID_MAX_LENGTH
  PJ_GUID, 49
pj_GUID_STRING_LENGTH
  PJ_GUID, 50
PJ_HAS_ERROR_STRING
  pj_config, 23
PJ_HAS_EXCEPTION_NAMES
  pj_config, 23
PJ_HAS_FLOATING_POINT
  pj_config, 23
PJ_HAS_POOL_ALT_API
  pj_config, 23
PJ_HAS_STRICMP_ALNUM
  pj_config, 23
PJ_HASH
  pj_hash_calc, 52
  pj_hash_calc_tolower, 52
  pj_hash_count, 52
  pj_hash_create, 52
  pj_hash_entry_buf, 51
  PJ_HASH_ENTRY_BUF_SIZE, 51
  pj_hash_first, 52
  pj_hash_get, 53
  PJ_HASH_KEY_STRING, 51
  pj_hash_next, 53
  pj_hash_set, 53
  pj_hash_set_np, 54
  pj_hash_this, 54
  pj_hash_table_t,
  PJ_BASIC, 175
  pj_hash_this
  PJ_HASH, 54
  pj_hex_digit_to_val
  PJ_HASH, 27
  pj_hex_digits
  PJ_HASH, 27
  pj_hostent, 194
  h_addr_list, 194
  h_addrtype, 194
  h_aliases, 194
  h_length, 194
  h_name, 194
  pj_htonl
  PJ_SOCK, 134
  pj_htons
  PJ_SOCK, 134
PJ_IDECL
  config.h, 223
PJ_IDEF
  config.h, 223
  pj_in6_addr, 195
  in6_u, 195
  u6_addr16, 195
  u6_addr32, 195
  u6_addr8, 195
PJ_INET6ADDR_ANY_INIT
  PJ_SOCK, 132
PJ_INET6ADDR_LOOPBACK_INIT
  PJ_SOCK, 133
  pj_in_addr, 196
  s_addr, 196
PJ_INETADDR_ANY
  PJ_SOCK, 133
PJ_INETADDR_BROADCAST
  PJ_SOCK, 133
PJ_INETADDR_NONE
  PJ_SOCK, 133
pj_inet_addr
INDEX

PJ_LIST, 70
pj_list_merge_last
PJ_LIST, 70
pj_list_push_back
PJ_LIST, 71
pj_list_push_front
PJ_LIST, 71
pj_list_search
PJ_LIST, 71
pj_list_size
PJ_LIST, 71
pj_list_type
PJ_BASIC, 176

PJ_LOCK
    pj_lock_acquire, 73
    pj_lock_create_null_mutex, 73
    pj_lock_create_recursive_mutex, 73
    pj_lock_create_semaphore, 74
    pj_lock_create_simple_mutex, 74
    pj_lock_destroy, 74
    pj_lock_release, 75
    pj_lock_tryacquire, 75
pj_lock_acquire
    PJ_LOCK, 73
pj_lock_create_null_mutex
    PJ_LOCK, 73
pj_lock_create_recursive_mutex
    PJ_LOCK, 73
pj_lock_create_semaphore
    PJ_LOCK, 74
pj_lock_create_simple_mutex
    PJ_LOCK, 74
pj_lock_destroy
    PJ_LOCK, 74
pj_lock_release
    PJ_LOCK, 75
pj_lock_t
    PJ_BASIC, 176
pj_lock_tryacquire
    PJ_LOCK, 75

PJ_LOG
    PJ_LOG_HAS_CR, 79
    PJ_LOG_HAS_DAY_NAME, 78
    PJ_LOG_HAS_DAY_OF_MON, 78
    PJ_LOG_HAS_MICRO_SEC, 78
    PJ_LOG_HAS_MONTH, 78
    PJ_LOG_HAS_NEWLINE, 79
    PJ_LOG_HAS_SENDER, 79
    PJ_LOG_HAS_TIME, 78
    PJ_LOG_HAS_YEAR, 78
pj_log
    PJ_LOG, 79
pj_log_1
        log.h, 242
pj_log_2
        log.h, 242
pj_log_3
        log.h, 242
pj_log_4
        log.h, 242
pj_log_decoration
    PJ_LOG, 78
pj_log_func
    PJ_LOG, 78
pj_log_get_decor
    PJ_LOG, 79
pj_log_get_level
    PJ_LOG, 79
pj_log_get_log_func
    PJ_LOG, 79
    PJ_LOG_HAS_CR
    PJ_LOG, 79
    PJ_LOG_HAS_DAY_NAME
    PJ_LOG, 79
    PJ_LOG_HAS_DAY_OF_MON
    PJ_LOG, 79
    PJ_LOG_HAS_MICRO_SEC
    PJ_LOG, 79
    PJ_LOG_HAS_MONTH
    PJ_LOG, 79
    PJ_LOG_HAS_NEWLINE
    PJ_LOG, 79
    PJ_LOG_HAS_SENDER
    PJ_LOG, 79
    PJ_LOG_HAS_TIME
    PJ_LOG, 79
    PJ_LOG_HAS_YEAR
    PJ_LOG, 79
pj_log_set_decor
    PJ_LOG, 79
pj_log_set_level
    PJ_LOG, 79
pj_log_set_log_func
    PJ_LOG, 80

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
PJ_LOG_USE_STACK_BUFFER
  pj_config, 24
pj_log_wrapper_1
  log.h, 241
pj_log_wrapper_2
  log.h, 241
pj_log_wrapper_3
  log.h, 241
pj_log_wrapper_4
  log.h, 241
pj_log_wrapper_5
  log.h, 241
pj_log_wrapper_6
  log.h, 241
pj_log_write
  PJ_LOG, 80
PJ_MAX_EXCEPTION_ID
  pj_config, 25
PJ_MAX_HOSTNAME
  pj_config, 25
PJ_MAX_OBJ_NAME
  PJ_BASIC, 174
PJ_MAXINT32
  PJ_BASIC, 174
PJ_MAXPATH
  pj_config, 25
pj_memchr
  PJ_PSTR, 156
pj_memcmp
  PJ_PSTR, 156
pj_memcpy
  PJ_PSTR, 156
pj_memmove
  PJ_PSTR, 156
pj_memset
  PJ_PSTR, 157
PJ_MSG_DONTROUTE
  PJ_SOCK, 147
pj_MSG_DONTROUTE
  PJ_SOCK, 136
PJ_MSG_OOB
  PJ_SOCK, 147
pj_MSG_OOB
  PJ_SOCK, 136
PJ_MSG_PEEK
  PJ_SOCK, 147
pj_MSG_PEEK
  PJ_SOCK, 136
PJ_MUTEX
  pj_mutex_create, 93
  pj_mutex_create_recursive, 93
  pj_mutex_create_simple, 93
  pj_mutex_destroy, 93
  pj_mutex_is_locked, 92
  pj_mutex_lock, 94
  pj_mutex_trylock, 94
  pj_mutex_type_e, 92
  pj_mutex_unlock, 94
pj_mutex_create
  PJ_MUTEX, 93
pj_mutex_create_recursive
  PJ_MUTEX, 93
pj_mutex_create_simple
  PJ_MUTEX, 93
pj_mutex_destroy
  PJ_MUTEX, 93
pj_mutex_is_locked
  PJ_MUTEX, 92
pj_mutex_lock
  PJ_MUTEX, 94
pj_mutex_t
  PJ_BASIC, 176
pj_mutex_trylock
  PJ_MUTEX, 94
pj_mutex_type_e
  PJ_MUTEX, 92
pj_mutex_unlock
  PJ_MUTEX, 94
PJ_NATIVE_ERR_POSITIVE
  pj_config, 25
PJ_NATIVE_TO_STRING
  PJ_UNICODE, 180
pj_NO_MEMORY_EXCEPTION
  PJ_POOL_FACTORY, 119
pj_ntohl
  PJ_SOCk, 137
pj_ntohs
  PJ_SOCk, 137
PJ_O_APPEND
  PJ_FILE_IO, 44
PJ_O_RDONLY
  PJ_FILE_IO, 44
PJ_O_RDWR
  PJ_FILE_IO, 44
PJ_O_WRONLY
  PJ_FILE_IO, 44
pj_off_t
  PJ_BASIC, 176
pj_oschandle_t
  PJ_BASIC, 176
pj_parsed_time, 201
day, 201
hour, 201
min, 201
mon, 201
msec, 201
sec, 201
wday, 201

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
INDEX

pj_rwlockmutex_unlock_read, 96
pj_rwlockmutex_unlock_write, 96
pj_rwlockmutex_create
  PJ_RW_MUTEX, 95
pj_rwlockmutex_destroy
  PJ_RW_MUTEX, 95
pj_rwlockmutex_lock_read
  PJ_RW_MUTEX, 96
pj_rwlockmutex_lock_write
  PJ_RW_MUTEX, 96
pj_rwlockmutex_t
  PJ_RW_MUTEX, 95
pj_rwlockmutex_unlock_read
  PJ_RW_MUTEX, 96
pj_rwlockmutex_unlock_write
  PJ_RW_MUTEX, 96
PJ_SAFE_POOL
  pj_config, 25
PJ_SD_BOTH
  PJ_SOCK, 133
PJ_SD_RECEIVE
  PJ_SOCK, 133
PJ_SD_SEND
  PJ_SOCK, 133
PJ_SEEK_CUR
  PJ_FILE_IO, 45
PJ_SEEK_END
  PJ_FILE_IO, 45
PJ_SEEK_SET
  PJ_FILE_IO, 45
PJ_SEM
  pj_sem_create, 99
  pj_sem_destroy, 99
  pj_sem_post, 99
  pj_sem_trywait, 100
  pj_sem_wait, 100
pj_sem_create
  PJ_SEM, 99
pj_sem_destroy
  PJ_SEM, 99
pj_sem_post
  PJ_SEM, 99
pj_sem_t
  PJ_BASIC, 177
pj_sem_trywait
  PJ_SEM, 100
pj_sem_wait
  PJ_SEM, 100
pj_set_nertos_error
  pj_errno, 34
pj_set_os_error
  pj_errno, 34
pj_set_timestamp32
  PJ_TIMESTAMP, 107
PJ_SHUT_RD
  PJ_SOCK, 133
PJ_SHUT_RDWR
  PJ_SOCK, 133
PJ_SHUT_WR
  PJ_SOCK, 133
pj_shutdown
  PJ_BASIC, 178
pj_size_t
  PJ_BASIC, 177
PJ_SO_RCVBUF
  PJ_SOCK, 147
pj_SO_RCVBUF
  PJ_SOCK, 137
PJ_SO_SNDBUF
  PJ_SOCK, 147
pj_SO_SNDBUF
  PJ_SOCK, 137
PJ_SO_TYPE
  PJ_SOCK, 137
pj_SO_TYPE
  PJ_SOCK, 137
PJ_SOCK
  PJ_SD_BOTH, 133
  PJ_SD_RECEIVE, 133
  PJ_SD_SEND, 133
  PJ_SHUT_RD, 133
  PJ_SHUT_RDWR, 133
  PJ_SHUT_WR, 133
PJ_AF_INET
  PJ_AF_INET, 134
pj_AF_INET
  PJ_AF_INET, 134
PJ_AF_INET6
  PJ_AF_INET6, 134
pj_AF_INET6
  PJ_AF_INET6, 134
PJ_AF_IRDA
  PJ_AF_IRDA, 134
pj_AF_IRDA
  PJ_AF_IRDA, 134
PJ_AF_LOCAL
  PJ_AF_LOCAL, 132
pj_AF_PACKET
  PJ_AF_PACKET, 145
pj_AF_PACKET
  PJ_AF_PACKET, 134
PJ_AF_UNIX
  PJ_AF_UNIX, 146
pj_AF_UNIX
  PJ_AF_UNIX, 134
pj_gethostaddr
  134
pj_gethostname
  134
pj_htonl
  134
pj_htons
  134
PJ_INET6ADDR_ANY_INIT
  132
PJ_INET6ADDR_LOOPBACK_INIT
  133
PJ_INETADDR
  133
PJ_INETADDR_BROADCAST
  133
PJ_INETADDR_NONE
  133
pj_inet_addr
  135
pj_inet_addr2
  135
pj_inet_aton
  135
pj_inet_ntoa
  136
INDEX

PJ_INVALID_SOCKET, 133
PJ_IP_TOS, 146
pj_IP_TOS, 136
PJ_IPTOS_LOWDELAY, 146
pj_IPTOS_LOWDELAY, 136
PJ_IPTOS_MINCOST, 136
pj_IPTOS_MINCOST, 136
PJ_IPTOS_RELIABILITY, 146
pj_IPTOS_RELIABILITY, 136
PJ_IPTOS_THROUGHPUT, 146
pj_IPTOS_THROUGHPUT, 136
PJ_MSG_DONTROUTE, 147
pj_MSG_DONTROUTE, 136
PJ_MSG_OOB, 147
pj_MSG_OOB, 136
PJ_MSG_PEEK, 147
pj_MSG_PEEK, 136
pj_ntohl, 137
pj_ntohs, 137
PJ_SO_RCVBUF, 147
pj_SO_RCVBUF, 137
PJ_SO_SNDBUF, 147
pj_SO_SNDBUF, 137
PJ_SO_TYPE, 147
pj_SO_TYPE, 137
pj_sock_accept, 137
pj_sock_bind, 138
pj_sock_bind_in, 138
pj_sock_close, 138
pj_sock_connect, 138
PJ_SOCK_DGRAM, 148
pj_SOCK_DGRAM, 139
pj_sock_getpeername, 139
pj_sock_getsockname, 139
pj_sock_getsockopt, 140
pj_sock_listen, 140
PJ_SOCK_RAW, 148
pj_SOCK_RAW, 140
PJ_SOCK_RDM, 148
pj_SOCK_RDM, 140
pj_sock_recv, 140
pj_sock_recvfrom, 141
pj_sock_send, 141
pj_sock_sendto, 141
pj_sock_setsockopt, 142
pj_sock_shutdown, 142
pj_sock_socket, 142
PJ_SOCK_STREAM, 148
pj_sockstream_get_addr, 143
pj_sockstream_get_port, 143
pj_sockstream_init, 143
pj_sockstream_setopt, 144
pj_sockstream_setopt, 144
pj_sockaddr_in_set_str_addr, 144
pj_socket_sd_type, 133
PJ_SOL_IP, 148
pj_SOL_IP, 144
PJ_SOL_IPV6, 148
pj_SOL_IPV6, 145
PJ_SOL_SOCKET, 149
pj_SOL_SOCKET, 145
PJ_SOL_TCP, 149
pj_SOL_TCP, 145
PJ_SOL_UDP, 149
pj_SOL_UDP, 145
PJ_SOMAXCONN, 133
pj_sock_accept
PJ_SOCK, 137
pj_sock_bind
PJ_SOCK, 138
pj_sock_bind_in
PJ_SOCK, 138
pj_sock_close
PJ_SOCK, 138
pj_sock_connect
PJ_SOCK, 138
PJ_SOCK_DGRAM
pj_SOCK_DGRAM
PJ_SOCK, 139
pj_sock_getpeername
PJ_SOCK, 139
pj_sock_getsockopt
PJ_SOCK, 139
pj_sock_getsockopt
PJ_SOCK, 140
pj_sock_listen
PJ_SOCK, 140
PJ_SOCK_RAW
pj_SOCK_RAW
PJ_SOCK, 148
pj_sock_recv
PJ_SOCK, 140
pj_sock_recvfrom
PJ_SOCK, 140
pj_sock_select
PJ_SOCK, 141
PJ_SOCK_SELECT
PJ_FD_CLR, 150
PJ_FD_COUNT, 150
PJ_FD_ISSET, 151
PJ_FD_SET, 151
PJ_FD_ZERO, 151
pj_sock_select, 151
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pj_sock_select</td>
<td>151</td>
</tr>
<tr>
<td>PJ_SOCK_SELECT</td>
<td></td>
</tr>
<tr>
<td>pj_sock_send</td>
<td>141</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sock_sendto</td>
<td>141</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sock_setssockopt</td>
<td>142</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sock_shutdown</td>
<td>142</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sock_socket</td>
<td>142</td>
</tr>
<tr>
<td>PJ_SOCK_STREAM</td>
<td>148</td>
</tr>
<tr>
<td>pj_sockaddr_in</td>
<td>212</td>
</tr>
<tr>
<td>PJ_SOCK_STREAM</td>
<td></td>
</tr>
<tr>
<td>pj_sock_t</td>
<td>177</td>
</tr>
<tr>
<td>PJ_BASIC</td>
<td></td>
</tr>
<tr>
<td>pj_sockaddr_in6</td>
<td>213</td>
</tr>
<tr>
<td>PJ_SOCK/tcp</td>
<td></td>
</tr>
<tr>
<td>sin_addr</td>
<td>212</td>
</tr>
<tr>
<td>sin_family</td>
<td>212</td>
</tr>
<tr>
<td>sin_port</td>
<td>212</td>
</tr>
<tr>
<td>sin_zero</td>
<td>212</td>
</tr>
<tr>
<td>pj_sockaddr_in6_get_addr</td>
<td>143</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sockaddr_in_port</td>
<td>143</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sockaddr_init</td>
<td>143</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sockaddr_in_set_addr</td>
<td>144</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sockaddr_in_set_port</td>
<td>144</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sockaddr_in_set_str_addr</td>
<td>144</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>pj_sock_addr_t</td>
<td>177</td>
</tr>
<tr>
<td>PJ_BASIC</td>
<td></td>
</tr>
<tr>
<td>pj_socket_sd_type</td>
<td>133</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>PJ_SOL_IPV6</td>
<td>148</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>PJ_SOL_IPV6</td>
<td>145</td>
</tr>
<tr>
<td>PJ_SOCK_STREAM</td>
<td>149</td>
</tr>
<tr>
<td>PJ_SOL_SOCKET</td>
<td></td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td>145</td>
</tr>
<tr>
<td>PJ_SOL_TCP</td>
<td>149</td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td></td>
</tr>
<tr>
<td>PJ_SOL_TCP</td>
<td>145</td>
</tr>
<tr>
<td>PJ_SOCK_UDP</td>
<td>149</td>
</tr>
<tr>
<td>PJ_SOL_UDP</td>
<td></td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td>145</td>
</tr>
<tr>
<td>PJ_SOL_UDP</td>
<td>145</td>
</tr>
<tr>
<td>PJ_SOMAXCONN</td>
<td></td>
</tr>
<tr>
<td>PJ_SOCK</td>
<td>133</td>
</tr>
<tr>
<td>pj_srand</td>
<td>125</td>
</tr>
<tr>
<td>PJ_RAND</td>
<td></td>
</tr>
<tr>
<td>pj_ssize_t</td>
<td>177</td>
</tr>
<tr>
<td>PJ_BASIC</td>
<td></td>
</tr>
<tr>
<td>PJ_STATUS_FROM_OS</td>
<td>32</td>
</tr>
<tr>
<td>pj_errno</td>
<td></td>
</tr>
<tr>
<td>PJ_STATUS_TO_OS</td>
<td>32</td>
</tr>
<tr>
<td>pj_errno</td>
<td></td>
</tr>
<tr>
<td>pj_str</td>
<td>157</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strbuf</td>
<td>157</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strcat</td>
<td>158</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strcat2</td>
<td>158</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strchr</td>
<td>158</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strcmp</td>
<td>158</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strcmp2</td>
<td>159</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strdup</td>
<td>158</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>pj_strdup</td>
<td>159</td>
</tr>
<tr>
<td>PJ_PSTR</td>
<td></td>
</tr>
<tr>
<td>Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen</td>
<td></td>
</tr>
</tbody>
</table>
pj_strdup2
PJ_PSTR, 160
pj_strdup2_with_null
PJ_PSTR, 160
pj_strdup3
PJ_PSTR, 160
pj_strdup_with_null
PJ_PSTR, 160
pj_strerror
pj_errno, 34
pj_stricmp
PJ_PSTR, 161
pj_stricmp2
PJ_PSTR, 161
pj_stricmp_alnum
PJ_PSTR, 154
PJ_STRING_TO_NATIVE
PJ_UNICODE, 180
pj_strlen
PJ_PSTR, 161
pj_strltrim
PJ_PSTR, 162
pj_strncmp
PJ_PSTR, 162
pj_strncmp2
PJ_PSTR, 162
pj_strncpy
PJ_PSTR, 163
pj_strncpy_with_null
PJ_PSTR, 163
pj_strnicmp
PJ_PSTR, 163
pj_strnicmp2
PJ_PSTR, 163
pj_strrtrim
PJ_PSTR, 164
pj_strset
PJ_PSTR, 164
pj_strset2
PJ_PSTR, 164
pj_strset3
PJ_PSTR, 164
pj_strtoul
PJ_PSTR, 165
pj_strtoul2
PJ_PSTR, 165
pj_strtrim
PJ_PSTR, 165
pj_sub_timestamp
PJ_TIMESTAMP, 108
pj_sub_timestamp32
PJ_TIMESTAMP, 108
PJ_SUCCESS
PJ_BASIC, 174
pj_swap16
PJ_BASIC, 179
pj_swap32
PJ_BASIC, 179
PJ_SYMBIAN_OS
pj_symbianos_poll, 87
pj_symbianos_poll
PJ_SYMBIAN_OS, 87
PJ_T
PJ_BASIC, 174
PJ_TERM_COLOR_B
PJTERM_COLOR_BRIGHT
PJTERM_COLOR_G
PJTERM_COLOR_R
PJ_TERM_HAS_COLOR
pj_config, 25
PJ_THREAD
int, 83
PJ_CHECK_STACK, 83
pj getpid, 83
pj_thread_create, 83
pj_thread_create_flags, 83
pj_thread_desc, 83
PJ_THREAD_DESC_SIZE, 83
pj_thread_destroy, 84
pj_thread_get_name, 84
pj_thread_get_stack_info, 83
pj_thread_get_stack_max_usage, 83
pj_thread_is_registered, 84
pj_thread_join, 84
pj_thread_register, 85
pj_thread_resume, 85
pj_thread_sleep, 85
pj_thread_this, 85
pj_thread_create
PJ_THREAD, 83
pj_thread_create_flags
PJ_THREAD, 83
PJ_THREAD_DEFAULT_STACK_SIZE
pj config, 25
pj_thread_desc
PJ_THREAD, 83
PJ_THREAD_DESC_SIZE
PJ_THREAD, 83
pj_thread_destroy
PJ_THREAD, 84
pj_thread_get_name
PJ_THREAD, 84
pj_thread_get_stack_info
PJ_THREAD, 84
INDEX 387

pj_thread_get_stack_max_usage
  PJ_THREAD, 83
pj_thread_init
  os.h, 245
pj_thread_is_registered
  PJ_THREAD, 84
pj_thread_join
  PJ_THREAD, 84
pj_thread_local_alloc
  PJ_TLS, 88
pj_thread_local_free
  PJ_TLS, 88
pj_thread_local_get
  PJ_TLS, 88
pj_thread_local_set
  PJ_TLS, 88
pj_thread_register
  PJ_THREAD, 85
pj_thread_resume
  PJ_THREAD, 85
pj_thread_sleep
  PJ_THREAD, 85
pj_thread_t
  PJ_BASIC, 177
pj_thread_this
  PJ_THREAD, 85
PJ_THROWS
  except.h, 230
pj_throw_exception_
  except.h, 230
PJ_TIME
  pj_gettimeofday, 184
  pj_time_decode, 184
  pj_time_encode, 185
  pj_time_gmt_to_local, 185
  pj_time_local_to_gmt, 185
  PJ_TIME_VAL_ADD, 182
  PJ_TIME_VAL_EQ, 182
  PJ_TIME_VAL_GT, 183
  PJ_TIME_VAL_GTE, 183
  PJ_TIME_VAL_LT, 183
  PJ_TIME_VAL_LTE, 183
  PJ_TIME_VAL_MSEC, 184
  pj_time_val_normalize, 185
  PJ_TIME_VAL_SUB, 185
pj_time_decode
  PJ_TIME, 184
pj_time_encode
  PJ_TIME, 185
pj_time_gmt_to_local
  PJ_TIME, 185
pj_time_local_to_gmt
  PJ_TIME, 185
pj_time_val, 215
msec, 215
sec, 215

PJ_TIMER
  pj_timer_entry_init, 168
  pj_timer_heap_callback, 168
  pj_timer_heap_cancel, 168
  pj_timer_heap_count, 168
  pj_timer_heap_create, 169
  pj_timer_heap_destroy, 169
  pj_timer_heap_earliest_time, 169
  pj_timer_heap_mem_size, 169
  pj_timer_heap_poll, 170
  pj_timer_heap_schedule, 170
  pj_timer_heap_set_lock, 170
  pj_timer_heap_set_max_timed_out_per_poll, 170
  pj_timer_id_t, 168
pj_timer_entry, 216
  _timer_id, 216
  _timer_value, 216
  cb, 216
  id, 216
  PJ_BASIC, 178
  user_data, 216
pj_timer_entry_init
  PJ_TIMER, 168
pj_timer_heap_callback
  PJ_TIMER, 168
pj_timer_heap_cancel
  PJ_TIMER, 168
pj_timer_heap_count
  PJ_TIMER, 168
pj_timer_heap_create
  PJ_TIMER, 169
pj_timer_heap_destroy
  PJ_TIMER, 169

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen
INDEX

s6_addr32
   sock.h, 252
s_addr
   pj_in_addr, 196
sa_family
   pj_addr_hdr, 187
sec
   pj_parsed_time, 201
   pj_time_val, 215
Semaphores., 99
sin6_addr
   pj_sockaddr_in6, 213
sin6_family
   pj_sockaddr_in6, 213
sin6_flowinfo
   pj_sockaddr_in6, 213
sin6_port
   pj_sockaddr_in6, 213
sin6_scope_id
   pj_sockaddr_in6, 213
sin_addr
   pj_sockaddr_in, 212
sin_family
   pj_sockaddr_in, 212
sin_port
   pj_sockaddr_in, 212
sin_zero
   pj_sockaddr_in, 212
size
   pj_file_stat, 192
   pj_rbtree, 209
slen
   pj_str_t, 214
sock.h, 250
   s6_addr, 252
   s6_addr16, 252
   s6_addr32, 252
sock_select.h, 253
Socket Abstraction, 130
Socket select() API., 150
Stack/Buffer Based Memory Pool Allocator, 123
state
   pj_exception_state_t, 190
String Operations, 153
string.h, 254
strnicmp_alnum
   PJ_PSTR, 155
Unicode Support, 180
unicode.h, 260
used_count
   pj_caching_pool, 188
used_list
   pj_caching_pool, 189
used_size
   pj_caching_pool, 188
user_data
   PJ_IOQUEUE, 66
   pj_rbtree_node, 210
   pj_timer_entry, 216
wday
   pj_parsed_time, 201
year
   pj_parsed_time, 201
types.h, 257
   PJ_TERM_COLOR_B, 259
   PJ_TERM_COLOR_BRIGHT, 259
   PJ_TERM_COLOR_G, 259
   PJ_TERM_COLOR_R, 259
u32
   pj_timestamp, 217
u6_addr16
   pj_in6_addr, 195
u6_addr32
   pj_in6_addr, 195
u6_addr8
   pj_in6_addr, 195
Unicode Support, 180
unicode.h, 260
used_count
   pj_caching_pool, 188
used_list
   pj_caching_pool, 189
used_size
   pj_caching_pool, 188
user_data
   PJ_IOQUEUE, 66
   pj_rbtree_node, 210
   pj_timer_entry, 216
wday
   pj_parsed_time, 201
year
   pj_parsed_time, 201

Generated on Thu Aug 23 10:20:33 2007 for PJLIB Reference by Doxygen