

# Process Management Interface for Exascale (PMIx) Standard

**Version 4.0 (Draft)**

*Created on September 16, 2020*

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 4.0 (Draft).

**Comments:** Please provide comments on the PMIx Standard by filing issues on the document repository <https://github.com/pmix/pmix-standard/issues> or by sending them to the PMIx Community mailing list at <https://groups.google.com/forum/#!forum/pmix>. Comments should include the version of the PMIx standard you are commenting about, and the page, section, and line numbers that you are referencing. Please note that messages sent to the mailing list from an unsubscribed e-mail address will be ignored.

Copyright © 2018-2020 PMIx Administrative Steering Committee (ASC).

Permission to copy without fee all or part of this material is granted, provided the PMIx ASC copyright notice and the title of this document appear, and notice is given that copying is by permission of PMIx ASC.

*This page intentionally left blank*

# Contents

---

<b>1. Introduction</b>	<b>1</b>
1.1. Background . . . . .	1
1.2. PMIx Architecture Overview . . . . .	1
1.3. Portability of Functionality . . . . .	3
1.3.1. Attributes in PMIx . . . . .	3
<b>2. PMIx Terms and Conventions</b>	<b>6</b>
2.1. Notational Conventions . . . . .	8
2.2. Semantics . . . . .	10
2.3. Naming Conventions . . . . .	10
2.4. Procedure Conventions . . . . .	10
<b>3. Data Structures and Types</b>	<b>12</b>
3.1. Constants . . . . .	13
3.1.1. PMIx Return Status Constants . . . . .	14
3.1.1.1. User-Defined Error and Event Constants . . . . .	15
3.2. Data Types . . . . .	15
3.2.1. Key Structure . . . . .	16
3.2.1.1. Key support macros . . . . .	16
3.2.2. Namespace Structure . . . . .	17
3.2.2.1. Namespace support macros . . . . .	17
3.2.3. Rank Structure . . . . .	18
3.2.4. Process Structure . . . . .	19
3.2.4.1. Process structure support macros . . . . .	19
3.2.5. Process State Structure . . . . .	22
3.2.6. Process Information Structure . . . . .	23
3.2.6.1. Process information structure support macros . . . . .	24
3.2.7. Job State Structure . . . . .	26

3.2.8.	Value Structure . . . . .	26
3.2.8.1.	Value structure support macros . . . . .	27
3.2.9.	Info Structure . . . . .	31
3.2.9.1.	Info structure support macros . . . . .	31
3.2.10.	Info Type Directives . . . . .	34
3.2.10.1.	Info Directive support macros . . . . .	35
3.2.11.	Environmental Variable Structure . . . . .	37
3.2.11.1.	Environmental variable support macros . . . . .	37
3.2.12.	Byte Object Type . . . . .	39
3.2.12.1.	Byte object support macros . . . . .	39
3.2.13.	Data Array Structure . . . . .	40
3.2.13.1.	Data array support macros . . . . .	40
3.2.14.	Argument Array Macros . . . . .	42
3.2.15.	Set Environment Variable . . . . .	46
3.3.	Generalized Data Types Used for Packing/Unpacking . . . . .	46
3.4.	General Callback Functions . . . . .	48
3.4.1.	Release Callback Function . . . . .	48
3.4.2.	Op Callback Function . . . . .	49
3.4.3.	Value Callback Function . . . . .	49
3.4.4.	Info Callback Function . . . . .	50
3.4.5.	Handler registration callback function . . . . .	50
3.5.	PMIx Datatype Value String Representations . . . . .	51
<b>4.</b>	<b>Client Initialization and Finalization</b>	<b>55</b>
4.1.	<b>PMIx_Initialized</b> . . . . .	55
4.2.	<b>PMIx_Get_version</b> . . . . .	56
4.3.	<b>PMIx_Init</b> . . . . .	56
4.3.1.	Initialization events . . . . .	58
4.3.2.	Initialization attributes . . . . .	58
4.3.2.1.	Connection attributes . . . . .	59
4.3.2.2.	Programming model attributes . . . . .	59
4.4.	<b>PMIx_Finalize</b> . . . . .	60
4.4.1.	Finalize attributes . . . . .	61

<b>5. Synchronization and Data Access Operations</b>	<b>62</b>
5.1. <b>PMIx_Fence</b> . . . . .	62
5.2. <b>PMIx_Fence_nb</b> . . . . .	64
5.2.1. Fence-related attributes . . . . .	66
5.3. <b>PMIx_Get</b> . . . . .	67
5.3.1. <b>PMIx_Get_nb</b> . . . . .	69
5.3.2. Retrieval-specific constants . . . . .	71
5.3.3. Retrieval attributes . . . . .	72
5.4. Query . . . . .	72
5.4.1. <b>PMIx_Resolve_peers</b> . . . . .	73
5.4.2. <b>PMIx_Resolve_nodes</b> . . . . .	73
5.4.3. <b>PMIx_Query_info</b> . . . . .	74
5.4.4. <b>PMIx_Query_info_nb</b> . . . . .	78
5.4.5. Query-specific constants . . . . .	83
5.4.6. Query attributes . . . . .	83
5.4.7. Query Structure . . . . .	86
5.4.7.1. Query structure support macros . . . . .	86
5.5. Using Get vs Query . . . . .	88
5.6. Accessing attribute support information . . . . .	88
<b>6. Reserved Keys</b>	<b>91</b>
6.1. Data realms . . . . .	91
6.1.1. Session realm attributes . . . . .	92
6.1.2. Job realm attributes . . . . .	94
6.1.3. Application realm attributes . . . . .	95
6.1.4. Process realm attributes . . . . .	97
6.1.5. Node realm keys . . . . .	98
6.2. Retrieval rules for reserved keys . . . . .	99
6.2.1. Accessing information: examples . . . . .	100
6.2.1.1. Session-level information . . . . .	100
6.2.1.2. Job-level information . . . . .	102
6.2.1.3. Application-level information . . . . .	102
6.2.1.4. Process-level information . . . . .	103
6.2.1.5. Node-level information . . . . .	103

<b>7. Process-Related Non-Reserved Keys</b>	<b>105</b>
7.1. Posting Key/Value Pairs . . . . .	106
7.1.1. <b>PMIx_Put</b> . . . . .	106
7.1.1.1. Scope of Put Data . . . . .	107
7.1.2. <b>PMIx_Store_internal</b> . . . . .	107
7.1.3. <b>PMIx_Commit</b> . . . . .	108
7.2. Retrieval rules for non-reserved keys . . . . .	109
<b>8. Publish/Lookup Operations</b>	<b>111</b>
8.1. <b>PMIx_Publish</b> . . . . .	111
8.2. <b>PMIx_Publish_nb</b> . . . . .	113
8.3. Publish-specific constants . . . . .	114
8.4. Publish-specific attributes . . . . .	114
8.5. Publish-Lookup Datatypes . . . . .	115
8.5.1. Range of Published Data . . . . .	115
8.5.2. Data Persistence Structure . . . . .	115
8.6. <b>PMIx_Lookup</b> . . . . .	116
8.7. <b>PMIx_Lookup_nb</b> . . . . .	118
8.7.1. Lookup Returned Data Structure . . . . .	120
8.7.1.1. Lookup data structure support macros . . . . .	120
8.7.2. Lookup Callback Function . . . . .	123
8.8. Retrieval rules for published data . . . . .	123
8.9. <b>PMIx_Unpublish</b> . . . . .	124
8.10. <b>PMIx_Unpublish_nb</b> . . . . .	126
<b>9. Event Notification</b>	<b>128</b>
9.1. Notification and Management . . . . .	128
9.1.1. Events versus status constants . . . . .	130
9.1.2. <b>PMIx_Register_event_handler</b> . . . . .	130
9.1.3. Event registration constants . . . . .	133
9.1.4. System events . . . . .	134
9.1.5. Event handler registration and notification attributes . . . . .	134
9.1.5.1. Fault tolerance event attributes . . . . .	135
9.1.6. Notification Function . . . . .	135

9.1.7. <b>PMIx_Deregister_event_handler</b> . . . . .	137
9.1.8. <b>PMIx_Notify_event</b> . . . . .	138
9.1.9. Notification Handler Completion Callback Function . . . . .	142
9.1.9.1. Completion Callback Function Status Codes . . . . .	142
<b>10. Data Packing and Unpacking</b>	<b>143</b>
10.1. Data Buffer Type . . . . .	143
10.2. Support Macros . . . . .	144
10.3. General Routines . . . . .	145
10.3.1. <b>PMIx_Data_pack</b> . . . . .	145
10.3.2. <b>PMIx_Data_unpack</b> . . . . .	147
10.3.3. <b>PMIx_Data_copy</b> . . . . .	149
10.3.4. <b>PMIx_Data_print</b> . . . . .	149
10.3.5. <b>PMIx_Data_copy_payload</b> . . . . .	150
<b>11. Process Management</b>	<b>152</b>
11.1. Abort . . . . .	152
11.1.1. <b>PMIx_Abort</b> . . . . .	152
11.2. Process Creation . . . . .	153
11.2.1. <b>PMIx_Spawn</b> . . . . .	153
11.2.2. <b>PMIx_Spawn_nb</b> . . . . .	158
11.2.3. Spawn-specific constants . . . . .	163
11.2.4. Spawn attributes . . . . .	164
11.2.5. Application Structure . . . . .	167
11.2.5.1. App structure support macros . . . . .	168
11.2.5.2. Spawn Callback Function . . . . .	170
11.3. Connecting and Disconnecting Processes . . . . .	170
11.3.1. <b>PMIx_Connect</b> . . . . .	171
11.3.2. <b>PMIx_Connect_nb</b> . . . . .	173
11.3.3. <b>PMIx_Disconnect</b> . . . . .	175
11.3.4. <b>PMIx_Disconnect_nb</b> . . . . .	177
11.4. Process Locality . . . . .	179
11.4.1. <b>PMIx_Load_topology</b> . . . . .	179

11.4.2. <b>PMIx_Get_relative_locality</b> . . . . .	180
11.4.2.1. Topology description . . . . .	180
11.4.2.2. Initialize the topology description structure . . . . .	181
11.4.2.3. Relative locality of two processes . . . . .	181
11.4.2.4. Locality keys . . . . .	181
11.4.3. <b>PMIx_Get_cpuset</b> . . . . .	181
<b>12. Job Management and Reporting</b>	<b>183</b>
12.1. Allocation Requests . . . . .	183
12.1.1. <b>PMIx_Allocation_request</b> . . . . .	183
12.1.2. <b>PMIx_Allocation_request_nb</b> . . . . .	186
12.1.3. Job Allocation attributes . . . . .	189
12.1.4. Job Allocation Directives . . . . .	191
12.2. Job Control . . . . .	191
12.2.1. <b>PMIx_Job_control</b> . . . . .	192
12.2.2. <b>PMIx_Job_control_nb</b> . . . . .	194
12.2.3. Job control constants . . . . .	197
12.2.4. Job control events . . . . .	198
12.2.5. Job control attributes . . . . .	198
12.3. Process and Job Monitoring . . . . .	199
12.3.1. <b>PMIx_Process_monitor</b> . . . . .	199
12.3.2. <b>PMIx_Process_monitor_nb</b> . . . . .	202
12.3.3. <b>PMIx_Heartbeat</b> . . . . .	204
12.3.4. Monitoring events . . . . .	204
12.3.5. Monitoring attributes . . . . .	204
12.4. Logging . . . . .	205
12.4.1. <b>PMIx_Log</b> . . . . .	205
12.4.2. <b>PMIx_Log_nb</b> . . . . .	208
12.4.3. Log attributes . . . . .	211
<b>13. Process Sets and Groups</b>	<b>213</b>
13.1. Process Sets . . . . .	213
13.1.1. Process Set Constants . . . . .	214
13.1.2. Process Set Attributes . . . . .	215

13.2. Process Groups . . . . .	215
13.2.1. Relation to the host environment . . . . .	215
13.2.2. Construction procedure . . . . .	216
13.2.3. Destruct procedure . . . . .	217
13.2.4. Process Group Events . . . . .	217
13.2.5. Process Group Attributes . . . . .	218
13.2.6. <b>PMIx_Group_construct</b> . . . . .	220
13.2.7. <b>PMIx_Group_construct_nb</b> . . . . .	223
13.2.8. <b>PMIx_Group_destruct</b> . . . . .	226
13.2.9. <b>PMIx_Group_destruct_nb</b> . . . . .	227
13.2.10. <b>PMIx_Group_invite</b> . . . . .	229
13.2.11. <b>PMIx_Group_invite_nb</b> . . . . .	232
13.2.12. <b>PMIx_Group_join</b> . . . . .	235
13.2.13. <b>PMIx_Group_join_nb</b> . . . . .	237
13.2.13.1. Group accept/decline directives . . . . .	238
13.2.14. <b>PMIx_Group_leave</b> . . . . .	238
13.2.15. <b>PMIx_Group_leave_nb</b> . . . . .	240
<b>14. Fabric Support Definitions</b>	<b>242</b>
14.1. Fabric Support Events . . . . .	245
14.2. Fabric Support Datatypes . . . . .	245
14.2.1. Fabric Endpoint Structure . . . . .	245
14.2.2. Fabric endpoint support macros . . . . .	246
14.2.3. Fabric Device Distance Structure . . . . .	247
14.2.4. Fabric device distance support macros . . . . .	247
14.2.5. Fabric Coordinate Structure . . . . .	249
14.2.6. Fabric coordinate support macros . . . . .	249
14.2.7. Fabric Geometry Structure . . . . .	250
14.2.8. Fabric geometry support macros . . . . .	251
14.2.9. Fabric Coordinate Views . . . . .	252
14.2.10. Fabric Link State . . . . .	253
14.2.11. Fabric Operation Constants . . . . .	253
14.2.12. Fabric registration structure . . . . .	253
14.2.12.1. Initialize the fabric structure . . . . .	256

14.3. Fabric Support Attributes . . . . .	257
14.4. Fabric Support Functions . . . . .	260
14.4.1. <b>PMIx_Fabric_register</b> . . . . .	261
14.4.2. <b>PMIx_Fabric_register_nb</b> . . . . .	262
14.4.3. <b>PMIx_Fabric_update</b> . . . . .	263
14.4.4. <b>PMIx_Fabric_update_nb</b> . . . . .	264
14.4.5. <b>PMIx_Fabric_deregister</b> . . . . .	264
14.4.6. <b>PMIx_Fabric_deregister_nb</b> . . . . .	265
14.4.7. <b>PMIx_Fabric_update_distances</b> . . . . .	265
14.4.8. <b>PMIx_Fabric_update_distances_nb</b> . . . . .	266
<b>15. Security</b>	<b>268</b>
15.1. Obtaining Credentials . . . . .	268
15.1.1. <b>PMIx_Get_credential</b> . . . . .	269
15.1.2. <b>PMIx_Get_credential_nb</b> . . . . .	270
15.1.3. Credential Attributes . . . . .	271
15.2. Validating Credentials . . . . .	272
15.2.1. <b>PMIx_Validate_credential</b> . . . . .	272
15.2.2. <b>PMIx_Validate_credential_nb</b> . . . . .	273
<b>16. Server-Specific Interfaces</b>	<b>276</b>
16.1. Server Initialization and Finalization . . . . .	276
16.1.1. <b>PMIx_server_init</b> . . . . .	276
16.1.2. <b>PMIx_server_finalize</b> . . . . .	279
16.1.3. Server Initialization Attributes . . . . .	280
16.2. Server Support Functions . . . . .	281
16.2.1. <b>PMIx_generate_regex</b> . . . . .	281
16.2.2. <b>PMIx_generate_ppn</b> . . . . .	282
16.2.3. <b>PMIx_server_register_nspace</b> . . . . .	283
16.2.3.1. Namespace registration attributes . . . . .	292
16.2.3.2. Assembling the registration information . . . . .	294
16.2.4. <b>PMIx_server_deregister_nspace</b> . . . . .	302
16.2.5. <b>PMIx_server_register_resources</b> . . . . .	303
16.2.6. <b>PMIx_server_deregister_resources</b> . . . . .	304

16.2.7. <b>PMIx_server_register_client</b>	305
16.2.8. <b>PMIx_server_deregister_client</b>	306
16.2.9. <b>PMIx_server_setup_fork</b>	307
16.2.10. <b>PMIx_server_dmodex_request</b>	308
16.2.10.1. Server Direct Modex Response Callback Function	309
16.2.11. <b>PMIx_server_setup_application</b>	310
16.2.11.1. Server Setup Application Callback Function	313
16.2.11.2. Server Setup Application Attributes	314
16.2.12. <b>PMIx_Register_attributes</b>	314
16.2.12.1. Attribute registration constants	316
16.2.12.2. Attribute registration structure	316
16.2.12.3. Attribute registration structure descriptive attributes	317
16.2.12.4. Attribute registration structure support macros	317
16.2.13. <b>PMIx_server_setup_local_support</b>	319
16.2.14. <b>PMIx_server_IOF_deliver</b>	321
16.2.15. <b>PMIx_server_collect_inventory</b>	322
16.2.16. <b>PMIx_server_deliver_inventory</b>	323
16.2.17. <b>PMIx_server_generate_locality_string</b>	324
16.2.18. <b>PMIx_server_generate_cpuset_string</b>	324
16.2.18.1. Cpuset Structure	325
16.2.18.2. Cpuset support macros	325
16.2.19. <b>PMIx_server_define_process_set</b>	326
16.2.20. <b>PMIx_server_delete_process_set</b>	327
16.3. Server Function Pointers	328
16.3.1. <b>pmix_server_module_t</b> Module	328
16.3.2. <b>pmix_server_client_connected_fn_t</b>	330
16.3.3. <b>pmix_server_client_connected2_fn_t</b>	330
16.3.4. <b>pmix_server_client_finalized_fn_t</b>	332
16.3.5. <b>pmix_server_abort_fn_t</b>	333
16.3.6. <b>pmix_server_fencenb_fn_t</b>	334
16.3.6.1. Modex Callback Function	338
16.3.7. <b>pmix_server_dmodex_req_fn_t</b>	338
16.3.7.1. Dmodex attributes	340

16.3.8. <b><code>pmix_server_publish_fn_t</code></b> . . . . .	340
16.3.9. <b><code>pmix_server_lookup_fn_t</code></b> . . . . .	342
16.3.10. <b><code>pmix_server_unpublish_fn_t</code></b> . . . . .	344
16.3.11. <b><code>pmix_server_spawn_fn_t</code></b> . . . . .	346
16.3.11.1. Server spawn attributes . . . . .	352
16.3.12. <b><code>pmix_server_connect_fn_t</code></b> . . . . .	352
16.3.13. <b><code>pmix_server_disconnect_fn_t</code></b> . . . . .	354
16.3.14. <b><code>pmix_server_register_events_fn_t</code></b> . . . . .	356
16.3.15. <b><code>pmix_server_deregister_events_fn_t</code></b> . . . . .	358
16.3.16. <b><code>pmix_server_notify_event_fn_t</code></b> . . . . .	360
16.3.17. <b><code>pmix_server_listener_fn_t</code></b> . . . . .	361
16.3.17.1. PMIx Client Connection Callback Function . . . . .	362
16.3.18. <b><code>pmix_server_query_fn_t</code></b> . . . . .	363
16.3.19. <b><code>pmix_server_tool_connection_fn_t</code></b> . . . . .	365
16.3.19.1. Tool connection attributes . . . . .	368
16.3.19.2. PMIx Tool Connection Callback Function . . . . .	368
16.3.20. <b><code>pmix_server_log_fn_t</code></b> . . . . .	368
16.3.21. <b><code>pmix_server_alloc_fn_t</code></b> . . . . .	370
16.3.22. <b><code>pmix_server_job_control_fn_t</code></b> . . . . .	373
16.3.23. <b><code>pmix_server_monitor_fn_t</code></b> . . . . .	376
16.3.24. <b><code>pmix_server_get_cred_fn_t</code></b> . . . . .	379
16.3.24.1. Credential callback function . . . . .	380
16.3.25. <b><code>pmix_server_validate_cred_fn_t</code></b> . . . . .	381
16.3.26. Credential validation callback function . . . . .	383
16.3.27. <b><code>pmix_server_iof_fn_t</code></b> . . . . .	384
16.3.27.1. IOF delivery function . . . . .	387
16.3.28. <b><code>pmix_server_stdin_fn_t</code></b> . . . . .	388
16.3.29. <b><code>pmix_server_grp_fn_t</code></b> . . . . .	389
16.3.29.1. Group Operation Constants . . . . .	392
16.3.30. <b><code>pmix_server_fabric_fn_t</code></b> . . . . .	392
<b>17. Tools and Debuggers</b>	<b>394</b>
17.1. Connection Mechanisms . . . . .	394
17.1.1. Rendezvousing with a local server . . . . .	397

17.1.2.	Connecting to a remote server . . . . .	398
17.1.3.	Attaching to running jobs . . . . .	398
17.1.4.	Tool initialization attributes . . . . .	399
17.1.5.	Tool initialization environmental variables . . . . .	399
17.1.6.	Tool connection attributes . . . . .	400
17.2.	Launching Applications with Tools . . . . .	401
17.2.1.	Direct launch . . . . .	401
17.2.2.	Indirect launch . . . . .	405
17.2.3.	Tool spawn-related attributes . . . . .	408
17.2.4.	Tool rendezvous-related events . . . . .	409
17.3.	IO Forwarding . . . . .	409
17.3.1.	Forwarding stdout/stderr . . . . .	410
17.3.2.	Forwarding stdin . . . . .	412
17.3.3.	IO Forwarding Channels . . . . .	413
17.3.4.	IO Forwarding constants . . . . .	413
17.3.5.	IO Forwarding attributes . . . . .	414
17.4.	Debugger Support . . . . .	415
17.4.1.	Co-Location of Debugger Daemons . . . . .	417
17.4.2.	Co-Spawn of Debugger Daemons . . . . .	419
17.4.3.	Debugger Agents . . . . .	420
17.4.4.	Tracking the job lifecycle . . . . .	421
17.4.4.1.	Job lifecycle events . . . . .	422
17.4.4.2.	Job lifecycle attributes . . . . .	423
17.4.5.	Debugger-related constants . . . . .	423
17.4.6.	Debugger attributes . . . . .	423
17.5.	Tool-Specific APIs . . . . .	425
17.5.1.	<b>PMIx_tool_init</b> . . . . .	425
17.5.2.	<b>PMIx_tool_finalize</b> . . . . .	428
17.5.3.	<b>PMIx_tool_disconnect</b> . . . . .	429
17.5.4.	<b>PMIx_tool_attach_to_server</b> . . . . .	429
17.5.5.	<b>PMIx_tool_get_servers</b> . . . . .	431
17.5.6.	<b>PMIx_tool_set_server</b> . . . . .	432
17.5.7.	<b>PMIx_IOF_pull</b> . . . . .	432

17.5.8. <b>PMIx_IOF_deregister</b> . . . . .	434
17.5.9. <b>PMIx_IOF_push</b> . . . . .	435
<b>A. Python Bindings</b>	<b>439</b>
A.1. Design Considerations . . . . .	439
A.1.1. Error Codes vs Python Exceptions . . . . .	439
A.1.2. Representation of Structured Data . . . . .	439
A.2. Datatype Definitions . . . . .	440
A.2.1. Example . . . . .	446
A.3. Callback Function Definitions . . . . .	447
A.3.1. IOF Delivery Function . . . . .	447
A.3.2. Event Handler . . . . .	447
A.3.3. Server Module Functions . . . . .	448
A.3.3.1. Client Connected . . . . .	448
A.3.3.2. Client Finalized . . . . .	449
A.3.3.3. Client Aborted . . . . .	449
A.3.3.4. Fence . . . . .	450
A.3.3.5. Direct Modex . . . . .	451
A.3.3.6. Publish . . . . .	451
A.3.3.7. Lookup . . . . .	452
A.3.3.8. Unpublish . . . . .	452
A.3.3.9. Spawn . . . . .	453
A.3.3.10. Connect . . . . .	453
A.3.3.11. Disconnect . . . . .	454
A.3.3.12. Register Events . . . . .	454
A.3.3.13. Deregister Events . . . . .	455
A.3.3.14. Notify Event . . . . .	455
A.3.3.15. Query . . . . .	455
A.3.3.16. Tool Connected . . . . .	456
A.3.3.17. Log . . . . .	456
A.3.3.18. Allocate Resources . . . . .	457
A.3.3.19. Job Control . . . . .	457
A.3.3.20. Monitor . . . . .	458
A.3.3.21. Get Credential . . . . .	458

A.3.3.22. Validate Credential . . . . .	459
A.3.3.23. IO Forward . . . . .	460
A.3.3.24. IO Push . . . . .	460
A.3.3.25. Group Operations . . . . .	461
A.3.3.26. Fabric Operations . . . . .	461
<b>A.4. PMIxClient . . . . .</b>	<b>462</b>
A.4.1. Client.init . . . . .	462
A.4.2. Client.initialized . . . . .	462
A.4.3. Client.get_version . . . . .	463
A.4.4. Client.finalize . . . . .	463
A.4.5. Client.abort . . . . .	463
A.4.6. Client.store_internal . . . . .	464
A.4.7. Client.put . . . . .	464
A.4.8. Client.commit . . . . .	465
A.4.9. Client.fence . . . . .	465
A.4.10. Client.get . . . . .	466
A.4.11. Client.publish . . . . .	466
A.4.12. Client.lookup . . . . .	467
A.4.13. Client.unpublish . . . . .	467
A.4.14. Client.spawn . . . . .	468
A.4.15. Client.connect . . . . .	468
A.4.16. Client.disconnect . . . . .	469
A.4.17. Client.resolve_peers . . . . .	469
A.4.18. Client.resolve_nodes . . . . .	470
A.4.19. Client.query . . . . .	470
A.4.20. Client.log . . . . .	471
A.4.21. Client.allocate . . . . .	471
A.4.22. Client.job_ctrl . . . . .	472
A.4.23. Client.monitor . . . . .	472
A.4.24. Client.get_credential . . . . .	473
A.4.25. Client.validate_credential . . . . .	473
A.4.26. Client.group_construct . . . . .	474
A.4.27. Client.group_invite . . . . .	475

A.4.28.	Client.group_join . . . . .	475
A.4.29.	Client.group_leave . . . . .	476
A.4.30.	Client.group_destruct . . . . .	476
A.4.31.	Client.register_event_handler . . . . .	477
A.4.32.	Client.deregister_event_handler . . . . .	477
A.4.33.	Client.notify_event . . . . .	478
A.4.34.	Client.fabric_register . . . . .	478
A.4.35.	Client.fabric_update . . . . .	479
A.4.36.	Client.fabric_deregister . . . . .	479
A.4.37.	Client.load_topology . . . . .	479
A.4.38.	Client.get_relative_locality . . . . .	480
A.4.39.	Client.error_string . . . . .	480
A.4.40.	Client.proc_state_string . . . . .	481
A.4.41.	Client.scope_string . . . . .	481
A.4.42.	Client.persistence_string . . . . .	482
A.4.43.	Client.data_range_string . . . . .	482
A.4.44.	Client.info_directives_string . . . . .	482
A.4.45.	Client.data_type_string . . . . .	483
A.4.46.	Client.alloc_directive_string . . . . .	483
A.4.47.	Client.iof_channel_string . . . . .	484
A.4.48.	Client.job_state_string . . . . .	484
A.4.49.	Client.get_attribute_string . . . . .	484
A.4.50.	Client.get_attribute_name . . . . .	485
A.4.51.	Client.link_state_string . . . . .	485
A.5.	PMIxServer . . . . .	486
A.5.1.	Server.init . . . . .	486
A.5.2.	Server.finalize . . . . .	486
A.5.3.	Server.generate_regex . . . . .	487
A.5.4.	Server.generate_ppn . . . . .	487
A.5.5.	Server.register_nspace . . . . .	488
A.5.6.	Server.deregister_nspace . . . . .	488
A.5.7.	Server.register_resources . . . . .	488
A.5.8.	Server.deregister_resources . . . . .	489

A.5.9.	Server.register_client . . . . .	489
A.5.10.	Server.deregister_client . . . . .	490
A.5.11.	Server.setup_fork . . . . .	490
A.5.12.	Server.dmodex_request . . . . .	491
A.5.13.	Server.setup_application . . . . .	491
A.5.14.	Server.register_attributes . . . . .	492
A.5.15.	Server.setup_local_support . . . . .	492
A.5.16.	Server.iof_deliver . . . . .	493
A.5.17.	Server.collect_inventory . . . . .	493
A.5.18.	Server.deliver_inventory . . . . .	494
A.5.19.	Server.generate_locality_string . . . . .	494
A.5.20.	Server.define_process_set . . . . .	495
A.5.21.	Server.delete_process_set . . . . .	495
A.6.	PMIxTool . . . . .	496
A.6.1.	Tool.init . . . . .	496
A.6.2.	Tool.finalize . . . . .	496
A.6.3.	Tool.disconnect . . . . .	497
A.6.4.	Tool.attach_to_server . . . . .	497
A.6.5.	Tool.get_servers . . . . .	498
A.6.6.	Tool.iof_pull . . . . .	498
A.6.7.	Tool.iof_deregister . . . . .	499
A.6.8.	Tool.iof_push . . . . .	500
A.7.	Example Usage . . . . .	500
A.7.1.	Python Client . . . . .	500
A.7.2.	Python Server . . . . .	503

<b>B. Revision History</b>	<b>507</b>	
B.1.	Version 1.0: June 12, 2015 . . . . .	507
B.2.	Version 2.0: Sept. 2018 . . . . .	508
B.2.1.	Removed/Modified Application Programming Interfaces (APIs) . . . . .	508
B.2.2.	Deprecated constants . . . . .	508
B.2.3.	Deprecated attributes . . . . .	509
B.3.	Version 2.1: Dec. 2018 . . . . .	509
B.4.	Version 2.2: Jan 2019 . . . . .	510

B.5.	Version 3.0: Dec. 2018 . . . . .	510
B.5.1.	Removed constants . . . . .	511
B.5.2.	Deprecated attributes . . . . .	511
B.5.3.	Removed attributes . . . . .	511
B.6.	Version 3.1: Jan. 2019 . . . . .	512
B.7.	Version 3.2: Sept. 2020 . . . . .	512
B.8.	Version 4.0: Sept 2020 . . . . .	513
B.8.1.	Added Constants . . . . .	515
B.8.2.	Added Attributes . . . . .	517
B.8.3.	Added Environmental Variables . . . . .	529
B.8.4.	Deprecated APIs . . . . .	530
B.8.5.	Deprecated constants . . . . .	530
B.8.6.	Removed constants . . . . .	530
B.8.7.	Deprecated attributes . . . . .	531
B.8.8.	Removed attributes . . . . .	532
<b>C. Acknowledgements</b>		<b>534</b>
C.1.	Version 4.0 . . . . .	534
C.2.	Version 3.0 . . . . .	534
C.3.	Version 2.0 . . . . .	535
C.4.	Version 1.0 . . . . .	536
<b>Bibliography</b>		<b>538</b>
<b>Index</b>		<b>539</b>
<b>Index of APIs</b>		<b>541</b>
<b>Index of Support Macros</b>		<b>547</b>
<b>Index of Data Structures</b>		<b>550</b>
<b>Index of Constants</b>		<b>552</b>
<b>Index of Environmental Variables</b>		<b>559</b>



## CHAPTER 1

# Introduction

---

1 Process Management Interface - Exascale (PMIx) is an application programming interface standard  
2 that provides libraries and programming models with portable and well-defined access to commonly  
3 needed services in distributed and parallel computing systems. A typical example of such a service  
4 is the portable and scalable exchange of network addresses to establish communication channels  
5 between the processes of a parallel application or service. As such, PMIx gives distributed system  
6 software providers a better understanding of how programming models and libraries can interface  
7 with and use system-level services. As a standard, PMIx provides APIs that allow for portable  
8 access to these varied system software services and the functionalities they offer. Although these  
9 services can be defined and implemented directly by the system software components providing  
10 them, the community represented by the ASC feels that the development of a shared standard better  
11 serves the community. As a result, PMIx enables programming languages and libraries to focus on  
12 their core competencies without having to provide their own system-level services.

## 1.1 Background

14 The Process Management Interface (PMI) has been used for quite some time as a means of  
15 exchanging wireup information needed for inter-process communication. Two versions (PMI-1 and  
16 PMI-2 [2]) have been released as part of the MPICH effort, with PMI-2 demonstrating better  
17 scaling properties than its PMI-1 predecessor.

18 PMI-1 and PMI-2 can be implemented using PMIx though PMIx is not a strict superset of either.  
19 Since its introduction, PMIx has expanded on earlier PMI efforts by providing an extended version  
20 of the PMI APIs which provide necessary functionality for launching and managing parallel  
21 applications and tools at scale.

22 The increase in adoption has motivated the creation of this document to formally specify the  
23 intended behavior of the PMIx APIs.

24 More information about the PMIx standard and affiliated projects can be found at the PMIx web  
25 site: <https://pmix.org>

## 1.2 PMIx Architecture Overview

27 The presentation of the PMIx APIs within this document makes some basic assumptions about how  
28 these APIs are used and implemented. These assumptions are generally made only to simplify the  
29 presentation and explain PMIx with the expectation that most readers have similar concepts on how

1 computing systems are organized today. However, ultimately this document should only be  
2 assumed to define a set of APIs.

3 A concept that is fundamental to PMIx is that a PMIx implementation might operate primarily as a  
4 *messenger*, and not a *doer* — i.e., a PMIx implementation might rely heavily or fully on other  
5 software components to provide functionality [1]. Since a PMIx implementation might only deliver  
6 requests and responses to other software components, the API calls include ways to provide  
7 arbitrary information to the backend components that actually implement the functionality. Also,  
8 because PMIx implementations generally rely heavily on other system software, a PMIx  
9 implementation might not be able to guarantee that a feature is available on all platforms the  
10 implementation supports. These aspects are discussed in detail in the remainder of this chapter.

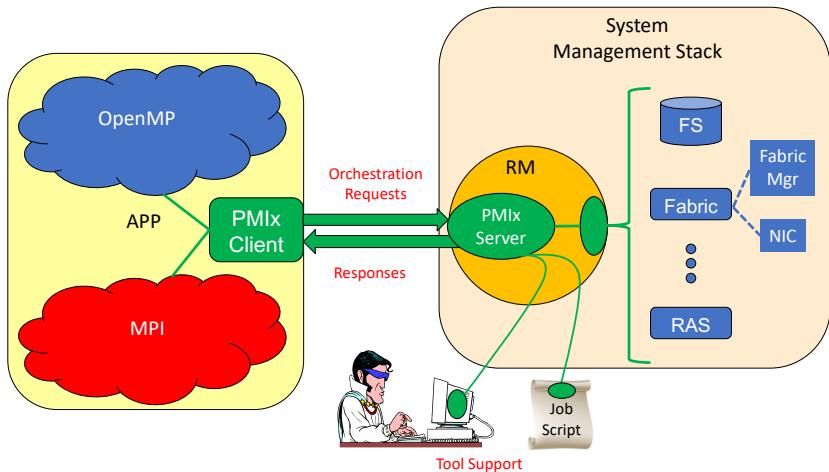


Figure 1.1.: PMIx-SMS Interactions

11 Fig. 1.1 shows a typical PMIx implementation in which the application is built against a PMIx  
12 client library that contains the client-side APIs, attribute definitions, and communication support  
13 for interacting with the local PMIx server. PMIx clients are processes which are started through the  
14 PMIx infrastructure, either by the PMIx implementation directly or through a System Management  
15 Software stack (SMS) component, and have registered as clients. A PMIx client is created in such a  
16 way that the PMIx client library will have sufficient information available to authenticate with  
17 the PMIx server. The PMIx server will have sufficient knowledge about the process which it  
18 created, either directly or through other SMS, to authenticate the process and provide information  
19 to the process requests such as its identity and the identity of its peers.

20 As clients invoke PMIx APIs, it is possible that some client requests can be handled at the client  
21 level. Other requests might require communication with the local PMIx server, which subsequently  
22 might request services from the host SMS (represented here by a Resource Manager (RM)  
23 daemon). The interaction between the PMIx server and SMS are achieved using callback functions  
24 registered during server initialization. The host SMS can indicate its lack of support for any

1 operation by simply providing a *NULL* for the associated callback function, or can create a function  
2 entry that returns *not supported* when called.

3 Recognizing the burden this places on SMS vendors, the PMIx community has included interfaces  
4 by which the host SMS (containing the local PMIx service instance) can request support from local  
5 SMS elements via the PMIx API. Once the SMS has transferred the request to an appropriate  
6 location, a PMIx server interface can be used to pass the request between SMS subsystems. For  
7 example, a request for network traffic statistics can utilize the PMIx networking abstractions to  
8 retrieve the information from the Fabric Manager. This reduces the portability and interoperability  
9 issues between the individual subsystems by transferring the burden of defining the interoperable  
10 interfaces from the SMS subsystems to the PMIx community, which continues to work with those  
11 providers to develop the necessary support.

12 Fig. 1.1 shows how tools can interact with the PMIx architecture. Tools, whether standalone or  
13 embedded in job scripts, are an exception to the normal client registration process. A process can  
14 register as a tool, provided the PMIx client library has adequate rendezvous information to connect  
15 to the appropriate PMIx server (either hosted on the local machine or on a remote machine). This  
16 allows processes which were not created by the PMIx infrastructure to request access to PMIx  
17 functionality.

## 18 1.3 Portability of Functionality

19 It is difficult to define a portable API that will provide access to the many and varied features  
20 underlying the operations for which PMIx provides access. For example, the options and features  
21 provided to request the creation of new processes varied dramatically between different systems  
22 existing at the time PMIx was introduced. Many RMs provide rich interfaces to specify the  
23 resources assigned to processes. As a result, PMIx is faced with the challenge of attempting to meet  
24 the seemingly conflicting goals of creating an API which allows access to these diverse features  
25 while being portable across a wide range of existing software environments. In addition, the  
26 functionalities required by different clients vary greatly. Producing a PMIx implementation which  
27 can provide the needs of all possible clients on all of its target systems could be so burdensome as  
28 to discourage PMIx implementations.

29 To help address this issue, the PMIx APIs are designed to allow resource managers and other  
30 system management stack components to decide on support of a particular function and allow client  
31 applications to query and adjust to the level of support available. PMIx clients should be written to  
32 account for the possibility that a PMIx API might return an error code indicating that the call is not  
33 supported. The PMIx community continues to look at ways to assist SMS implementers in their  
34 decisions on what functionality to support by highlighting functions and attributes that are critical  
35 to basic application execution (e.g., [PMIx\\_Get](#)) for certain classes of applications.

### 36 1.3.1 Attributes in PMIx

37 An area where differences between support on different systems can be challenging is regarding the  
38 attributes that provide information to the client process and/or control the behavior of a PMIx API.

1 Most PMIx API calls can accept additional information or attributes specified in the form of  
2 key/value pairs. These attributes provide information to the PMIx implementation that influence the  
3 behavior of the API call. In addition to API calls being optional, support for the individual  
4 attributes of an API call can vary between systems or implementations.

5 An application can adapt to the attribute support on a particular system in one of two ways. PMIx  
6 provides an API to enable an application to query the attributes supported by a particular API (See  
7 [5.6](#)). Through this API, the PMIx implementation can provide detailed information about the  
8 attributes supported on a system for each API call queried. Alternatively, the application can mark  
9 attributes as required using a flag within the `pmix_info_t` (See [3.2.9](#)). If the required attribute is  
10 not available on the system or the desired value for the attribute is not available, the call will return  
11 the error code for *not supported*.

12 For example, the `PMIX_TIMEOUT` attribute can be used to specify the time (in seconds) before the  
13 requested operation should time out. The intent of this attribute is to allow the client to avoid  
14 “hanging” in a request that takes longer than the client wishes to wait, or may never return (e.g., a  
15 `PMIx_Fence` that a blocked participant never enters).

16 The application can query the attribute support for `PMIx_Fence` and search whether  
17 `PMIX_TIMEOUT` is listed as a supported attribute. The application can also set the required flag in  
18 the `pmix_info_t` for that attribute when making the `PMIx_Fence` call. This will return an  
19 error if this attribute is not supported. If the required flag is not set, the library and SMS host are  
20 allowed to treat the attribute as optional, ignoring it if support is not available.

21 It is therefore critical that users and application implementers:

- 22 a) consider whether or not a given attribute is required, marking it accordingly; and
- 23 b) check the return status on all PMIx function calls to ensure support was present and that the  
24 request was accepted. Note that for non-blocking APIs, a return of `PMIX_SUCCESS` only  
25 indicates that the request had no obvious errors and is being processed – the eventual callback  
26 will return the status of the requested operation itself.

27 PMIx clients (e.g., tools, parallel programming libraries) may find that they depend only on a small  
28 subset of interfaces and attributes to work correctly. PMIx clients are strongly advised to define a  
29 document itemizing the PMIx interfaces and associated attributes that are required for correct  
30 operation, and are optional but recommended for full functionality. The PMIx standard cannot  
31 define this list for all given PMIx clients, but such a list is valuable to RMs desiring to support these  
32 clients.

33 A PMIx implementation may be able to support only a subset of the PMIx API and attributes on a  
34 particular system due to either its own limitations or limitations of the SMS with which it  
35 interfaces. A PMIx implemenation may also provide additional attributes beyond those defined  
36 herein in order to allow applications to access the full features of the underlying SMS. PMIx  
37 implementations are strongly advised to document the PMIx interfaces and associated attributes  
38 they support, with any annotations about behavior limitations. The PMIx standard cannot define  
39 this support for implementations, but such documentation is valuable to PMIx clients desiring to  
40 support a broad range of systems.

1 While a PMIx library implementer, or an SMS component server, may choose to support a  
2 particular PMIx API, they are not required to support every attribute that might apply to it. This  
3 would pose a significant barrier to entry for an implementer as there can be a broad range of  
4 applicable attributes to a given API, at least some of which may rarely be used.

5 Note that an environment that does not include support for a particular attribute/API pair is not  
6 "incomplete" or of lower quality than one that does include that support. Vendors must decide  
7 where to invest their time based on the needs of their target markets, and it is perfectly reasonable  
8 for them to perform cost/benefit decisions when considering what functions and attributes to  
9 support.

10 Attributes in this document are organized according to their primary usage, either grouped with a  
11 specific API or included in an appropriate functional chapter. Attributes in the PMIx Standard all  
12 start with "**PMIX**" in their name, and many include a functional description as part of their name  
13 (e.g., the use of "**PMIX\_FABRIC\_**" at the beginning of fabric-specific attributes). The PMIx  
14 Standard also defines an attribute that can be used to indicate that an attribute variable has not yet  
15 been set:

16 **PMIX\_ATTR\_UNDEF** "pmix.undef" (NULL)

17 A default attribute name signifying that the attribute field of a PMIx structure (e.g., a  
18 [pmix\\_info\\_t](#)) has not yet been defined.

## CHAPTER 2

# PMIx Terms and Conventions

---

In this chapter we describe some common terms and conventions used throughout this document. The PMIx Standard has adopted the widespread use of key-value *attributes* to add flexibility to the functionality expressed in the existing APIs. Accordingly, the ASC has chosen to require that the definition of each standard API include the passing of an array of attributes. These provide a means of customizing the behavior of the API as future needs emerge without having to alter or create new variants of it. In addition, attributes provide a mechanism by which researchers can easily explore new approaches to a given operation without having to modify the API itself.

In an effort to maintain long-term backward compatibility, PMIx does not include large numbers of APIs that each focus on a narrow scope of functionality, but instead relies on the definition of fewer generic APIs that include arrays of key-value attributes for “tuning” the function’s behavior. Thus, modifications to the PMIx standard primarily consist of the definition of new attributes along with a description of the APIs to which they relate and the expected behavior when used with those APIs.

The following terminology is used throughout this document:

• *session* refers to a pool of resources with a unique identifier (a.k.a., the *session ID*) assigned by the WorkLoad Manager (WLM) that has been reserved for one or more users. Historically, High Performance Computing (HPC) sessions have consisted of a static allocation of resources - e.g., a block of nodes assigned to a user in response to a specific request and managed as a unified collection. However, this is changing in response to the growing use of dynamic programming models that require on-the-fly allocation and release of system resources. Accordingly, the term *session* in this document refers to a potentially dynamic entity, perhaps comprised of resources accumulated as a result of multiple allocation requests that are managed as a single unit by the WLM.

• *job* refers to a set of one or more *applications* executed as a single invocation by the user within a session with a unique identifier (a.k.a, the *job ID*) assigned by the RM or launcher. For example, the command line “`mpiexec -n 1 app1 : -n 2 app2`” generates a single Multiple Program Multiple Data (MPMD) job containing two applications. A user may execute multiple *jobs* within a given session, either sequentially or in parallel.

• *namespace* refers to a character string value assigned by the RM or launcher (e.g., `mpieexec`) to a *job*. All *applications* executed as part of that *job* share the same *namespace*. The *namespace* assigned to each *job* must be unique within the scope of the governing RM and often is implemented as a string representation of a numerical job ID. The *namespace* and *job* terms will be used interchangeably throughout the document.

• *application* refers to a single executable (binary, script, etc.) member of a *job*.

- *process* refers to an operating system process, also commonly referred to as a *heavyweight* process. A process is often comprised of multiple *lightweight threads*, commonly known as simply *threads*.
- *client* refers to a process that was registered with the PMIx server prior to being started, and connects to that PMIx server via `PMIx_Init` using its assigned namespace and rank with the information required to connect to that server being provided to the process at time of start of execution.
- *clone* refers to a process that was directly started by a PMIx client (e.g., using *fork/exec*) and calls `PMIx_Init`, thus connecting to its local PMIx server using the same namespace and rank as its parent process.
- *slot* refers to an allocated entry for a process. WLMs frequently allocate entire nodes to a *session*, but can also be configured to define the maximum number of processes that can simultaneously be executed on each node. This often corresponds to the number of hardware Processing Units (PUs) (typically cores, but can also be defined as hardware threads) on the node. However, the correlation between hardware PUs and slot allocations strictly depends upon system configuration.
- *rank* refers to the numerical location (starting from zero) of a process within the defined scope. Thus, *job rank* is the rank of a process within its *job* and is synonymous with its unqualified *rank*, while *application rank* is the rank of that process within its *application*.
- *peer* refers to another process within the same *job*.
- *workflow* refers to an orchestrated execution plan frequently involving multiple *jobs* carried out under the control of a *workflow manager* process. An example workflow might first execute a computational job to generate the flow of liquid through a complex cavity, followed by a visualization job that takes the output of the first job as its input to produce an image output.
- *scheduler* refers to the component of the SMS responsible for scheduling resource allocations. This is also generally referred to as the *system workflow manager* - for the purposes of this document, the *WLM* acronym will be used interchangeably to refer to the scheduler.
- *resource manager* is used in a generic sense to represent the subsystem that will host the PMIx server library. This could be a vendor-supplied resource manager or a third-party agent such as a programming model's runtime library.
- *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.
- *node* refers to a single operating system instance. Note that this may encompass one or more physical objects.
- *package* refers to a single object that is either soldered or connected to a printed circuit board via a mechanical socket. Packages may contain multiple chips that include (but are not limited to) processing units, memory, and peripheral interfaces.

- *processing unit*, or *PU*, is the electronic circuitry within a computer that executes instructions. Depending upon architecture and configuration settings, it may consist of a single hardware thread or multiple hardware threads collectively organized as a *core*.
- *fabric* is used in a generic sense to refer to the networks within the system regardless of speed or protocol. Any use of the term *network* in the document should be considered interchangeable with *fabric*.
- *fabric device* (or *fabric devices*) refers to an operating system fabric interface, which may be physical or virtual. Any use of the term Network Interface Card (NIC) in the document should be considered interchangeable with *fabric device*.
- *fabric plane* refers to a collection of fabric devices in a common logical or physical configuration. Fabric planes are often implemented in HPC clusters as separate overlay or physical networks controlled by a dedicated fabric manager.
- *attribute* refers to a key-value pair comprised of a string key (represented by a `pmix_key_t` structure) and an associated value containing a PMIx data type (e.g., boolean, integer, or a more complex PMIx structure). Attributes are used both as directives when passed as qualifiers to APIs (e.g., in a `pmix_info_t` array), and to identify the contents of information (e.g., to specify that the contents of the corresponding `pmix_value_t` in a `pmix_info_t` represent the `PMIX_UNIV_SIZE`).
- *key* refers to the string component of a defined *attribute*. The PMIx Standard will often refer to passing of a *key* to an API (e.g., to the `PMIx_Query_info` or `PMIx_Get` APIs) as a means of identifying requested information. In this context, the *data type* specified in the *attribute's* definition indicates the data type the caller should expect to receive in return. Note that not all *attributes* can be used as *keys* as some have specific uses solely as API qualifiers.
- *instant on* refers to a PMIx concept defined as: "All information required for setup and communication (including the address vector of endpoints for every process) is available to each process at start of execution"

The following sections provide an overview of the conventions used throughout the PMIx Standard document.

## 2.1 Notational Conventions

Some sections of this document describe programming language specific examples or APIs. Text that applies only to programs for which the base language is C is shown as follows:

1 C specific text...

2 `int foo = 42;`

3 Some text is for information only, and is not part of the normative specification. These take several  
4 forms, described in their examples below:

5 Note: General text...

### Rationale

6 Throughout this document, the rationale for the design choices made in the interface specification is  
7 set off in this section. Some readers may wish to skip these sections, while readers interested in  
8 interface design may want to read them carefully.

### Advice to users

9 Throughout this document, material aimed at users and that illustrates usage is set off in this  
10 section. Some readers may wish to skip these sections, while readers interested in programming  
11 with the PMIx API may want to read them carefully.

### Advice to PMIx library implementers

12 Throughout this document, material that is primarily commentary to PMIx library implementers is  
13 set off in this section. Some readers may wish to skip these sections, while readers interested in  
14 PMIx implementations may want to read them carefully.

### Advice to PMIx server hosts

15 Throughout this document, material that is primarily commentary aimed at host environments (e.g.,  
16 RMIs and RunTime Environments (RTEs)) providing support for the PMIx server library is set off in  
17 this section. Some readers may wish to skip these sections, while readers interested in integrating  
18 PMIx servers into their environment may want to read them carefully.

19 Attributes added in this version of the standard are shown in **magenta** to distinguish them from  
20 those defined in prior versions, which are shown in **black**. Deprecated attributes are shown in **green**  
21 and may be removed in a future version of the standard.

## 1    2.2 Semantics

2    The following terms will be taken to mean:

- 3    • *shall, must* and *will* indicate that the specified behavior is *required* of all conforming  
4    implementations
- 5    • *should* and *may* indicate behaviors that a complete implementation would include, but are not  
6    required of all conforming implementations

## 7    2.3 Naming Conventions

8    The PMIx standard has adopted the following conventions:

- 9    • PMIx constants and attributes are prefixed with **PMIX**.
- 10   • Structures and type definitions are prefixed with **pmix**.
- 11   • Underscores are used to separate words in a function or variable name.
- 12   • Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the  
13   first letter of the word following it. For example, **PMIx\_Get\_version**.
- 14   • PMIx server and tool APIs are all lower case letters following the prefix - e.g.,  
15   **PMIx\_server\_register\_nspace**.
- 16   • The **PMIx\_** prefix is used to denote functions.
- 17   • The **pmix\_** prefix is used to denote function pointer and type definitions.

18   Users should not use the "**PMIX**", "**PMIx**", or "**pmix**" prefixes in their applications or libraries  
19   so as to avoid symbol conflicts with current and later versions of the PMIx Standard.

## 20   2.4 Procedure Conventions

21   While the current APIs are based on the C programming language, it is not the intent of the PMIx  
22   Standard to preclude the use of other languages. Accordingly, the procedure specifications in the  
23   PMIx Standard are written in a language-independent syntax with the arguments marked as IN,  
24   OUT, or INOUT. The meanings of these are:

- 25   • IN: The call may use the input value but does not update the argument from the perspective of  
26   the caller at any time during the calls execution,
- 27   • OUT: The call may update the argument but does not use its input value
- 28   • INOUT: The call may both use and update the argument.

1 Many PMIx interfaces, particularly nonblocking interfaces, use a **(void\*)** callback data object  
2 passed to the function that is then passed to the associated callback. On the client side, the callback  
3 data object is an opaque, client-provided context that the client can pass to a non-blocking call.  
4 When the nonblocking call completes, the callback data object is passed back to the client without  
5 modification by the PMIx library, thus allowing the client to associate a context with that callback.  
6 This is useful if there are many outstanding nonblocking calls.

7 A similar model is used for the server module functions (see [16.3.1](#)). In this case, the PMIx library  
8 is making an upcall into its host via the PMIx server module callback function and passing a  
9 specific callback function pointer and callback data object. The PMIx library expects the host to  
10 call the cbfunc with the necessary arguments and pass back the original callback data object upon  
11 completing the operation. This gives the server-side PMIx library the ability to associate a context  
12 with the call back (since multiple operations may be outstanding). The host has no visibility into  
13 the contents of the callback data object object, nor is permitted to alter it in any way.

## CHAPTER 3

# Data Structures and Types

This chapter defines PMIx standard data structures (along with macros for convenient use), types, and constants. These apply to all consumers of the PMIx interface. Where necessary for clarification, the description of, for example, an attribute may be copied from this chapter into a section where it is used.

A PMIx implementation may define additional attributes beyond those specified in this document.

### Advice to PMIx library implementers

Structures, types, and macros in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner.

If a PMIx implementation chooses to define additional attributes they should avoid using the "**PMIX**" prefix in their name or starting the attribute string with a "**pmix**" prefix. This helps the end user distinguish between what is defined by the PMIx standard and what is specific to that PMIx implementation, and avoids potential conflicts with attributes defined by the Standard.

### Advice to users

Use of increment/decrement operations on indices inside PMIx macros is discouraged due to unpredictable behavior. For example, the following sequence:

```
15  PMIX_INFO_LOAD(&array[n++], "mykey", &mystring, PMIX_STRING);  
16  PMIX_INFO_LOAD(&array[n++], "mykey2", &myint, PMIX_INT);
```

will load the given key-values into incorrect locations if the macro is implemented as:

```
18  define PMIX_INFO_LOAD(m, k, v, t)          \  
19    do {                                     \  
20      if (NULL != (k)) {                     \  
21        pmix_strncpy((m)->key, (k), PMIX_MAX_KEYLEN);  \  
22      }                                     \  
23      (m)->flags = 0;                      \  
24      pmix_value_load(&((m)->value), (v), (t));  \  
25    } while (0)
```

since the index is cited more than once in the macro. The PMIx standard only governs the existence and syntax of macros - it does not specify their implementation. Given the freedom of implementation, a safer call sequence might be as follows:

```
1 PMIX_INFO_LOAD(&array[n], "mykey", &mystring, PMIX_STRING);  
2 ++n;  
3 PMIX_INFO_LOAD(&array[n], "mykey2", &myint, PMIX_INT);  
4 ++n;
```

5 Users are also advised to use the macros for creating, loading, and releasing PMIx structures to  
6 avoid potential issues with release of memory. For example, pointing a `pmix_envar_t` element  
7 at a static string variable and then using `PMIX_ENVAR_DESTRUCT` to clear it would generate an  
8 error as the static string had not been allocated.

## 9 3.1 Constants

10 PMIx defines a few values that are used throughout the standard to set the size of fixed arrays or as  
11 a means of identifying values with special meaning. The community makes every attempt to  
12 minimize the number of such definitions. The constants defined in this section may be used before  
13 calling any PMIx library initialization routine. Additional constants associated with specific data  
14 structures or types are defined in the section describing that data structure or type.

15 **PMIX\_MAX\_NSLEN** Maximum namespace string length as an integer.

### Advice to PMIx library implementers

16 **PMIX\_MAX\_NSLEN** should have a minimum value of 63 characters. Namespace arrays in PMIx  
17 defined structures must reserve a space of size `PMIX_MAX_NSLEN`+1 to allow room for the **NULL**  
18 terminator

19 **PMIX\_MAX\_KEYLEN** Maximum key string length as an integer.

### Advice to PMIx library implementers

20 **PMIX\_MAX\_KEYLEN** should have a minimum value of 63 characters. Key arrays in PMIx defined  
21 structures must reserve a space of size `PMIX_MAX_KEYLEN`+1 to allow room for the **NULL**  
22 terminator

23 **PMIX\_APP\_WILDCARD** A value to indicate that the user wants the data for the given key from  
24 every application that posted that key, or that the given value applies to all applications within  
25 the given namespace.

### 3.1.1 PMIx Return Status Constants

The `pmix_status_t` structure is an `int` type for return status. The tables shown in this section define the possible values for `pmix_status_t`. PMIx errors are required to always be negative, with `0` reserved for `PMIX_SUCCESS`. Values in the list that were deprecated in later standards are denoted as such. Values added to the list in this version of the standard are shown in magenta.

#### Advice to PMIx library implementers

A PMIx implementation must define all of the constants defined in this section, even if they will never return the specific value to the caller.

#### Advice to users

Other than `PMIX_SUCCESS` (which is required to be zero), the actual value of any PMIx error constant is left to the PMIx library implementer. Thus, users are advised to always refer to constant by name, and not a specific implementation's value, for portability between implementations and compatibility across library versions.

The following values are general constants used in a variety of places.

`PMIX_SUCCESS` Success.

`PMIX_ERROR` General Error.

`PMIX_ERR_EXISTS` Requested operation would overwrite an existing value - typically returned when an operation would overwrite an existing file or directory.

`PMIX_ERR_INVALID_CRED` Invalid security credentials.

`PMIX_ERR_WOULD_BLOCK` Operation would block.

`PMIX_ERR_UNKNOWN_DATA_TYPE` The data type specified in an input to the PMIx library is not recognized by the implementation.

`PMIX_ERR_TYPE_MISMATCH` The data type found in an object does not match the expected data type as specified in the API call - e.g., a request to unpack a `PMIX_BOOL` value from a buffer that does not contain a value of that type in the current unpack location.

`PMIX_ERR_UNPACK_INADEQUATE_SPACE` Inadequate space to unpack data - the number of values in the buffer exceeds the specified number to unpack.

`PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER` Unpacking past the end of the provided buffer - the number of values in the buffer is less than the specified number to unpack, or a request was made to unpack a buffer beyond the buffer's end.

`PMIX_ERR_UNPACK_FAILURE` The unpack operation failed for an unspecified reason.

`PMIX_ERR_PACK_FAILURE` The pack operation failed for an unspecified reason.

`PMIX_ERR_NO_PERMISSIONS` The user lacks permissions to execute the specified operation.

`PMIX_ERR_TIMEOUT` Either a user-specified or system-internal timeout expired.

`PMIX_ERR_UNREACH` The specified target server or client process is not reachable - i.e., a suitable connection either has not been or can not be made.

1           **PMIX\_ERR\_BAD\_PARAM**    An incorrect parameter was passed to a PMIx API.  
2           **PMIX\_ERR\_RESOURCE\_BUSY**   Resource busy - typically seen when an attempt to establish a  
3           connection to another process (e.g., a PMIx server) cannot be made due to a communication  
4           failure.  
5           **PMIX\_ERR\_OUT\_OF\_RESOURCE**   Resource exhausted.  
6           **PMIX\_ERR\_INIT**            Error during initialization.  
7           **PMIX\_ERR\_NOMEM**          Out of memory.  
8           **PMIX\_ERR\_NOT\_FOUND**       The requested information was not found.  
9           **PMIX\_ERR\_NOT\_SUPPORTED**   The requested operation is not supported by either the PMIx  
10           implementation or the host environment.  
11           **PMIX\_ERR\_COMM\_FAILURE**    Communication failure - a message failed to be sent or  
12           received, but the connection remains intact.  
13           **PMIX\_ERR\_LOST\_CONNECTION**   Lost connection between server and client or tool.  
14           **PMIX\_ERR\_INVALID\_OPERATION**   The requested operation is supported by the  
15           implementation and host environment, but fails to meet a requirement (e.g., requesting to  
16           *disconnect* from processes without first *connecting* to them, inclusion of conflicting  
17           directives, or a request to perform an operation that conflicts with an ongoing one).  
18           **PMIX\_OPERATION\_IN\_PROGRESS**   A requested operation is already in progress - the  
19           duplicate request shall therefore be ignored.  
20           **PMIX\_OPERATION\_SUCCEEDED**   The requested operation was performed atomically - no  
21           callback function will be executed.  
22           **PMIX\_ERR\_PARTIAL\_SUCCESS**   The operation is considered successful but not all elements  
23           of the operation were concluded (e.g., some members of a group construct operation chose  
24           not to participate).

### 25     3.1.1.1 User-Defined Error and Event Constants

26     PMIx establishes a boundary for constants defined in the PMIx standard. Negative values larger  
27     (i.e., more negative) than this (and any positive values greater than zero) are guaranteed not to  
28     conflict with PMIx values.

29           **PMIX\_EXTERNAL\_ERR\_BASE**    A starting point for user-level defined error and event  
30           constants. Negative values that are more negative than the defined constant are guaranteed not  
31           to conflict with PMIx values. Definitions should always be based on the  
32           **PMIX\_EXTERNAL\_ERR\_BASE** constant and not a specific value as the value of the constant  
33           may change.

## 34     3.2 Data Types

35     This section defines various data types used by the PMIx APIs. The version of the standard in  
36     which a particular data type was introduced is shown in the margin.

## 1 3.2.1 Key Structure

2 The `pmix_key_t` structure is a statically defined character array of length  
3 `PMIX_MAX_KEYLEN`+1, thus supporting keys of maximum length `PMIX_MAX_KEYLEN` while  
4 preserving space for a mandatory `NULL` terminator.

PMIx v2.0

```
5     typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];
```

6 Characters in the key must be standard alphanumeric values supported by common utilities such as  
7 `strcmp`.

### Advice to users

8 References to keys in PMIx v1 were defined simply as an array of characters of size  
9 `PMIX_MAX_KEYLEN+1`. The `pmix_key_t` type definition was introduced in version 2 of the  
10 standard. The two definitions are code-compatible and thus do not represent a break in backward  
11 compatibility.

12 Passing a `pmix_key_t` value to the standard `sizeof` utility can result in compiler warnings of  
13 incorrect returned value. Users are advised to avoid using `sizeof(pmix_key_t)` and instead rely on  
14 the `PMIX_MAX_KEYLEN` constant.

### 15 3.2.1.1 Key support macros

16 The following macros are provided for convenience when working with PMIx keys.

#### 17 Check key macro

18 Compare the key in a `pmix_info_t` to a given value.

PMIx v3.0

```
19     PMIX_CHECK_KEY(a, b)
```

20 **IN a**  
21 Pointer to the structure whose key is to be checked (pointer to `pmix_info_t`)  
22 **IN b**  
23 String value to be compared against (`char*`)  
24 Returns `true` if the key matches the given value

1      **Load key macro**

2      Load a key into a `pmix_info_t`.

PMIx v4.0

C

3      `PMIX_LOAD_KEY(a, b)`

C

4      **IN a**

5      Pointer to the structure whose key is to be loaded (pointer to `pmix_info_t`)

6      **IN b**

7      String value to be loaded (`char*`)

8      No return value.

9      **3.2.2 Namespace Structure**

10     The `pmix_nspace_t` structure is a statically defined character array of length

11     `PMIX_MAX_NSLEN+1`, thus supporting namespaces of maximum length `PMIX_MAX_NSLEN`  
12     while preserving space for a mandatory `NULL` terminator.

PMIx v2.0

C

13     `typedef char pmix_nspace_t [PMIX_MAX_NSLEN+1];`

C

14     Characters in the namespace must be standard alphanumeric values supported by common utilities  
15     such as `strcmp`.

▼      **Advice to users**      ▼

16     References to namespace values in PMIx v1 were defined simply as an array of characters of size  
17     `PMIX_MAX_NSLEN+1`. The `pmix_nspace_t` type definition was introduced in version 2 of the  
18     standard. The two definitions are code-compatible and thus do not represent a break in backward  
19     compatibility.

20     Passing a `pmix_nspace_t` value to the standard `sizeof` utility can result in compiler warnings of  
21     incorrect returned value. Users are advised to avoid using `sizeof(pmix_nspace_t)` and instead rely  
22     on the `PMIX_MAX_NSLEN` constant.

23      **3.2.2.1 Namespace support macros**

24      The following macros are provided for convenience when working with PMIx namespace  
25      structures.

1      **Check namespace macro**

2      Compare the string in a `pmix_nspace_t` to a given value.

3      *PMIx v3.0*

4      `PMIX_CHECK_NSPACE(a, b)`

C

5      **IN a**

6      Pointer to the structure whose value is to be checked (pointer to `pmix_nspace_t`)

7      **IN b**

8      String value to be compared against (`char*`)

9      Returns `true` if the namespace matches the given value

10     **Load namespace macro**

11     Load a namespace into a `pmix_nspace_t`.

12     *PMIx v4.0*

13     `PMIX_LOAD_NSPACE(a, b)`

C

14     **IN a**

15     Pointer to the target structure (pointer to `pmix_nspace_t`)

16     **IN b**

17     String value to be loaded - if `NULL` is given, then the target structure will be initialized to zero's (`char*`)

18     No return value.

### 18 3.2.3 Rank Structure

19     The `pmix_rank_t` structure is a `uint32_t` type for rank values.

20     *PMIx v1.0*

21     `typedef uint32_t pmix_rank_t;`

C

C

22     The following constants can be used to set a variable of the type `pmix_rank_t`. All definitions  
23     were introduced in version 1 of the standard unless otherwise marked. Valid rank values start at  
zero.

24     **PMIX\_RANK\_UNDEF**    A value to request job-level data where the information itself is not  
25     associated with any specific rank, or when passing a `pmix_proc_t` identifier to an  
26     operation that only references the namespace field of that structure.

27     **PMIX\_RANK\_WILDCARD**    A value to indicate that the user wants the data for the given key  
28     from every rank that posted that key.

```
1 PMIX_RANK_LOCAL_NODE    Special rank value used to define groups of ranks. This constant  
2      defines the group of all ranks on a local node.  
3 PMIX_RANK_LOCAL_PEERS   Special rank value used to define groups of ranks. This  
4      constant defines the group of all ranks on a local node within the same namespace as the  
5      current process.  
6 PMIX_RANK_INVALID       An invalid rank value.  
7 PMIX_RANK_VALID        Define an upper boundary for valid rank values.
```

## 8 3.2.4 Process Structure

9 The `pmix_proc_t` structure is used to identify a single process in the PMIx universe. It contains  
10 a reference to the namespace and the `pmix_rank_t` within that namespace.

PMIx v1.0

```
11     typedef struct pmix_proc {  
12         pmix_nspace_t nspace;  
13         pmix_rank_t rank;  
14     } pmix_proc_t;
```

C

C

### 15 3.2.4.1 Process structure support macros

16 The following macros are provided to support the `pmix_proc_t` structure.

#### 17 Initialize the proc structure

18 Initialize the `pmix_proc_t` fields.

PMIx v1.0

```
19     PMIX_PROC_CONSTRUCT (m)
```

C

C

20 IN m

21 Pointer to the structure to be initialized (pointer to `pmix_proc_t`)

#### 22 Destruct the proc structure

23 Destruct the `pmix_proc_t` fields.

C

C

```
24     PMIX_PROC_DESTRUCT (m)
```

25 IN m

26 Pointer to the structure to be destructed (pointer to `pmix_proc_t`)

27 There is nothing to release here as the fields in `pmix_proc_t` are either a statically-declared array  
28 (the namespace) or a single value (the rank). However, the macro is provided for symmetry in the  
29 code and for future-proofing should some allocated field be included some day.

1           **Create a proc array**  
2       Allocate and initialize an array of `pmix_proc_t` structures.  
3        `PMIX_PROC_CREATE (m, n)`                                      C  
4           **INOUT m**  
5        Address where the pointer to the array of `pmix_proc_t` structures shall be stored (handle)  
6           **IN n**  
7        Number of structures to be allocated (`size_t`)  
8           **Free a proc structure**  
9       Release a `pmix_proc_t` structure.  
10       `PMIX_PROC_RELEASE (m)`                                      C  
11           **IN m**  
12       Pointer to a `pmix_proc_t` structure (handle)  
13           **Free a proc array**  
14       Release an array of `pmix_proc_t` structures.  
15       `PMIX_PROC_FREE (m, n)`                                      C  
16           **IN m**  
17       Pointer to the array of `pmix_proc_t` structures (handle)  
18           **IN n**  
19       Number of structures in the array (`size_t`)  
20           **Load a proc structure**  
21       Load values into a `pmix_proc_t`.  
PMIx v2.0

1 PMIX\_PROC\_LOAD (m, n, r) C

2 IN m C  
3 Pointer to the structure to be loaded (pointer to [pmix\\_proc\\_t](#))

4 IN n C  
5 Namespace to be loaded ([pmix\\_nspace\\_t](#))

6 IN r C  
7 Rank to be assigned ([pmix\\_rank\\_t](#))

8 No return value. Deprecated in favor of [PMIX\\_LOAD\\_PROCID](#)

9 **Compare identifiers**

10 Compare two [pmix\\_proc\\_t](#) identifiers.

11 PMIx v3.0 PMIX\_CHECK\_PROCID (a, b) C

12 IN a C  
13 Pointer to a structure whose ID is to be compared (pointer to [pmix\\_proc\\_t](#))

14 IN b C  
15 Pointer to a structure whose ID is to be compared (pointer to [pmix\\_proc\\_t](#))

16 Returns **true** if the two structures contain matching namespaces and:

17 • the ranks are the same value

18 • one of the ranks is [PMIX\\_RANK\\_WILDCARD](#)

19 **Load a procID structure**

20 Load values into a [pmix\\_proc\\_t](#).

21 PMIx v4.0 PMIX\_LOAD\_PROCID (m, n, r) C

22 IN m C  
23 Pointer to the structure to be loaded (pointer to [pmix\\_proc\\_t](#))

24 IN n C  
25 Namespace to be loaded ([pmix\\_nspace\\_t](#))

26 IN r C  
27 Rank to be assigned ([pmix\\_rank\\_t](#))

1      **Construct a multi-cluster namespace**

2      Construct a multi-cluster identifier containing a cluster ID and a namespace.

3      *PMIx v4.0*

C

3      **PMIX\_MULTICLUSTER\_NSPACE\_CONSTRUCT (m, n, r)**

C

4      IN    m

5            *pmix\_nspace\_t* structure that will contain the multi-cluster identifier (*pmix\_nspace\_t*)

6      IN    n

7            Cluster identifier (*char\**)

8      IN    n

9            Namespace to be loaded (*pmix\_nspace\_t*)

10     Combined length of the cluster identifier and namespace must be less than **PMIX\_MAX\_NSLEN-2**.

11     **Parse a multi-cluster namespace**

12     Parse a multi-cluster identifier into its cluster ID and namespace parts.

13     *PMIx v4.0*

C

13     **PMIX\_MULTICLUSTER\_NSPACE\_PARSE (m, n, r)**

C

14     IN    m

15            *pmix\_nspace\_t* structure containing the multi-cluster identifier (pointer to  
16            *pmix\_nspace\_t*)

17     IN    n

18            Location where the cluster ID is to be stored (*pmix\_nspace\_t*)

19     IN    n

20            Location where the namespace is to be stored (*pmix\_nspace\_t*)

21     **3.2.5 Process State Structure**

22     *PMIx v2.0*    The *pmix\_proc\_state\_t* structure is a *uint8\_t* type for process state values. The following  
23       constants can be used to set a variable of the type *pmix\_proc\_state\_t*.

24                  **Advice to users**

24     The fine-grained nature of the following constants may exceed the ability of an RM to provide  
25       updated process state values during the process lifetime. This is particularly true of states for  
26       short-lived processes.

```

1   PMIX_PROC_STATE_UNDEF      Undefined process state.
2   PMIX_PROC_STATE_PREPPED    Process is ready to be launched.
3   PMIX_PROC_STATE_LAUNCH_UNDERWAY  Process launch is underway.
4   PMIX_PROC_STATE_RESTART     Process is ready for restart.
5   PMIX_PROC_STATE_TERMINATE  Process is marked for termination.
6   PMIX_PROC_STATE_RUNNING    Process has been locally fork'ed by the RM.
7   PMIX_PROC_STATE_CONNECTED  Process has connected to PMIx server.
8   PMIX_PROC_STATE_UNTERMINATED Define a "boundary" between the terminated states
9       and PMIX_PROC_STATE_CONNECTED so users can easily and quickly determine if a
10      process is still running or not. Any value less than this constant means that the process has not
11      terminated.
12  PMIX_PROC_STATE_TERMINATED  Process has terminated and is no longer running.
13  PMIX_PROC_STATE_ERROR      Define a boundary so users can easily and quickly determine if
14      a process abnormally terminated. Any value above this constant means that the process has
15      terminated abnormally.
16  PMIX_PROC_STATE_KILLED_BY_CMD Process was killed by a command.
17  PMIX_PROC_STATE_ABORTED    Process was aborted by a call to PMIx_Abort.
18  PMIX_PROC_STATE_FAILED_TO_START Process failed to start.
19  PMIX_PROC_STATE_ABORTED_BY_SIG Process aborted by a signal.
20  PMIX_PROC_STATE_TERM_WO_SYNC Process exited without calling PMIx_Finalize.
21  PMIX_PROC_STATE_COMM_FAILED Process communication has failed.
22  PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED Process exceeded a specified sensor
23      limit.
24  PMIX_PROC_STATE_CALLED_ABORT Process called PMIx_Abort.
25  PMIX_PROC_STATE_HEARTBEAT_FAILED Process failed to send heartbeat within
26      specified time limit.
27  PMIX_PROC_STATE_MIGRATING    Process failed and is waiting for resources before
28      restarting.
29  PMIX_PROC_STATE_CANNOT_RESTART Process failed and cannot be restarted.
30  PMIX_PROC_STATE_TERM_NON_ZERO Process exited with a non-zero status.
31  PMIX_PROC_STATE_FAILED_TO_LAUNCH Unable to launch process.

```

### 32 3.2.6 Process Information Structure

33 The **pmix\_proc\_info\_t** structure defines a set of information about a specific process  
 34 including it's name, location, and state.

```
1  typedef struct pmix_proc_info {
2      /** Process structure */
3      pmix_proc_t proc;
4      /** Hostname where process resides */
5      char *hostname;
6      /** Name of the executable */
7      char *executable_name;
8      /** Process ID on the host */
9      pid_t pid;
10     /** Exit code of the process. Default: 0 */
11     int exit_code;
12     /** Current state of the process */
13     pmix_proc_state_t state;
14 } pmix_proc_info_t;
```

### 3.2.6.1 Process information structure support macros

The following macros are provided to support the [pmix\\_proc\\_info\\_t](#) structure.

#### Initialize the process information structure

Initialize the [pmix\\_proc\\_info\\_t](#) fields.

PMIx v2.0

```
PMIX_PROC_INFO_CONSTRUCT (m)
```

IN m

Pointer to the structure to be initialized (pointer to [pmix\\_proc\\_info\\_t](#))

#### Destruct the process information structure

Destruct the [pmix\\_proc\\_info\\_t](#) fields.

PMIx v2.0

```
PMIX_PROC_INFO_DESTRUCT (m)
```

IN m

Pointer to the structure to be destructed (pointer to [pmix\\_proc\\_info\\_t](#))

```
1      Create a process information array
2      Allocate and initialize a pmix_proc_info_t array.
3      PMIx v2.0   ▼———— C —————▶
4          INOUT m
5          Address where the pointer to the array of pmix_proc_info_t structures shall be stored
6          (handle)
7          IN n
8          Number of structures to be allocated (size_t)
9      Free a process information structure
10     Release a pmix_proc_info_t structure.
11     PMIx v2.0   ▼———— C —————▶
12     IN m
13     Pointer to a pmix_proc_info_t structure (handle)
14     Free a process information array
15     Release an array of pmix_proc_info_t structures.
16     PMIx v2.0   ▼———— C —————▶
17     IN m
18     Pointer to the array of pmix_proc_info_t structures (handle)
19     IN n
20     Number of structures in the array (size_t)
```

### 1 3.2.7 Job State Structure

2 *PMIx v4.0* The `pmix_job_state_t` structure is a `uint8_t` type for job state values. The following  
3 constants can be used to set a variable of the type `pmix_job_state_t`.

#### Advice to users

4 The fine-grained nature of the following constants may exceed the ability of an RM to provide  
5 updated job state values during the job lifetime. This is particularly true for short-lived jobs.

6     `PMIX_JOB_STATE_UNDEF`     Undefined job state.  
7     `PMIX_JOB_STATE_AWAITING_ALLOC`     Job is waiting for resources to be allocated to it.  
8     `PMIX_JOB_STATE_LAUNCH_UNDERWAY`     Job launch is underway.  
9     `PMIX_JOB_STATE_RUNNING`     All processes in the job have been spawned and are executing.  
10     `PMIX_JOB_STATE_SUSPENDED`     All processes in the job have been suspended.  
11     `PMIX_JOB_STATE_CONNECTED`     All processes in the job have connected to their PMIx  
12         server.  
13     `PMIX_JOB_STATE_UNTERMINATED`     Define a “boundary” between the terminated states  
14         and `PMIX_JOB_STATE_TERMINATED` so users can easily and quickly determine if a job is  
15         still running or not. Any value less than this constant means that the job has not terminated.  
16     `PMIX_JOB_STATE_TERMINATED`     All processes in the job have terminated and are no  
17         longer running - typically will be accompanied by the job exit status in response to a query.  
18     `PMIX_JOB_STATE_TERMINATED_WITH_ERROR`     Define a boundary so users can easily  
19         and quickly determine if a job abnormally terminated - typically will be accompanied by a  
20         job-related error code in response to a query. Any value above this constant means that the job  
21         terminated abnormally.

### 22 3.2.8 Value Structure

23     The `pmix_value_t` structure is used to represent the value passed to `PMIx_Put` and retrieved  
24     by `PMIx_Get`, as well as many of the other PMIx functions.

25     A collection of values may be specified under a single key by passing a `pmix_value_t`  
26     containing an array of type `pmix_data_array_t`, with each array element containing its own  
27     object. All members shown below were introduced in version 1 of the standard unless otherwise  
28     marked.

*PMIx v1.0*

```

1  typedef struct pmix_value {
2      pmix_data_type_t type;
3      union {
4          bool flag;
5          uint8_t byte;
6          char *string;
7          size_t size;
8          pid_t pid;
9          int integer;
10         int8_t int8;
11         int16_t int16;
12         int32_t int32;
13         int64_t int64;
14         unsigned int uint;
15         uint8_t uint8;
16         uint16_t uint16;
17         uint32_t uint32;
18         uint64_t uint64;
19         float fval;
20         double dval;
21         struct timeval tv;
22         time_t time;                                // version 2.0
23         pmix_status_t status;                      // version 2.0
24         pmix_rank_t rank;                          // version 2.0
25         pmix_proc_t *proc;                         // version 2.0
26         pmix_byte_object_t bo;
27         pmix_persistence_t persist;                // version 2.0
28         pmix_scope_t scope;                       // version 2.0
29         pmix_data_range_t range;                  // version 2.0
30         pmix_proc_state_t state;                 // version 2.0
31         pmix_proc_info_t *pinfo;                 // version 2.0
32         pmix_data_array_t *darray;                // version 2.0
33         void *ptr;                               // version 2.0
34         pmix_alloc_directive_t adir;              // version 2.0
35     } data;
36 } pmix_value_t;

```

### 37 3.2.8.1 Value structure support macros

38 The following macros are provided to support the [pmix\\_value\\_t](#) structure.

```

1      Initialize the value structure
2      Initialize the pmix\_value\_t fields.
3          PMIx v1.0   ▼----- C -----▼
4          PMIX_VALUE_CONSTRUCT(m)    C
5          IN   m
6          Pointer to the structure to be initialized (pointer to pmix\_value\_t)
7
8      Destruct the value structure
9      Destruct the pmix\_value\_t fields.
10     PMIx v1.0  ▼----- C -----▼
11     PMIX_VALUE_DESTRUCT(m)    C
12     IN   m
13     Pointer to the structure to be destructed (pointer to pmix\_value\_t)
14
15      Create a value array
16      Allocate and initialize an array of pmix\_value\_t structures.
17     PMIx v1.0  ▼----- C -----▼
18     PMIX_VALUE_CREATE(m, n)    C
19     INOUT m
20     Address where the pointer to the array of pmix\_value\_t structures shall be stored (handle)
21     IN   n
22     Number of structures to be allocated (size\_t)
23
24      Free a value structure
25      Release a pmix\_value\_t structure.
26     PMIx v4.0  ▼----- C -----▼
27     PMIX_VALUE_RELEASE(m)    C
28     IN   m
29     Pointer to a pmix\_value\_t structure (handle)

```

1      **Free a value array**

2      Release an array of `pmix_value_t` structures.

3      *PMIx v1.0*

4      `PMIX_VALUE_FREE(m, n)`

C

C

5      **IN m**

6      Pointer to the array of `pmix_value_t` structures (handle)

7      **IN n**

8      Number of structures in the array (`size_t`)

9      **Load a value structure**

10     Load data into a `pmix_value_t` structure.

11     *PMIx v2.0*

12     `PMIX_VALUE_LOAD(v, d, t);`

C

C

13     **IN v**

14     The `pmix_value_t` into which the data is to be loaded (pointer to `pmix_value_t`)

15     **IN d**

16     Pointer to the data value to be loaded (handle)

17     **IN t**

18     Type of the provided data value (`pmix_data_type_t`)

19     This macro simplifies the loading of data into a `pmix_value_t` by correctly assigning values to  
20    the structure's fields.

19     **Advice to users**

20     The data will be copied into the `pmix_value_t` - thus, any data stored in the source value can be  
modified or free'd without affecting the copied data once the macro has completed.

1      **Unload a value structure**  
2      Unload data from a `pmix_value_t` structure.

PMIx v2.2

3      `PMIX_VALUE_UNLOAD(r, v, d, t);`

C

C

4      **OUT r**

5      Status code indicating result of the operation `pmix_status_t`

6      **IN v**

7      The `pmix_value_t` from which the data is to be unloaded (pointer to `pmix_value_t`)

8      **INOUT d**

9      Pointer to the location where the data value is to be returned (handle)

10     **INOUT t**

11     Pointer to return the data type of the unloaded value (handle)

12    This macro simplifies the unloading of data from a `pmix_value_t`.

▼                  **Advice to users**                  ▼

13    Memory will be allocated and the data will be in the `pmix_value_t` returned - the source  
14    `pmix_value_t` will not be altered.

15    **Transfer data between value structures**

16    Transfer the data value between two `pmix_value_t` structures.

PMIx v2.0

17    `PMIX_VALUE_XFER(r, d, s);`

C

C

18    **OUT r**

19    Status code indicating success or failure of the transfer (`pmix_status_t`)

20    **IN d**

21    Pointer to the `pmix_value_t` destination (handle)

22    **IN s**

23    Pointer to the `pmix_value_t` source (handle)

24    This macro simplifies the transfer of data between two `pmix_value_t` structures, ensuring that  
25    all fields are properly copied.

▼                  **Advice to users**                  ▼

26    The data will be copied into the destination `pmix_value_t` - thus, any data stored in the source  
27    value can be modified or free'd without affecting the copied data once the macro has completed.

```
1     Retrieve a numerical value from a value struct
2     Retrieve a numerical value from a pmix_value_t structure.
3
4     PMIx v3.0   C
5         PMIX_VALUE_GET_NUMBER(s, m, n, t)   C
6
7     OUT s           C
8         Status code for the request (pmix_status_t)
9     IN m            C
10    Pointer to the pmix_value_t structure (handle)
11    OUT n           C
12    Variable to be set to the value (match expected type)
13    IN t            C
14    Type of number expected in m (pmix_data_type_t)
15
16    Sets the provided variable equal to the numerical value contained in the given pmix_value_t,
17    returning success if the data type of the value matches the expected type and
18    PMIX_ERR_BAD_PARAM if it doesn't
```

### 3.2.9 Info Structure

```
16      The pmix_info_t structure defines a key/value pair with associated directive. All fields were
17      defined in version 1.0 unless otherwise marked.
```

```
18      PMIx v1.0   C
19          typedef struct pmix_info_t {   C
20                  pmix_key_t key;   C
21                  pmix_info_directives_t flags; // version 2.0   C
22                  pmix_value_t value;   C
23          } pmix_info_t;   C
```

#### 3.2.9.1 Info structure support macros

```
24      The following macros are provided to support the pmix_info_t structure.
```

##### Initialize the info structure

```
25      Initialize the pmix_info_t fields.
```

```
26
27      PMIx v1.0   C
28          PMIX_INFO_CONSTRUCT(m)   C
29
30      IN m           C
31      Pointer to the structure to be initialized (pointer to pmix_info_t)
```

1                   **Destruct the info structure**  
2                   Destruct the `pmix_info_t` fields.

PMIx v1.0

6

3 PMIX\_INFO\_DESTRUCT(m)

9

4 IN

Pointer to the structure to be destructed (pointer to **pmix\_info\_t**)

6 Create an info array

Allocate and initialize an array of info structures.

PMIx v1.0

6

### PMIX\_INFO\_CREATE(m, n)

9

9           **INOUT** *m*

Address where the pointer to the array of `pmix_info_t` structures shall be stored (handle)

IN n

Number of structures to be allocated (**size t**)

## 13 Free an info array

Release an array of `pmix_info_t` structures.

PMIx v1.0

0

**PMIX INFO FREE(m, n)**

6

16 IN m

Pointer to the array of `pmix_info_t` structures (handle)

IN n

Number of structures in the array (**size\_t**)

## 20 Load key and value data into a info struct

PMIx v1.0

1            C  
2    **PMIX\_INFO\_LOAD**(*v*, *k*, *d*, *t*);        C  
3            C

2    **IN**    *v*  
3      Pointer to the **pmix\_info\_t** into which the key and data are to be loaded (pointer to  
4      **pmix\_info\_t**)  
5    **IN**    *k*  
6      String key to be loaded - must be less than or equal to **PMIX\_MAX\_KEYLEN** in length  
7      (handle)  
8    **IN**    *d*  
9      Pointer to the data value to be loaded (handle)  
10   **IN**    *t*  
11     Type of the provided data value (**pmix\_data\_type\_t**)

12   This macro simplifies the loading of key and data into a **pmix\_info\_t** by correctly assigning  
13   values to the structure's fields.

#### Advice to users

14   Both key and data will be copied into the **pmix\_info\_t** - thus, the key and any data stored in the  
15   source value can be modified or free'd without affecting the copied data once the macro has  
16   completed.

### Copy data between info structures

18   Copy all data (including key, value, and directives) between two **pmix\_info\_t** structures.

PMIx v2.0

19   C  
20   **PMIX\_INFO\_XFER**(*d*, *s*);        C  
21            C

20   **IN**    *d*  
21      Pointer to the destination **pmix\_info\_t** (pointer to **pmix\_info\_t**)  
22   **IN**    *s*  
23      Pointer to the source **pmix\_info\_t** (pointer to **pmix\_info\_t**)

24   This macro simplifies the transfer of data between two **pmix\_info\_t** structures.

#### Advice to users

25   All data (including key, value, and directives) will be copied into the destination **pmix\_info\_t** -  
26   thus, the source **pmix\_info\_t** may be free'd without affecting the copied data once the macro  
27   has completed.

1      **Test a boolean info struct**

2      A special macro for checking if a boolean `pmix_info_t` is `true`.

3      *PMIx v2.0*

C

4      `PMIX_INFO_TRUE (m)`

C

5      **IN    m**

6      Pointer to a `pmix_info_t` structure (handle)

7      A `pmix_info_t` structure is considered to be of type `PMIX_BOOL` and value `true` if:

- 8      • the structure reports a type of `PMIX_UNDEF`, or  
      • the structure reports a type of `PMIX_BOOL` and the data flag is `true`

9      **3.2.10 Info Type Directives**

10     *PMIx v2.0*    The `pmix_info_directives_t` structure is a `uint32_t` type that defines the behavior of  
11      command directives via `pmix_info_t` arrays. By default, the values in the `pmix_info_t`  
12      array passed to a PMIx are *optional*.

13      **Advice to users**

14      A PMIx implementation or PMIx-enabled RM may ignore any `pmix_info_t` value passed to a  
15      PMIx API that it does not support or does not recognize if it is not explicitly marked as  
16      `PMIX_INFO_REQD`. This is because the values specified default to optional, meaning they can be  
17      ignored in such circumstances. This may lead to unexpected behavior when porting between  
18      environments or PMIx implementations if the user is relying on the behavior specified by the  
19      `pmix_info_t` value. Users relying on the behavior defined by the `pmix_info_t` are advised to  
set the `PMIX_INFO_REQD` flag using the `PMIX_INFO_REQUIRED` macro.

20      **Advice to PMIx library implementers**

21      The top 16-bits of the `pmix_info_directives_t` are reserved for internal use by PMIx  
22      library implementers - the PMIx standard will *not* specify their intent, leaving them for customized  
23      use by implementers. Implementers are advised to use the provided `PMIX_INFO_IS_REQUIRED`  
24      macro for testing this flag, and must return `PMIX_ERR_NOT_SUPPORTED` as soon as possible to  
the caller if the required behavior is not supported.

1 The following constants were introduced in version 2.0 (unless otherwise marked) and can be used  
2 to set a variable of the type `pmix_info_directives_t`.

3 **PMIX\_INFO\_REQD** The behavior defined in the `pmix_info_t` array is required, and not  
4 optional. This is a bit-mask value.

5 **PMIX\_INFO\_ARRAY\_END** Mark that this `pmix_info_t` struct is at the end of an array  
6 created by the `PMIX_INFO_CREATE` macro. This is a bit-mask value.

7 **PMIX\_INFO\_DIR\_RESERVED** A bit-mask identifying the bits reserved for internal use by  
8 implementers - these currently are set as `0xfffff0000`.

### Advice to PMIx server hosts

9 Host environments are advised to use the provided `PMIX_INFO_IS_REQUIRED` macro for  
10 testing this flag and must return `PMIX_ERR_NOT_SUPPORTED` as soon as possible to the caller if  
11 the required behavior is not supported.

#### 3.2.10.1 Info Directive support macros

13 The following macros are provided to support the setting and testing of `pmix_info_t` directives.

##### Mark an info structure as required

15 Set the `PMIX_INFO_REQD` flag in a `pmix_info_t` structure.

PMIx v2.0

16 `PMIX_INFO_REQUIRED(info);`

17 **IN info**

18 Pointer to the `pmix_info_t` (pointer to `pmix_info_t`)

19 This macro simplifies the setting of the `PMIX_INFO_REQD` flag in `pmix_info_t` structures.

##### Mark an info structure as optional

21 Unsets the `PMIX_INFO_REQD` flag in a `pmix_info_t` structure.

PMIx v2.0

22 `PMIX_INFO_OPTIONAL(info);`

23 **IN info**

24 Pointer to the `pmix_info_t` (pointer to `pmix_info_t`)

25 This macro simplifies marking a `pmix_info_t` structure as *optional*.

1      **Test an info structure for *required* directive**

2      Test the `PMIX_INFO_REQD` flag in a `pmix_info_t` structure, returning `true` if the flag is set.

3      *PMIx v2.0*

C

4      `PMIX_INFO_IS_REQUIRED(info);`

C

5      **IN    info**

6      Pointer to the `pmix_info_t` (pointer to `pmix_info_t`)

7      This macro simplifies the testing of the required flag in `pmix_info_t` structures.

8      **Test an info structure for *optional* directive**

9      Test a `pmix_info_t` structure, returning `true` if the structure is *optional*.

10     *PMIx v2.0*

C

11     `PMIX_INFO_IS_OPTIONAL(info);`

C

12     **IN    info**

13     Pointer to the `pmix_info_t` (pointer to `pmix_info_t`)

14     Test the `PMIX_INFO_REQD` flag in a `pmix_info_t` structure, returning `true` if the flag is *not* set.

15     **Test an info structure for *end of array* directive**

16     Test a `pmix_info_t` structure, returning `true` if the structure is at the end of an array created by the `PMIX_INFO_CREATE` macro.

17     *PMIx v2.2*

C

18     `PMIX_INFO_IS_END(info);`

C

19     **IN    info**

20     Pointer to the `pmix_info_t` (pointer to `pmix_info_t`)

21     This macro simplifies the testing of the end-of-array flag in `pmix_info_t` structures.

## 3.2.11 Environmental Variable Structure

Define a structure for specifying environment variable modifications. Standard environment variables (e.g., `PATH`, `LD_LIBRARY_PATH`, and `LD_PRELOAD`) take multiple arguments separated by delimiters. Unfortunately, the delimiters depend upon the variable itself - some use semi-colons, some colons, etc. Thus, the operation requires not only the name of the variable to be modified and the value to be inserted, but also the separator to be used when composing the aggregate value.

```
typedef struct {  
    char *envar;  
    char *value;  
    char separator;  
} pmix_envar_t;
```

### Environmental variable support macros

The following macros are provided to support the `pmix_envar_t` structure.

#### Initialize the envar structure

Initialize the `pmix_envar_t` fields.

PMIx v3.0

```
PMIX_ENVAR_CONSTRUCT(m)
```

IN m

Pointer to the structure to be initialized (pointer to `pmix_envar_t`)

#### Destruct the envar structure

Clear the `pmix_envar_t` fields.

PMIx v3.0

```
PMIX_ENVAR_DESTRUCT(m)
```

IN m

Pointer to the structure to be destructed (pointer to `pmix_envar_t`)

```

1      Create an envar array
2      Allocate and initialize an array of pmix_envar_t structures.
3      PMIx v3.0   C
4          INOUT m
5              Address where the pointer to the array of pmix_envar_t structures shall be stored (handle)
6          IN n
7              Number of structures to be allocated (size_t)
8      Free an envar array
9      Release an array of pmix_envar_t structures.
10     PMIx v3.0   C
11     IN m
12         Pointer to the array of pmix_envar_t structures (handle)
13     IN n
14         Number of structures in the array (size_t)
15     Load an envar structure
16     Load values into a pmix_envar_t.
17     PMIx v2.0   C
18     IN m
19         Pointer to the structure to be loaded (pointer to pmix_envar_t)
20     IN e
21         Environmental variable name (char*)
22     IN v
23         Value of variable (char*)
24     IN v
25         Separator character (char)

```

## 3.2.12 Byte Object Type

The `pmix_byte_object_t` structure describes a raw byte sequence.

PMIx v1.0

```
3     typedef struct pmix_byte_object {  
4         char *bytes;  
5         size_t size;  
6     } pmix_byte_object_t;
```

C

C

### 3.2.12.1 Byte object support macros

The following macros support the `pmix_byte_object_t` structure.

#### Initialize the byte object structure

Initialize the `pmix_byte_object_t` fields.

PMIx v2.0

```
11    PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

C

C

IN m

Pointer to the structure to be initialized (pointer to `pmix_byte_object_t`)

#### Destruct the byte object structure

Clear the `pmix_byte_object_t` fields.

PMIx v2.0

```
16    PMIX_BYTE_OBJECT_DESTRUCT(m)
```

C

C

IN m

Pointer to the structure to be destructed (pointer to `pmix_byte_object_t`)

#### Create a byte object structure

Allocate and intialize an array of `pmix_byte_object_t` structures.

PMIx v2.0

```
21    PMIX_BYTE_OBJECT_CREATE(m, n)
```

C

C

INOUT m

Address where the pointer to the array of `pmix_byte_object_t` structures shall be stored  
(handle)

IN n

Number of structures to be allocated (`size_t`)

1      **Free a byte object array**

2      Release an array of `pmix_byte_object_t` structures.

3      *PMIx v2.0*

C

3      `PMIX_BYTE_OBJECT_FREE(m, n)`

C

4      **IN m**

5      Pointer to the array of `pmix_byte_object_t` structures (handle)

6      **IN n**

7      Number of structures in the array (`size_t`)

8      **Load a byte object structure**

9      Load values into a `pmix_byte_object_t`.

10     *PMIx v2.0*

C

10     `PMIX_BYTE_OBJECT_LOAD(b, d, s)`

C

11     **IN b**

12     Pointer to the structure to be loaded (pointer to `pmix_byte_object_t`)

13     **IN d**

14     Pointer to the data to be loaded (`char*`)

15     **IN s**

16     Number of bytes in the data array (`size_t`)

17     **3.2.13 Data Array Structure**

18     The `pmix_data_array_t` structure defines an array data structure.

19     *PMIx v2.0*

C

```
19     typedef struct pmix_data_array {
20        pmix_data_type_t type;
21        size_t size;
22        void *array;
23     } pmix_data_array_t;
```

C

24     **3.2.13.1 Data array support macros**

25     The following macros support the `pmix_data_array_t` structure.

1           **Initialize a data array structure**

2        Initialize the [pmix\\_data\\_array\\_t](#) fields, allocating memory for the array of the indicated type.

3           *PMIx v2.2*      C

4           **PMIX\_DATA\_ARRAY\_CONSTRUCT (m, n, t)**

5           C

6           **IN m**

7           Pointer to the structure to be initialized (pointer to [pmix\\_data\\_array\\_t](#))

8           **IN n**

9           Number of elements in the array ([size\\_t](#))

10          **IN t**

11          PMIx data type of the array elements ([pmix\\_data\\_type\\_t](#))

12           **Destruct a data array structure**

13        Destruct the [pmix\\_data\\_array\\_t](#), releasing the memory in the array.

14           *PMIx v2.2*      C

15           **PMIX\_DATA\_ARRAY\_DESTROY (m)**

16           C

17           **IN m**

18           Pointer to the structure to be destructed (pointer to [pmix\\_data\\_array\\_t](#))

19           **Create a data array structure**

20        Allocate memory for the [pmix\\_data\\_array\\_t](#) object itself, and then allocate memory for the  
21        array of the indicated type.

22           *PMIx v2.2*      C

23           **PMIX\_DATA\_ARRAY\_CREATE (m, n, t)**

24           C

19           **INOUT m**

20           Variable to be set to the address of the structure (pointer to [pmix\\_data\\_array\\_t](#))

21           **IN n**

22           Number of elements in the array ([size\\_t](#))

23           **IN t**

24           PMIx data type of the array elements ([pmix\\_data\\_type\\_t](#))

1      **Free a data array structure**

2      Release the memory in the array, and then release the `pmix_data_array_t` object itself.

3      `PMIx v2.2`

C

4      `PMIX_DATA_ARRAY_FREE (m)`

C

5      **IN    m**

6      Pointer to the structure to be released (pointer to `pmix_data_array_t`)

7      **3.2.14 Argument Array Macros**

8      The following macros support the construction and release of **NULL**-terminated argv arrays of  
9      strings.

10     **Argument array extension**

11     Append a string to a NULL-terminated, argv-style array of strings.

C

12     `PMIX_ARGV_APPEND (r, a, b);`

C

13     **OUT   r**

14     Status code indicating success or failure of the operation (`pmix_status_t`)

15     **INOUT a**

16     Argument list (pointer to NULL-terminated array of strings)

17     **IN    b**

18     Argument to append to the list (string)

19     This function helps the caller build the **argv** portion of `pmix_app_t` structure, arrays of keys for  
20    querying, or other places where argv-style string arrays are required.

21     **Advice to users**

20     The provided argument is copied into the destination array - thus, the source string can be free'd  
21    without affecting the array once the macro has completed.

1           **Argument array prepend**  
2       Prepend a string to a NULL-terminated, argv-style array of strings.

3           **PMIX\_ARGV\_PREPEND (r, a, b);**

4           **OUT r**

5           Status code indicating success or failure of the operation ([pmix\\_status\\_t](#))

6           **INOUT a**

7           Argument list (pointer to NULL-terminated array of strings)

8           **IN b**

9           Argument to append to the list (string)

10          This function helps the caller build the **argv** portion of [pmix\\_app\\_t](#) structure, arrays of keys for  
11        querying, or other places where argv-style string arrays are required.

▼————— C —————▼  
**Advice to users**  
▲————— C —————▲

12          The provided argument is copied into the destination array - thus, the source string can be free'd  
13        without affecting the array once the macro has completed.  
▲————— C —————▲

14           **Argument array extension - unique**

15          Append a string to a NULL-terminated, argv-style array of strings, but only if the provided  
16        argument doesn't already exist somewhere in the array.

17           **PMIX\_ARGV\_APPEND\_UNIQUE (r, a, b);**

18           **OUT r**

19           Status code indicating success or failure of the operation ([pmix\\_status\\_t](#))

20           **INOUT a**

21           Argument list (pointer to NULL-terminated array of strings)

22           **IN b**

23           Argument to append to the list (string)

24          This function helps the caller build the **argv** portion of [pmix\\_app\\_t](#) structure, arrays of keys for  
25        querying, or other places where argv-style string arrays are required.

▼————— C —————▼  
**Advice to users**  
▲————— C —————▲

26          The provided argument is copied into the destination array - thus, the source string can be free'd  
27        without affecting the array once the macro has completed.  
▲————— C —————▲

1           **Argument array release**  
2       Free an argv-style array and all of the strings that it contains.

3           **PMIX\_ARGV\_FREE(a);**

4           **IN a**

5           Argument list (pointer to NULL-terminated array of strings)

6       This function releases the array and all of the strings it contains.

7           **Argument array split**  
8       Split a string into a NULL-terminated argv array.

9           **PMIX\_ARGV\_SPLIT(a, b, c);**

10          **OUT a**

11          Resulting argv-style array (**char\*\***)

12          **IN b**

13          String to be split (**char\***)

14          **IN c**

15          Delimiter character (**char**)

16       Split an input string into a NULL-terminated argv array. Do not include empty strings in the  
17       resulting array.

18           **Advice to users**

19       All strings are inserted into the argv array by value; the newly-allocated array makes no references  
20       to the src\_string argument (i.e., it can be freed after calling this function without invalidating the  
output argv array)

```
1      Argument array join
2      Join all the elements of an argv array into a single newly-allocated string.
3      PMIX_ARGV_JOIN(a, b, c);
4      OUT a
5          Resulting string (char*)
6      IN b
7          Argv-style array to be joined (char**)
8      IN c
9          Delimiter character (char)
10     Join all the elements of an argv array into a single newly-allocated string.

11     Argument array count
12     Return the length of a NULL-terminated argv array.
13     PMIX_ARGV_COUNT(r, a);
14     OUT r
15         Number of strings in the array (integer)
16     IN a
17         Argv-style array (char**)
18     Count the number of elements in an argv array

19     Argument array copy
20     Copy an argv array, including copying all of its strings.
21     PMIX_ARGV_COPY(a, b);
22     OUT a
23         New argv-style array (char**)
24     IN b
25         Argv-style array (char**)
26     Copy an argv array, including copying all of its strings.
```

### 1 3.2.15 Set Environment Variable

#### 2 Summary

3 Set an environment variable in a **NULL**-terminated, env-style array.

4 **PMIX\_SETENV(r, name, value, env);**

5 **OUT r**

6 Status code indicating success or failure of the operation (**pmix\_status\_t**)

7 **IN name**

8 Argument name (string)

9 **IN value**

10 Argument value (string)

11 **INOUT env**

12 Environment array to update (pointer to array of strings)

#### 13 Description

14 Similar to **setenv** from the C API, this allows the caller to set an environment variable in the  
15 specified **env** array, which could then be passed to the **pmix\_app\_t** structure or any other  
16 destination.

#### Advice to users

17 The provided name and value are copied into the destination environment array - thus, the source  
18 strings can be free'd without affecting the array once the macro has completed.

## 19 3.3 Generalized Data Types Used for Packing/Unpacking

20 The **pmix\_data\_type\_t** structure is a **uint16\_t** type for identifying the data type for  
21 packing/unpacking purposes. New data type values introduced in this version of the Standard are  
22 shown in **magenta**.

#### Advice to PMIx library implementers

23 The following constants can be used to set a variable of the type **pmix\_data\_type\_t**. Data  
24 types in the PMIx Standard are defined in terms of the C-programming language. Implementers  
25 wishing to support other languages should provide the equivalent definitions in a  
26 language-appropriate manner. Additionally, a PMIx implementation may choose to add additional  
27 types.

```

1   PMIX_UNDEF      Undefined.
2   PMIX_BOOL       Boolean (converted to/from native true/false) (bool).
3   PMIX_BYTE       A byte of data (uint8_t).
4   PMIX_STRING     NULL terminated string (char*).
5   PMIX_SIZE       Size size_t.
6   PMIX_PID        Operating Process IDentifier (PID) (pid_t).
7   PMIX_INT        Integer (int).
8   PMIX_INT8       8-byte integer (int8_t).
9   PMIX_INT16      16-byte integer (int16_t).
10  PMIX_INT32      32-byte integer (int32_t).
11  PMIX_INT64      64-byte integer (int64_t).
12  PMIX_UINT       Unsigned integer (unsigned int).
13  PMIX_UINT8      Unsigned 8-byte integer (uint8_t).
14  PMIX_UINT16      Unsigned 16-byte integer (uint16_t).
15  PMIX_UINT32      Unsigned 32-byte integer (uint32_t).
16  PMIX_UINT64      Unsigned 64-byte integer (uint64_t).
17  PMIX_FLOAT       Float (float).
18  PMIX_DOUBLE      Double (double).
19  PMIX_TIMEVAL    Time value (struct timeval).
20  PMIX_TIME       Time (time_t).
21  PMIX_STATUS     Status code pmix_status_t.
22  PMIX_VALUE      Value (pmix_value_t).
23  PMIX_PROC       Process (pmix_proc_t).
24  PMIX_APP        Application context.
25  PMIX_INFO       Info object.
26  PMIX_PDATA      Pointer to data.
27  PMIX_BUFFER      Buffer.
28  PMIX_BYTE_OBJECT Byte object (pmix_byte_object_t).
29  PMIX_KVAL        Key/value pair.
30  PMIX_PERSIST     Persistance (pmix_persistence_t).
31  PMIX_POINTER     Pointer to an object (void*).
32  PMIX_SCOPE       Scope (pmix_scope_t).
33  PMIX_DATA_RANGE Range for data (pmix_data_range_t).
34  PMIX_COMMAND     PMIx command code (used internally).
35  PMIX_INFO_DIRECTIVES Directives flag for pmix_info_t  

36  (pmix_info_directives_t).
37  PMIX_DATA_TYPE   Data type code (pmix_data_type_t).
38  PMIX_PROC_STATE  Process state (pmix_proc_state_t).
39  PMIX_PROC_INFO   Process information (pmix_proc_info_t).
40  PMIX_DATA_ARRAY  Data array (pmix_data_array_t).
41  PMIX_PROC_RANK   Process rank (pmix_rank_t).

```

```

1   PMIX_QUERY      Query structure (pmix\_query\_t).
2   PMIX_COMPRESSED_STRING  String compressed with zlib (char\*).
3   PMIX_ALLOC_DIRECTIVE Allocation directive (pmix\_alloc\_directive\_t).
4   PMIX_IOF_CHANNEL Input/output forwarding channel (pmix\_iof\_channel\_t).
5   PMIX_ENVAR      Environmental variable structure (pmix\_envar\_t).
6   PMIX_COORD      Structure containing fabric coordinates (pmix\_coord\_t).
7   PMIX_REGATTR    Structure supporting attribute registrations (pmix\_regattr\_t).
8   PMIX_REGEX      Regular expressions - can be a valid NULL-terminated string or an arbitrary
9     array of bytes.
10  PMIX_JOB_STATE Job state (pmix\_job\_state\_t).
11  PMIX_LINK_STATE Link state (pmix\_link\_state\_t).
12  PMIX_PROC_CPUSET Structure containing the binding bitmap of a process
13    (pmix\_cpuset\_t).
14  PMIX_GEOMETRY   Geometry structure containing the fabric coordinates of a specified
15    device. (pmix\_geometry\_t).
16  PMIX_DEVICE_DIST Structure containing the minimum and maximum relative distance
17    from the caller to a given fabric device. (pmix\_device\_distance\_t).
18  PMIX_ENDPOINT   Structure containing an assigned endpoint for a given fabric device.
19    (pmix\_endpoint\_t).
20  PMIX_DATA_TYPE_MAX A starting point for implementer-specific data types. Values above
21    this are guaranteed not to conflict with PMIx values. Definitions should always be based on
22    the PMIX\_DATA\_TYPE\_MAX constant and not a specific value as the value of the constant
23    may change.

```

## 3.4 General Callback Functions

PMIx provides blocking and nonblocking versions of most APIs. In the nonblocking versions, a callback is activated upon completion of the the operation. This section describes many of those callbacks.

### 3.4.1 Release Callback Function

#### Summary

The [pmix\\_release\\_cbfnc\\_t](#) is used by the [pmix\\_modex\\_cbfnc\\_t](#) and [pmix\\_info\\_cbfnc\\_t](#) operations to indicate that the callback data may be reclaimed/freed by the caller.

#### Format

*PMIx v1.0*

```

34  typedef void (*pmix_release_cbfnc_t)
35    (void *cbdata);

```

#### INOUT cbdata

Callback data passed to original API call (memory reference)

1           **Description**

2       Since the data is “owned” by the host server, provide a callback function to notify the host server  
3       that we are done with the data so it can be released.

4           **3.4.2 Op Callback Function**

5           **Summary**

6       The `pmix_op_cbfunc_t` is used by operations that simply return a status.

PMIx v1.0

C

```
7       typedef void (*pmix_op_cbfunc_t)
8              (pmix_status_t status, void *cbdata);
```

C

9           **IN    status**

10          Status associated with the operation (handle)

11           **IN    cbdata**

12          Callback data passed to original API call (memory reference)

13           **Description**

14       Used by a wide range of PMIx API’s including `PMIx_Fence_nb`,

15       `pmix_server_client_connected_fn_t`, `PMIx_server_register_nspace`. This  
16       callback function is used to return a status to an often nonblocking operation.

17           **3.4.3 Value Callback Function**

18           **Summary**

19       The `pmix_value_cbfunc_t` is used by `PMIx_Get_nb` to return data.

PMIx v1.0

C

```
20       typedef void (*pmix_value_cbfunc_t)
21              (pmix_status_t status,
22              pmix_value_t *kv, void *cbdata);
```

C

23           **IN    status**

24          Status associated with the operation (handle)

25           **IN    kv**

26          Key/value pair representing the data (`pmix_value_t`)

27           **IN    cbdata**

28          Callback data passed to original API call (memory reference)

1    **Description**

2    A callback function for calls to `PMIx_Get_nb`. The *status* indicates if the requested data was  
3    found or not. A pointer to the `pmix_value_t` structure containing the found data is returned.  
4    The pointer will be `NULL` if the requested data was not found.

5    **3.4.4 Info Callback Function**

6    **Summary**

7    The `pmix_info_cbfunc_t` is a general information callback used by various APIs.

PMIx v2.0

C

```
8       typedef void (*pmix_info_cbfunc_t)
9              (pmix_status_t status,
10              pmix_info_t info[], size_t ninfo,
11              void *cbdata,
12              pmix_release_cbfunc_t release_fn,
13              void *release_cbdata);
```

C

14    **IN status**

15    Status associated with the operation (`pmix_status_t`)

16    **IN info**

17    Array of `pmix_info_t` returned by the operation (pointer)

18    **IN ninfo**

19    Number of elements in the *info* array (`size_t`)

20    **IN cbdata**

21    Callback data passed to original API call (memory reference)

22    **IN release\_fn**

23    Function to be called when done with the *info* data (function pointer)

24    **IN release\_cbdata**

25    Callback data to be passed to *release\_fn* (memory reference)

26    **Description**

27    The *status* indicates if requested data was found or not. An array of `pmix_info_t` will contain  
28    the key/value pairs.

29    **3.4.5 Handler registration callback function**

30    **Summary**

31    Callback function for calls to register handlers, e.g., event notification and IOF requests.

```
1      Format  
2      PMIx v3.0   C  
3      typedef void (*pmix_hdlr_reg_cbfunc_t)  
4          (pmix_status_t status,  
5             size_t refid,  
6             void *cbdata);  
7      C  
8      IN  status  
9          PMIX_SUCCESS or an appropriate error constant (pmix_status_t)  
10     IN  refid  
11        reference identifier assigned to the handler by PMIx, used to deregister the handler (size_t)  
12     IN  cbdata  
13        object provided to the registration call (pointer)  
  
14    Description  
15      Callback function for calls to register handlers, e.g., event notification and IOF requests.
```

## 3.5 PMIx Datatype Value String Representations

Provide a string representation for several types of values. Note that the provided string is statically defined and must NOT be **free**'d.

```
17    Summary  
18    String representation of a pmix_status_t.  
19    PMIx v1.0   C  
20    const char*  
21    PMIx_Error_string(pmix_status_t status);  
22    C
```

```
21    Summary  
22    String representation of a pmix_proc_state_t.  
23    PMIx v2.0   C  
24    const char*  
25    PMIx_Proc_state_string(pmix_proc_state_t state);  
26    C
```

```
1      Summary  
2      String representation of a pmix\_scope\_t.  
3      PMIx v2.0   ┌─────────────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────────────┐  
4          const char*  
5          PMIx_Scope_string(pmix_scope_t scope);  
6          ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
  
7      Summary  
8      String representation of a pmix\_persistence\_t.  
9      PMIx v2.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
10     const char*  
11     PMIx_Persistence_string(pmix_persistence_t persist);  
12     ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
  
13     Summary  
14     String representation of a pmix\_data\_range\_t.  
15     PMIx v2.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
16     const char*  
17     PMIx_Data_range_string(pmix_data_range_t range);  
18     ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
  
19     Summary  
20     String representation of a pmix\_info\_directives\_t.  
21     PMIx v2.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
22     const char*  
23     PMIx_Info_directives_string(pmix_info_directives_t directives);  
24     ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
  
25     Summary  
26     String representation of a pmix\_data\_type\_t.  
27     PMIx v2.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
28     const char*  
29     PMIx_Data_type_string(pmix_data_type_t type);  
30     ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐
```

```
1      Summary  
2      String representation of a pmix\_alloc\_directive\_t.  
3      PMIx v2.0   ┌─────────────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────────────┐  
4          const char*  
5          PMIx_AllocDirective_string(pmix_alloc_directive_t directive);  
6          ┌─────────────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
  
5      Summary  
6      String representation of a pmix\_ifof\_channel\_t.  
7      PMIx v3.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
8          const char*  
9          PMIx_IFOF_Channel_string(pmix_ifof_channel_t channel);  
10         ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
  
9      Summary  
10     String representation of a pmix\_job\_state\_t.  
11     PMIx v4.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
12          const char*  
13          PMIx_JobState_string(pmix_job_state_t state);  
14          ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────┐  
  
13     Summary  
14     String representation of a PMIx attribute.  
15     PMIx v4.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
16          const char*  
17          PMIx_GetAttribute_string(char *attributename);  
18          ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────┐  
  
17     Summary  
18     Return the PMIx attribute name corresponding to the given attribute string.  
19     PMIx v4.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
20          const char*  
21          PMIx_GetAttribute_name(char *attributestring);  
22          ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────┐
```

1           **Summary**  
2       String representation of a [pmix\\_link\\_state\\_t](#).

PMIx v4.0

C

3           **const char\***  
4       [PMIx\\_Link\\_state\\_string](#)(pmix\_link\_state\_t state);

C

## CHAPTER 4

# Client Initialization and Finalization

---

The PMIx library is required to be initialized and finalized around the usage of most PMIx functions or macros. The APIs that may be used outside of the initialized and finalized region are noted. All other APIs must be used inside this region.

There are three sets of initialization and finalization functions depending upon the role of the process in the PMIx Standard - those associated with the PMIx *client* are defined in this chapter. Similar functions corresponding to the roles of *server* and *tool* are defined in Chapters 16 and 17, respectively.

Note that a process can only call *one* of the initialization/finalization functional pairs from the set of three - e.g., a process that calls the client initialization function cannot also call the tool or server initialization functions, and must call the corresponding client finalization function. Regardless of the role assumed by the process, all processes have access to the client APIs. Thus, the *server* and *tool* roles can be considered supersets of the PMIx client.

## 4.1 PMIx\_Initialized

### Summary

Determine if the PMIx library has been initialized. This function may be used outside of the initialized and finalized region, and is usable by servers and tools in addition to clients.

### Format

PMIx v1.0

`int PMIx_Initialized(void)`

C

C

A value of 1 (true) will be returned if the PMIx library has been initialized, and 0 (false) otherwise.

### Rationale

The return value is an integer for historical reasons as that was the signature of prior PMI libraries.

### Description

Check to see if the PMIx library has been initialized using any of the init functions: `PMIx_Init`, `PMIx_server_init`, or `PMIx_tool_init`.

## 1 4.2 PMIx\_Get\_version

### 2 Summary

3 Get the PMIx version information. This function may be used outside of the initialized and  
4 finalized region, and is usable by servers and tools in addition to clients.

### 5 Format

6 *PMIx v1.0*      C  
7      `const char* PMIx_Get_version(void)`

### 8 Description

9 Get the PMIx version string. Note that the provided string is statically defined and must *not* be  
free'd.

## 10 4.3 PMIx\_Init

### 11 Summary

12 Initialize the PMIx client library

### 13 Format

14 *PMIx v1.2*      C  
15      `pmix_status_t`  
16      `PMIx_Init(pmix_proc_t *proc,`  
17                `pmix_info_t info[], size_t ninfo)`

### 18 INOUT proc

19      proc structure (handle)

### 20 IN info

21      Array of `pmix_info_t` structures (array of handles)

### 22 IN ninfo

23      Number of elements in the *info* array (`size_t`)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

## Optional Attributes

1 The following attributes are optional for implementers of PMIx libraries:

2 **PMIX\_USOCK\_DISABLE** "pmix.usock.disable" (bool)

3 Disable legacy UNIX socket (usock) support. If the library supports Unix socket  
4 connections, this attribute may be supported for disabling it.

5 **PMIX\_SOCKET\_MODE** "pmix.sockmode" (uint32\_t)

6 POSIX *mode\_t* (9 bits valid). If the library supports socket connections, this attribute may  
7 be supported for setting the socket mode.

8 **PMIX\_SINGLE\_LISTENER** "pmix.sing.listnr" (bool)

9 Use only one rendezvous socket, letting priorities and/or environment parameters select the  
10 active transport. If the library supports multiple methods for clients to connect to servers,  
11 this attribute may be supported for disabling all but one of them.

12 **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (char\*)

13 If provided, directs that the TCP Uniform Resource Identifier (URI) be reported and indicates  
14 the desired method of reporting: '-' for stdout, '+' for stderr, or filename. If the library  
15 supports TCP socket connections, this attribute may be supported for reporting the URI.

16 **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)

17 Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to  
18 include when establishing the TCP connection. If the library supports TCP socket  
19 connections, this attribute may be supported for specifying the interfaces to be used.

20 **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)

21 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the  
22 TCP connection. If the library supports TCP socket connections, this attribute may be  
23 supported for specifying the interfaces that are *not* to be used.

24 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)

25 The IPv4 port to be used.. If the library supports IPV4 connections, this attribute may be  
26 supported for specifying the port to be used.

27 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)

28 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be  
29 supported for specifying the port to be used.

30 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)

31 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,  
32 this attribute may be supported for disabling it.

33 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)

34 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,  
35 this attribute may be supported for disabling it.

36 **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)

1            Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.



## 2            **Description**

3            Initialize the PMIx client, returning the process identifier assigned to this client's application in the  
4            provided **pmix\_proc\_t** struct. Passing a value of **NULL** for this parameter is allowed if the user  
5            wishes solely to initialize the PMIx system and does not require return of the identifier at that time.

6            When called, the PMIx client shall check for the required connection information of the local PMIx  
7            server and establish the connection. If the information is not found, or the server connection fails,  
8            then an appropriate error constant shall be returned.

9            If successful, the function shall return **PMIX\_SUCCESS** and fill the *proc* structure (if provided)  
10          with the server-assigned namespace and rank of the process within the application. In addition, all  
11          startup information provided by the resource manager shall be made available to the client process  
12          via subsequent calls to **PMIx\_Get**.

13          The PMIx client library shall be reference counted, and so multiple calls to **PMIx\_Init** are  
14          allowed by the standard. Thus, one way for an application process to obtain its namespace and rank  
15          is to simply call **PMIx\_Init** with a non-NULL *proc* parameter. Note that each call to  
16          **PMIx\_Init** must be balanced with a call to **PMIx\_Finalize** to maintain the reference count.

17          Each call to **PMIx\_Init** may contain an array of **pmix\_info\_t** structures passing directives to  
18          the PMIx client library as per the above attributes.

19          Multiple calls to **PMIx\_Init** shall not include conflicting directives. The **PMIx\_Init** function  
20          will return an error when directives that conflict with prior directives are encountered.

### 21          **4.3.1 Initialization events**

22          The following events are typically associated with calls to **PMIx\_Init**:

23          **PMIX\_MODEL\_DECLARED**      Model declared.

24          **PMIX\_MODEL\_RESOURCES**     Resource usage by a programming model has changed.

25          **PMIX\_OPENMP\_PARALLEL\_ENTERED**   An OpenMP parallel code region has been entered.

26          **PMIX\_OPENMP\_PARALLEL\_EXITED**   An OpenMP parallel code region has completed.

### 27          **4.3.2 Initialization attributes**

28          The following attributes influence the behavior of **PMIx\_Init**.

---

<sup>1</sup><http://libevent.org/>

### 4.3.2.1 Connection attributes

These attributes are used to describe a TCP socket for rendezvous with the local RM by passing them into the relevant initialization API - thus, they are not typically accessed via the [PMIx\\_Get](#) API.

```
PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)
    If provided, directs that the TCP URI be reported and indicates the desired method of
    reporting: '-' for stdout, '+' for stderr, or filename.
PMIX_TCP_URI "pmix.tcp.uri" (char*)
    The URI of the PMIx server to connect to, or a file name containing it in the form of
    file:<name of file containing it>.
PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*)
    Comma-delimited list of devices and/or CIDR notation to include when establishing the
    TCP connection.
PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*)
    Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
    TCP connection.
PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)
    The IPv4 port to be used..
PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)
    The IPv6 port to be used.
PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)
    Set to true to disable IPv4 family of addresses.
PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)
    Set to true to disable IPv6 family of addresses.
```

### 4.3.2.2 Programming model attributes

These attributes are associated with programming models.

```
PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*)
    Programming model being initialized (e.g., "MPI" or "OpenMP").
PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*)
    Programming model implementation ID (e.g., "OpenMPI" or "MPICH").
PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)
    Programming model version string (e.g., "2.1.1").
PMIX_THREADING_MODEL "pmix.threads" (char*)
    Threading model used (e.g., "pthreads").
PMIX_MODEL_NUM_THREADS "pmix.mdl.nthrds" (uint64_t)
    Number of active threads being used by the model.
PMIX_MODEL_NUM_CPUS "pmix.mdl.ncpu" (uint64_t)
    Number of cpus being used by the model.
PMIX_MODEL_CPU_TYPE "pmix.mdl.cputype" (char*)
    Granularity - "hwthread", "core", etc.
PMIX_MODEL_PHASE_NAME "pmix.mdl.phase" (char*)
```

```
1      User-assigned name for a phase in the application execution (e.g., "cfd reduction").  
2      PMIX_MODEL_PHASE_TYPE "pmix.mdl.ptype" (char*)  
3          Type of phase being executed (e.g., "matrix multiply").  
4      PMIX_MODEL_AFFINITY_POLICY "pmix.mdl.tap" (char*)  
5          Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread  
6          located on cpu close to master thread), "spread" (threads load-balanced across available  
7          cpus).
```

## 8 **4.4 PMIx\_Finalize**

### 9 **Summary**

10 Finalize the PMIx client library.

### 11 **Format**

12 *PMIx v1.0*

```
13     pmix_status_t  
14     PMIx_Finalize(const pmix_info_t info[], size_t ninfo)
```

15 **IN** **info**  
16 Array of **pmix\_info\_t** structures (array of handles)  
17 **IN** **ninfo**  
18 Number of elements in the *info* array (**size\_t**)

19 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

### 20           **Optional Attributes**

21 The following attributes are optional for implementers of PMIx libraries:

```
22     PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  
23         Execute a blocking fence operation before executing the specified operation.  
24         PMIx_Finalize does not include an internal barrier operation by default. This attribute  
25         directs PMIx_Finalize to execute a barrier as part of the finalize operation.
```

### 26 **Description**

27 Decrement the PMIx client library reference count. When the reference count reaches zero, the  
library will finalize the PMIx client, closing the connection with the local PMIx server and  
releasing all internally allocated memory.

## 1 4.4.1 Finalize attributes

2 The following attribute influences the behavior of [PMIx\\_Finalize](#).

3 **PMIX\_EMBED\_BARRIER "pmix.embed.barrier" (bool)**

4 Execute a blocking fence operation before executing the specified operation.

5 [PMIx\\_Finalize](#) does not include an internal barrier operation by default. This attribute  
6 directs [PMIx\\_Finalize](#) to execute a barrier as part of the finalize operation.

## CHAPTER 5

# Synchronization and Data Access Operations

1 Applications may need to synchronize their operations at various points in their execution.  
2 Depending on a variety of factors (e.g., the programming model and where the synchronization  
3 point lies), the application may choose to execute the operation using PMIx. This is particularly  
4 useful in situations where communication by other means is not yet available since PMIx relies on  
5 the host environment's infrastructure for such operations.

6 Synchronization operations also offer an opportunity for processes to exchange data at a known  
7 point in their execution. Where required, this can include information on communication endpoints  
8 for subsequent wireup of various messaging protocols.

9 This chapter covers both the synchronization and data retrieval functions provided under the PMIx  
10 Standard.

## 5.1 PMIx\_Fence

### Summary

13 Execute a blocking barrier across the processes identified in the specified array, collecting  
14 information posted via [PMIx\\_Put](#) as directed.

### Format

15 *PMIx v1.0*

C

```
16     pmix_status_t
17     PMIx_Fence(const pmix_proc_t procs[], size_t nprocs,
18                  const pmix_info_t info[], size_t ninfo);
```

19 **IN procs**  
20 Array of [pmix\\_proc\\_t](#) structures (array of handles)  
21 **IN nprocs**  
22 Number of elements in the *procs* array (integer)  
23 **IN info**  
24 Array of info structures (array of handles)  
25 **IN ninfo**  
26 Number of elements in the *info* array (integer)

27 Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

## Required Attributes

1 The following attributes are required to be supported by all PMIx libraries:

2 **PMIX\_COLLECT\_DATA** "pmix.collect" (bool)

3 Collect all data posted by the participants using **PMIx\_Put** that has been committed via  
4 **PMIx\_Commit**, making the collection locally available to each participant at the end of the  
5 operation. By default, this will include all job-level information that was locally generated  
6 by PMIx servers unless excluded using the **PMIX\_COLLECT\_GENERATED\_JOB\_INFO**  
7 attribute.

8 **PMIX\_COLLECT\_GENERATED\_JOB\_INFO** "pmix.collect.gen" (bool)

9 Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx  
10 servers. Some job-level information (e.g., distance between processes and fabric devices) is  
11 best determined on a distributed basis as it primarily pertains to local processes. Should  
12 remote processes need to access the information, it can either be obtained collectively using  
13 the **PMIx\_Fence** operation with this directive, or can be retrieved one peer at a time using  
14 **PMIx\_Get** without first having performed the job-wide collection.

## Optional Attributes

15 The following attributes are optional for PMIx implementations:

16 **PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)

17 All *clones* of the calling process must participate in the collective operation.

18 The following attributes are optional for host environments:

19 **PMIX\_TIMEOUT** "pmix.timeout" (int)

20 Time in seconds before the specified operation should time out (zero indicating infinite) and  
21 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
22 caused by multiple layers (client, server, and host) simultaneously timing the operation.

1            **Description**

2     Passing a **NULL** pointer as the *procs* parameter indicates that the fence is to span all processes in  
3     the client's namespace. Each provided **pmix\_proc\_t** struct can pass **PMIX\_RANK\_WILDCARD**  
4     to indicate that all processes in the given namespace are participating.

5     The *info* array is used to pass user directives regarding the behavior of the fence operation. Note  
6     that for scalability reasons, the default behavior for **PMIx\_Fence** is to not collect data posted by  
7     the operation's participants.

8            **Advice to PMIx library implementers**

9     **PMIx\_Fence** and its non-blocking form are both *collective* operations. Accordingly, the PMIx  
10    server library is required to aggregate participation by local clients, passing the request to the host  
   environment once all local participants have executed the API.

11           **Advice to PMIx server hosts**

12    The host will receive a single call for each collective operation. It is the responsibility of the host to  
13    identify the nodes containing participating processes, execute the collective across all participating  
   nodes, and notify the local PMIx server library upon completion of the global collective.

14        **5.2 PMIx\_Fence\_nb**

15           **Summary**

16    Execute a nonblocking **PMIx\_Fence** across the processes identified in the specified array of  
17    processes, collecting information posted via **PMIx\_Put** as directed.

1      **Format**

2      *PMIx v1.0*      C  
3      pmix\_status\_t  
4      PMIx\_Fence\_nb(const pmix\_proc\_t procs[], size\_t nprocs,  
5                    const pmix\_info\_t info[], size\_t ninfo,  
                  pmix\_op\_cbfunc\_t cbfunc, void \*cbdata);

6      **IN procs**  
7      Array of **pmix\_proc\_t** structures (array of handles)  
8      **IN nprocs**  
9      Number of elements in the *procs* array (integer)  
10     **IN info**  
11     Array of info structures (array of handles)  
12     **IN ninfo**  
13     Number of elements in the *info* array (integer)  
14     **IN cbfunc**  
15     Callback function (function reference)  
16     **IN cbdata**  
17     Data to be passed to the callback function (memory reference)

18     Returns one of the following:

- 19     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
20       will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
21       function prior to returning from the API.
- 22     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
23       returned *success* - the *cbfunc* will *not* be called. This can occur if the collective involved only  
24       processes on the local node.
- 25     • a PMIx error constant indicating either an error in the input or that the request was immediately  
26       processed and failed - the *cbfunc* will *not* be called.

27      **Required Attributes**

28     The following attributes are required to be supported by all PMIx libraries:

29     **PMIX\_COLLECT\_DATA** "pmix.collect" (bool)  
30       Collect all data posted by the participants using **PMIx\_Put** that has been committed via  
31       **PMIx\_Commit**, making the collection locally available to each participant at the end of the  
32       operation. By default, this will include all job-level information that was locally generated  
33       by PMIx servers unless excluded using the **PMIX\_COLLECT\_GENERATED\_JOB\_INFO**  
         attribute.

34     **PMIX\_COLLECT\_GENERATED\_JOB\_INFO** "pmix.collect.gen" (bool)

1      Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx  
2      servers. Some job-level information (e.g., distance between processes and fabric devices) is  
3      best determined on a distributed basis as it primarily pertains to local processes. Should  
4      remote processes need to access the information, it can either be obtained collectively using  
5      the **PMIx\_Fence** operation with this directive, or can be retrieved one peer at a time using  
6      **PMIx\_Get** without first having performed the job-wide collection.

## Optional Attributes

7      The following attributes are optional for PMIx implementations:

8      **PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)  
9      All *clones* of the calling process must participate in the collective operation.

10     The following attributes are optional for host environments that support this operation:

11     **PMIX\_TIMEOUT** "pmix.timeout" (int)  
12     Time in seconds before the specified operation should time out (zero indicating infinite) and  
13     return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
14     caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

15     Nonblocking version of the **PMIx\_Fence** routine. See the **PMIx\_Fence** description for further  
16     details.

### 5.2.1 Fence-related attributes

19     The following attributes are defined specifically to support the fence operation:

20     **PMIX\_COLLECT\_DATA** "pmix.collect" (bool)  
21     Collect all data posted by the participants using **PMIx\_Put** that has been committed via  
22     **PMIx\_Commit**, making the collection locally available to each participant at the end of the  
23     operation. By default, this will include all job-level information that was locally generated  
24     by PMIx servers unless excluded using the **PMIX\_COLLECT\_GENERATED\_JOB\_INFO**  
25     attribute.

26     **PMIX\_COLLECT\_GENERATED\_JOB\_INFO** "pmix.collect.gen" (bool)  
27     Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx  
28     servers. Some job-level information (e.g., distance between processes and fabric devices) is  
29     best determined on a distributed basis as it primarily pertains to local processes. Should  
30     remote processes need to access the information, it can either be obtained collectively using  
31     the **PMIx\_Fence** operation with this directive, or can be retrieved one peer at a time using  
32     **PMIx\_Get** without first having performed the job-wide collection.

33     **PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)  
34     All *clones* of the calling process must participate in the collective operation.

## 1 5.3 PMIx\_Get

### 2 Summary

3 Retrieve a key/value pair from the client's namespace.

### 4 Format

5 *PMIx v1.0*

C

```
6     pmix_status_t  
7     PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,  
8                 const pmix_info_t info[], size_t ninfo,  
9                 pmix_value_t **val);
```

C

#### 9 IN proc

10 Process identifier - a **NULL** value may be used in place of the caller's ID (handle)

#### 11 IN key

12 Key to retrieve (**pmix\_key\_t**)

#### 13 IN info

14 Array of info structures (array of handles)

#### 15 IN ninfo

16 Number of elements in the *info* array (integer)

#### 17 OUT val

18 value (handle)

19 Returns one of the following:

- 20 • **PMIX\_SUCCESS** The requested data has been returned.
- 21 • **PMIX\_ERR\_NOT\_FOUND** The requested data was not available.
- 22 • **PMIX\_ERR\_GET\_MALLOC\_REQD** Indicating that the returned value involves dynamically  
23 allocated memory instead of pointing to a static location as requested (only applies if  
24 **PMIX\_GET\_STATIC\_VALUES** was included in the request).
- 25 • a non-zero PMIx error constant indicating a reason for the request's failure.

### Required Attributes

26 The following attributes are required to be supported by all PMIx libraries:

27 **PMIX\_OPTIONAL "pmix.optional" (bool)**

28 Look only in the client's local data store for the requested value - do not request data from  
29 the PMIx server if not found.

30 **PMIX\_IMMEDIATE "pmix.immediate" (bool)**

31 Specified operation should immediately return an error from the PMIx server if the requested  
32 data cannot be found - do not request it from the host RM.

33 **PMIX\_DATA\_SCOPE "pmix.scope" (pmix\_scope\_t)**

```

1     Scope of the data to be searched in a PMIx_Get call.
2     PMIX_SESSION_INFO "pmix.ssn.info" (bool)
3         Return information regarding the session realm of the target process.
4     PMIX_JOB_INFO "pmix.job.info" (bool)
5         Return information regarding the job realm corresponding to the namespace in the target
6         process' identifier.
7     PMIX_APP_INFO "pmix.app.info" (bool)
8         Return information regarding the application realm to which the target process belongs - the
9         namespace of the target process serves to identify the job containing the target application. If
10        information about an application other than the one containing the target process is desired,
11        then the attribute array must contain a PMIX_APPNUM attribute identifying the desired
12        target application. This is useful in cases where there are multiple applications and the
13        mapping of processes to applications is unclear.
14     PMIX_NODE_INFO "pmix.node.info" (bool)
15        Return information from the node realm regarding the node upon which the specified
16        process is executing. If information about a node other than the one containing the specified
17        process is desired, then the attribute array must also contain either the PMIX_NODEID or
18        PMIX_HOSTNAME attribute identifying the desired target. This is useful for requesting
19        information about a specific node even if the identity of processes running on that node are
20        not known..
21     PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)
22        Request that any pointers in the returned value point directly to values in the key-value store.
23        The user must not release any returned data pointers. Note that a return status of
24        PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported - in
25        which case, the returned data contains allocated memory that the user must release.

```



## Optional Attributes



26 The following attributes are optional for host environments:

```

27     PMIX_TIMEOUT "pmix.timeout" (int)
28        Time in seconds before the specified operation should time out (zero indicating infinite) and
29        return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions
30        caused by multiple layers (client, server, and host) simultaneously timing the operation.

```



1    **Description**  
2    Retrieve information for the specified *key* associated with the process identified in the given  
3    **pmix\_proc\_t**, returning a pointer to the **pmix\_value\_t** containing the result in the given  
4    address. See Chapters 6 and 7 for details on rules governing retrieval of information.

5    This is a blocking operation - the caller will block until the retrieval rules of Chapters 6 or 7 are met.

6    The *info* array is used to pass user directives regarding the get operation.

7    **5.3.1 PMIx\_Get\_nb**

8    **Summary**

9    Nonblocking **PMIx\_Get** operation.

10   **Format**

PMIx v1.0

```
11   pmix_status_t
12   PMIx_Get_nb(const pmix_proc_t *proc, const char key[],
13                const pmix_info_t info[], size_t ninfo,
14                pmix_value_cbfunc_t cbfunc, void *cbdata);
```

15   **IN proc**

16   Process identifier - a **NULL** value may be used in place of the caller's ID (handle)

17   **IN key**

18   Key to retrieve (string)

19   **IN info**

20   Array of info structures (array of handles)

21   **IN ninfo**

22   Number of elements in the *info* array (integer)

23   **IN cbfunc**

24   Callback function (function reference)

25   **IN cbdata**

26   Data to be passed to the callback function (memory reference)

27   Returns one of the following:

- 28   • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
29   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
30   function prior to returning from the API.
- 31   • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
32   returned *success* - the *cbfunc* will *not* be called.
- 33   • a PMIx error constant indicating either an error in the input or that the request was immediately  
34   processed and failed - the *cbfunc* will *not* be called.

1 If executed, the status returned in the provided callback function will be one of the following  
2 constants:

- 3 • **PMIX\_SUCCESS** The requested data has been returned.  
4 • **PMIX\_ERR\_NOT\_FOUND** The requested data was not available.  
5 • **PMIX\_ERR\_GET\_MALLOC\_REQD** Indicating that the returned value involves dynamically  
6 allocated memory instead of pointing to a static location as requested (only applies if  
7 **PMIX\_GET\_STATIC\_VALUES** was included in the request).  
8 • a non-zero PMIx error constant indicating a reason for the request's failure.

9  **Required Attributes** 

10 The following attributes are required to be supported by all PMIx libraries:

11 **PMIX\_OPTIONAL "pmix.optional" (bool)**

12 Look only in the client's local data store for the requested value - do not request data from  
the PMIx server if not found.

13 **PMIX\_IMMEDIATE "pmix.immediate" (bool)**

14 Specified operation should immediately return an error from the PMIx server if the requested  
15 data cannot be found - do not request it from the host RM.

16 **PMIX\_DATA\_SCOPE "pmix.scope" (pmix\_scope\_t)**

17 Scope of the data to be searched in a **PMIx\_Get** call.

18 **PMIX\_SESSION\_INFO "pmix.ssn.info" (bool)**

19 Return information regarding the session realm of the target process.

20 **PMIX\_JOB\_INFO "pmix.job.info" (bool)**

21 Return information regarding the job realm corresponding to the namespace in the target  
22 process' identifier.

23 **PMIX\_APP\_INFO "pmix.app.info" (bool)**

24 Return information regarding the application realm to which the target process belongs - the  
25 namespace of the target process serves to identify the job containing the target application. If  
26 information about an application other than the one containing the target process is desired,  
27 then the attribute array must contain a **PMIX\_APPNUM** attribute identifying the desired  
28 target application. This is useful in cases where there are multiple applications and the  
29 mapping of processes to applications is unclear.

30 **PMIX\_NODE\_INFO "pmix.node.info" (bool)**

31 Return information from the node realm regarding the node upon which the specified  
32 process is executing. If information about a node other than the one containing the specified  
33 process is desired, then the attribute array must also contain either the **PMIX\_NODEID** or  
34 **PMIX\_HOSTNAME** attribute identifying the desired target. This is useful for requesting  
35 information about a specific node even if the identity of processes running on that node are  
36 not known..

```
1 PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)
2 Request that any pointers in the returned value point directly to values in the key-value store.
3 The user must not release any returned data pointers. Note that a return status of
4 PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported - in
5 which case, the returned data contains allocated memory that the user must release.
6 PMIX_GET_REFRESH_CACHE "pmix.get.refresh" (bool)
7 When retrieving data for a remote process, refresh the existing local data cache for the
8 process in case new values have been put and committed by the process since the last refresh.
9
```

---

10 The following attributes are required for host environments that support this operation:

```
11 PMIX_WAIT "pmix.wait" (int)
12 Caller requests that the PMIx server wait until at least the specified number of values are
13 found (a value of zero indicates all and is the default).
```

### Optional Attributes

15 The following attributes are optional for host environments that support this operation:

```
16 PMIX_TIMEOUT "pmix.timeout" (int)
17 Time in seconds before the specified operation should time out (zero indicating infinite) and
18 return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions
19 caused by multiple layers (client, server, and host) simultaneously timing the operation.
```

### Description

20 The callback function will be executed once the retrieval rules of Chapters 6 or 7 are met. See
21 **PMIx\_Get** for a full description.

## 5.3.2 Retrieval-specific constants

24 The following constants are defined for use by retrieval APIs:

```
25 PMIX_ERR_GET_MALLOC_REQD The data returned by PMIx_Get contains values that
26 include dynamic memory allocations (i.e., "malloc"), despite a request for static pointers to
27 the values in the key-value store. User is responsible for releasing the memory when done
28 with the information.
```

### 1    5.3.3 Retrieval attributes

2    The following attributes are defined for use by retrieval APIs:

3    **PMIX\_OPTIONAL "pmix.optional" (bool)**

4    Look only in the client's local data store for the requested value - do not request data from  
5    the PMIx server if not found.

6    **PMIX\_IMMEDIATE "pmix.immediate" (bool)**

7    Specified operation should immediately return an error from the PMIx server if the requested  
8    data cannot be found - do not request it from the host RM.

9    **PMIX\_GET\_STATIC\_VALUES "pmix.get.static" (bool)**

10   Request that any pointers in the returned value point directly to values in the key-value store.  
11   The user *must not* release any returned data pointers. Note that a return status of  
12   **PMIX\_ERR\_GET\_MALLOC\_REQD** indicates that direct pointers could not be supported - in  
13   which case, the returned data contains allocated memory that the user must release.

14   **PMIX\_GET\_REFRESH\_CACHE "pmix.get.refresh" (bool)**

15   When retrieving data for a remote process, refresh the existing local data cache for the  
16   process in case new values have been put and committed by the process since the last refresh.

17   **PMIX\_DATA\_SCOPE "pmix.scope" (pmix\_scope\_t)**

18   Scope of the data to be searched in a **PMIx\_Get** call.

19   **PMIX\_TIMEOUT "pmix.timeout" (int)**

20   Time in seconds before the specified operation should time out (zero indicating infinite) and  
21   return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
22   caused by multiple layers (client, server, and host) simultaneously timing the operation.

23   **PMIX\_WAIT "pmix.wait" (int)**

24   Caller requests that the PMIx server wait until at least the specified number of values are  
25   found (a value of zero indicates *all* and is the default).

## 26    5.4 Query

27   As the level of interaction between applications and the host SMS grows, so too does the need for  
28   the application to query the SMS regarding its capabilities and state information. PMIx provides a  
29   generalized query interface for this purpose, along with a set of standardized attribute keys to  
30   support a range of requests. This includes requests to determine the status of scheduling queues and  
31   active allocations, the scope of API and attribute support offered by the SMS, namespaces of active  
32   jobs, location and information about a job's processes, and information regarding available  
33   resources.

34   An example use-case for the **PMIx\_Query\_info\_nb** API is to ensure clean job completion.  
35   Time-shared systems frequently impose maximum run times when assigning jobs to resource  
36   allocations. To shut down gracefully (e.g., to write a checkpoint before termination) it is necessary  
37   for an application to periodically query the resource manager for the time remaining in its  
38   allocation. This is especially true on systems for which allocation times may be shortened or  
39   lengthened from the original time limit. Many resource managers provide APIs to dynamically  
40   obtain this information, but each API is specific to the resource manager.

1 PMIx supports this use-case by defining an attribute key (**PMIX\_TIME\_REMAINING**) that can be  
2 used with the **PMIx\_Query\_info\_nb** interface to obtain the number of seconds remaining in  
3 the current job allocation. Note that one could alternatively use the  
4 **PMIx\_Register\_event\_handler** API to register for an event indicating incipient job  
5 termination, and then use the **PMIx\_Job\_control\_nb** API to request that the host SMS  
6 generate an event a specified amount of time prior to reaching the maximum run time. PMIx  
7 provides such alternate methods as a means of maximizing the probability of a host system  
8 supporting at least one method by which the application can obtain the desired service.

9 The following APIs support query of various session and environment values.

## 10 5.4.1 **PMIx\_Resolve\_peers**

### 11 **Summary**

12 Obtain the array of processes within the specified namespace that are executing on a given node.

### 13 **Format**

14 *PMIx v1.0*

```
15 pmix_status_t
16 PMIx_Resolve_peers(const char *nodename,
17                     const pmix_nspace_t nspace,
18                     pmix_proc_t **procs, size_t *nprocs);
```

19 **IN nodename**

20 Name of the node to query - **NULL** can be used to denote the current local node (string)

21 **IN nspace**

22 namespace (string)

23 **OUT procs**

24 Array of process structures (array of handles)

25 **OUT nprocs**

26 Number of elements in the *procs* array (integer)

27 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

### 28 **Description**

29 Given a *nodename*, return the array of processes within the specified *nspace* that are executing on  
30 that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified  
31 node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will  
32 be zero. The caller is responsible for releasing the *procs* array when done with it. The  
**PMIX\_PROC\_FREE** macro is provided for this purpose.

## 33 5.4.2 **PMIx\_Resolve\_nodes**

### 34 **Summary**

35 Return a list of nodes hosting processes within the given namespace.

1           **Format**

2        `pmix_status_t`  
3        `PMIx_Resolve_nodes(const char *nspace, char **nodelist);`

4           **IN nspace**

5           Namespace (string)

6           **OUT nodelist**

7           Comma-delimited list of nodenames (string)

8           Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

9           **Description**

10          Given a *nspace*, return the list of nodes hosting processes within that namespace. The returned  
11          string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the  
12          string when done with it.

13 **5.4.3 PMIx\_Query\_info**

14           **Summary**

15          Query information about the system in general.

16           **Format**

17        `PMIx v4.0`

18        `pmix_status_t`  
19        `PMIx_Query_info(pmix_query_t queries[], size_t nqueries,`  
20                   `pmix_info_t *info[], size_t *ninfo);`

21           **IN queries**

22           Array of query structures (array of handles)

23           **IN nqueries**

24           Number of elements in the *queries* array (integer)

25           **INOUT info**

26           Address where a pointer to an array of `pmix_info_t` containing the results of the query can  
27           be returned (memory reference)

28           **INOUT ninfo**

29           Address where the number of elements in *info* can be returned (handle)

30           Returns one of the following:

- 31           • `PMIX_SUCCESS` All data was found and has been returned.
- 32           • `PMIX_ERR_NOT_FOUND` None of the requested data was available. The *info* array will be  
            `NULL` and *ninfo* zero.

- **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the requested data was found. The *info* array shall contain an element for each query key that returned a value.
- **PMIX\_ERR\_NOT\_SUPPORTED** The host RM does not support this function. The *info* array will be **NULL** and *ninfo* zero.
- a non-zero PMIx error constant indicating a reason for the request's failure. The *info* array will be **NULL** and *ninfo* zero.

## Required Attributes

PMIx libraries and host environments that support this API are required to support the following attributes:

**PMIX\_QUERY\_REFRESH\_CACHE "pmix.qry.rfsh" (bool)**  
10 Retrieve updated information from server. NO QUALIFIERS.

**PMIX\_SESSION\_INFO "pmix.ssn.info" (bool)**  
12 Return information regarding the session realm of the target process.

**PMIX\_JOB\_INFO "pmix.job.info" (bool)**  
14 Return information regarding the job realm corresponding to the namespace in the target  
15 process' identifier.

**PMIX\_APP\_INFO "pmix.app.info" (bool)**  
16 Return information regarding the application realm to which the target process belongs - the  
17 namespace of the target process serves to identify the job containing the target application. If  
18 information about an application other than the one containing the target process is desired,  
19 then the attribute array must contain a **PMIX\_APPNUM** attribute identifying the desired  
20 target application. This is useful in cases where there are multiple applications and the  
21 mapping of processes to applications is unclear.

**PMIX\_NODE\_INFO "pmix.node.info" (bool)**  
23 Return information from the node realm regarding the node upon which the specified  
24 process is executing. If information about a node other than the one containing the specified  
25 process is desired, then the attribute array must also contain either the **PMIX\_NODEID** or  
26 **PMIX\_HOSTNAME** attribute identifying the desired target. This is useful for requesting  
27 information about a specific node even if the identity of processes running on that node are  
28 not known..

**PMIX\_PROCID "pmix.proc\_id" (pmix\_proc\_t)**  
30 Process identifier. Used as a key in **PMIx\_Get** to retrieve the caller's own process identifier  
31 in a portion of the program that doesn't have access to the memory location in which it was  
32 originally stored (e.g., due to a call to **PMIx\_Init**). The process identifier in the  
33 **PMIx\_Get** call is ignored in this instance. In this context, specifies the process ID whose  
34 information is being requested - e.g., a query asking for the **pmix\_proc\_info\_t** of a  
35 specified process. Only required when the request is for information on a specific process.

**PMIX\_NSPACE "pmix.nspace" (char\*)**

1 Namespace of the job - may be a numerical value expressed as a string, but is often an  
2 alphanumeric string carrying information solely of use to the system. Specifies the  
3 namespace of the process whose information is being requested. Must be accompanied by  
4 the **PMIX\_RANK** attribute. Only required when the request is for information on a specific  
5 process.

6 **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)

7 Process rank within the job, starting from zero. Specifies the rank of the process whose  
8 information is being requested. Must be accompanied by the **PMIX\_NSPACE** attribute.  
9 Only required when the request is for information on a specific process.

10 **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** "pmix.qry.attrs" (bool)

11 Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or  
12 more of **PMIX\_CLIENT\_FUNCTIONS**, **PMIX\_SERVER\_FUNCTIONS**,  
13 **PMIX\_TOOL\_FUNCTIONS**, and **PMIX\_HOST\_FUNCTIONS**.

14 **PMIX\_CLIENT\_ATTRIBUTES** "pmix.client.attrs" (bool)

15 Request attributes supported by the PMIx client library.

16 **PMIX\_SERVER\_ATTRIBUTES** "pmix.srvr.attrs" (bool)

17 Request attributes supported by the PMIx server library.

18 **PMIX\_HOST\_ATTRIBUTES** "pmix.host.attrs" (bool)

19 Request attributes supported by the host environment.

20 **PMIX\_TOOL\_ATTRIBUTES** "pmix.setup.env" (bool)

21 Request attributes supported by the PMIx tool library functions.

22 Note that inclusion of both the **PMIX\_PROCID** directive and either the **PMIX\_NSPACE** or the  
23 **PMIX\_RANK** attribute will return a **PMIX\_ERR\_BAD\_PARAM** result, and that the inclusion of a  
24 process identifier must apply to all keys in that **pmix\_query\_t**. Queries for information on  
25 multiple specific processes therefore requires submitting multiple **pmix\_query\_t** structures,  
26 each referencing one process.

27 PMIx libraries are not required to directly support any other attributes for this function. However,  
28 all provided attributes must be passed to the host SMS daemon for processing. The PMIx library is  
29 *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process making  
30 the request.

## Optional Attributes

31 The following attributes are optional for host environments that support this operation:

32 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (char\*)

33 Request a comma-delimited list of active namespaces. NO QUALIFIERS.

34 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (pmix\_status\_t)

```

1      Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE
2      indicating the namespace whose status is being queried.

3      PMIX_QUERY_QUEUE_LIST "pmixqryqlst" (char*)
4          Request a comma-delimited list of scheduler queues. NO QUALIFIERS.

5      PMIX_QUERY_QUEUE_STATUS "pmixqryqst" (char*)
6          Returns status of a specified scheduler queue, expressed as a string. OPTIONAL
7          QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being
8          requested.

9      PMIX_QUERY_PROC_TABLE "pmixqryptable" (char*)
10         Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
11         process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
12         PMIX_NSPACE indicating the namespace whose process table is being queried.

13     PMIX_QUERY_LOCAL_PROC_TABLE "pmixqrylptable" (char*)
14         Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
15         process in the specified namespace executing on the same node as the requester, ordered by
16         process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace
17         whose local process table is being queried. OPTIONAL QUALIFIER: PMIX_HOSTNAME
18         indicating the host whose local process table is being queried. By default, the query assumes
19         that the host upon which the request was made is to be used.

20     PMIX_QUERY_SPAWN_SUPPORT "pmixqryspawn" (bool)
21         Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

22     PMIX_QUERY_DEBUG_SUPPORT "pmixqrydebug" (bool)
23         Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

24     PMIX_QUERY_MEMORY_USAGE "pmixqrymem" (bool)
25         Return information on memory usage for the processes indicated in the qualifiers.
26         OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
27         specific process(es) whose memory usage is being requested.

28     PMIX_QUERY_REPORT_AVG "pmixqryavg" (bool)
29         Report only average values for sampled information. NO QUALIFIERS.

30     PMIX_QUERY_REPORT_MINMAX "pmixqryminmax" (bool)
31         Report minimum and maximum values. NO QUALIFIERS.

32     PMIX_QUERY_ALLOC_STATUS "pmixqueryalloc" (char*)
33         String identifier of the allocation whose status is being requested. NO QUALIFIERS.

34     PMIX_TIME_REMAINING "pmixtimerenaining" (char*)
35         Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
36         OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
37         requested (defaults to allocation containing the caller).

```

```
1 PMIX_SERVER_URI "pmix.srvr.uri" (char*)
2     URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
3     PMIx connection. Defaults to requesting the information for the local PMIx server.
4 PMIX_PROC_URI "pmix.puri" (char*)
5     URI containing contact information for the specified process. Requests the URI of the
6     specified PMIx server's out-of-band connection. Defaults to requesting the information for
7     the local PMIx server.
```

## 8 **Description**

9 Query information about the system in general. This can include a list of active namespaces, fabric  
10 topology, etc. Also can be used to query node-specific info such as the list of peers executing on a  
11 given node. The host environment is responsible for exercising appropriate access control on the  
12 information.

13 The returned *status* indicates if requested data was found or not. The returned *info* array will  
14 contain a **PMIX\_QUERY\_RESULTS** element for each query of the *queries* array. If qualifiers were  
15 included in the query, then the first element of each results array shall contain the  
16 **PMIX\_QUERY\_QUALIFIERS** key with a **pmix\_data\_array\_t** containing the qualifiers. The  
17 remaining **pmix\_info\_t** shall contain the results of the query, one entry for each key that was  
18 found. Note that duplicate keys in the *queries* array shall result in duplicate responses within the  
19 constraints of the accompanying qualifiers. The caller is responsible for releasing the returned array.

### Advice to PMIx library implementers

20 Information returned from **PMIX\_Query\_info** shall be locally cached so that retrieval by  
21 subsequent calls to **PMIX\_Get**, **PMIX\_Query\_info**, or **PMIX\_Query\_info\_nb** can succeed  
22 with minimal overhead. The local cache shall be checked prior to querying the PMIx server and/or  
23 the host environment. Queries that include the **PMIX\_QUERY\_REFRESH\_CACHE** attribute shall  
24 bypass the local cache and retrieve a new value for the query, refreshing the values in the cache  
25 upon return.

## 26 **5.4.4 PMIX\_Query\_info\_nb**

### 27 **Summary**

28 Query information about the system in general.

1           **Format**

2        **pmix\_status\_t**  
3        **PMIx\_Query\_info\_nb**(**pmix\_query\_t** *queries*[], **size\_t** *nqueries*,  
4                            **pmix\_info\_cbfunc\_t** *cbfunc*, **void** \**cbdata*);

5        **IN    queries**  
6            Array of query structures (array of handles)  
7        **IN    nqueries**  
8            Number of elements in the *queries* array (integer)  
9        **IN    cbfunc**  
10          Callback function **pmix\_info\_cbfunc\_t** (function reference)  
11        **IN    cbdata**  
12          Data to be passed to the callback function (memory reference)

13      Returns one of the following:

- 14      • **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided  
15        callback function will be executed upon completion of the operation. Note that the library must  
16        not invoke the callback function prior to returning from the API.
- 17      • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this  
18        case, the provided callback function will not be executed.

19      If executed, the status returned in the provided callback function will be one of the following  
20        constants:

- 21      • **PMIX\_SUCCESS** All data was found and has been returned.  
22      • **PMIX\_ERR\_NOT\_FOUND** None of the requested data was available. The *info* array will be  
23        **NULL** and *ninfo* zero.  
24      • **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the requested data was found. The *info* array shall  
25        contain an element for each query key that returned a value.  
26      • **PMIX\_ERR\_NOT\_SUPPORTED** The host RM does not support this function. The *info* array will  
27        be **NULL** and *ninfo* zero.  
28      • a non-zero PMIx error constant indicating a reason for the request's failure. The *info* array will  
29        be **NULL** and *ninfo* zero.

30           **Required Attributes**

31      PMIx libraries and host environments that support this API are required to support the following  
32        attributes:

33      **PMIX\_QUERY\_REFRESH\_CACHE** "pmixqryrfsh" (**bool**)  
          Retrieve updated information from server. NO QUALIFIERS.

```

1   PMIX_SESSION_INFO "pmix.ssn.info" (bool)
2     Return information regarding the session realm of the target process.
3
4   PMIX_JOB_INFO "pmix.job.info" (bool)
5     Return information regarding the job realm corresponding to the namespace in the target
6     process' identifier.
7
8   PMIX_APP_INFO "pmix.app.info" (bool)
9     Return information regarding the application realm to which the target process belongs - the
10    namespace of the target process serves to identify the job containing the target application. If
11    information about an application other than the one containing the target process is desired,
12    then the attribute array must contain a PMIX_APPNUM attribute identifying the desired
13    target application. This is useful in cases where there are multiple applications and the
14    mapping of processes to applications is unclear.
15
16   PMIX_NODE_INFO "pmix.node.info" (bool)
17     Return information from the node realm regarding the node upon which the specified
18     process is executing. If information about a node other than the one containing the specified
19     process is desired, then the attribute array must also contain either the PMIX_NODEID or
20     PMIX_HOSTNAME attribute identifying the desired target. This is useful for requesting
21     information about a specific node even if the identity of processes running on that node are
22     not known..
23
24   PMIX_PROCID "pmix.proc_id" (pmix_proc_t)
25     Process identifier. Used as a key in PMIx_Get to retrieve the caller's own process identifier
26     in a portion of the program that doesn't have access to the memory location in which it was
27     originally stored (e.g., due to a call to PMIx_Init). The process identifier in the
28     PMIx_Get call is ignored in this instance. In this context, specifies the process ID whose
29     information is being requested - e.g., a query asking for the pmix_proc_info_t of a
30     specified process. Only required when the request is for information on a specific process.
31
32   PMIX_NSPACE "pmix.nspace" (char*)
33     Namespace of the job - may be a numerical value expressed as a string, but is often an
34     alphanumeric string carrying information solely of use to the system. Specifies the
35     namespace of the process whose information is being requested. Must be accompanied by
36     the PMIX_RANK attribute. Only required when the request is for information on a specific
37     process.
38
39   PMIX_RANK "pmix.rank" (pmix_rank_t)
40     Process rank within the job, starting from zero. Specifies the rank of the process whose
41     information is being requested. Must be accompanied by the PMIX_NSPACE attribute.
42     Only required when the request is for information on a specific process.
43
44   PMIX_QUERY_ATTRIBUTE_SUPPORT "pmixqry.attrs" (bool)
45     Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or
46     more of PMIX_CLIENT_FUNCTIONS, PMIX_SERVER_FUNCTIONS,
47     PMIX_TOOL_FUNCTIONS, and PMIX_HOST_FUNCTIONS.

```

```

1  PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
2      Request attributes supported by the PMIx client library.
3  PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
4      Request attributes supported by the PMIx server library.
5  PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
6      Request attributes supported by the host environment.
7  PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
8      Request attributes supported by the PMIx tool library functions.

9 Note that inclusion of both the PMIX_PROCID directive and either the PMIX_NSPACE or the
10 PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a
11 process identifier must apply to all keys in that pmix_query_t. Queries for information on
12 multiple specific processes therefore requires submitting multiple pmix_query_t structures,
13 each referencing one process.

14 PMIx libraries are not required to directly support any other attributes for this function. However,
15 all provided attributes must be passed to the host SMS daemon for processing. The PMIx library is
16 required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making
17 the request.

```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

```

19 PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*)
20     Request a comma-delimited list of active namespaces. NO QUALIFIERS.

21 PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t)
22     Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE
23     indicating the namespace whose status is being queried.

24 PMIX_QUERY_QUEUE_LIST "pmix.qry qlst" (char*)
25     Request a comma-delimited list of scheduler queues. NO QUALIFIERS.

26 PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (char*)
27     Returns status of a specified scheduler queue, expressed as a string. OPTIONAL
28     QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being
29     requested.

30 PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
31     Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
32     process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
33     PMIX_NSPACE indicating the namespace whose process table is being queried.

34 PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)

```

```

1 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
2 process in the specified namespace executing on the same node as the requester, ordered by
3 process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace
4 whose local process table is being queried. OPTIONAL QUALIFIER: PMIX_HOSTNAME
5 indicating the host whose local process table is being queried. By default, the query assumes
6 that the host upon which the request was made is to be used.

7 PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
8     Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

9 PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
10    Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

11 PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
12     Return information on memory usage for the processes indicated in the qualifiers.
13     OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
14     specific process(es) whose memory usage is being requested.

15 PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
16     Report only average values for sampled information. NO QUALIFIERS.

17 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
18     Report minimum and maximum values. NO QUALIFIERS.

19 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
20     String identifier of the allocation whose status is being requested. NO QUALIFIERS.

21 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
22     Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
23     OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
24     requested (defaults to allocation containing the caller).

25 PMIX_SERVER_URI "pmix.srvr.uri" (char*)
26     URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
27     PMIx connection. Defaults to requesting the information for the local PMIx server.

28 PMIX_PROC_URI "pmix.puri" (char*)
29     URI containing contact information for the specified process. Requests the URI of the
30     specified PMIx server's out-of-band connection. Defaults to requesting the information for
31     the local PMIx server.

```



## 32 **Description**

33 Non-blocking form of the `PMIx_Query_info` API.

## 5.4.5 Query-specific constants

**PMIX\_QUERY\_PARTIAL\_SUCCESS** Some, but not all, of the requested information was returned.

## 5.4.6 Query attributes

Attributes used to direct behavior of the **PMIx\_Query\_info** APIs.

**PMIX\_QUERY\_RESULTS** "pmix.qry.res" (**pmix\_data\_array\_t**)  
Contains an array of query results for a given **pmix\_query\_t** passed to the **PMIx\_Query\_info** APIs. If qualifiers were included in the query, then the first element of the array shall be the **PMIX\_QUERY\_QUALIFIERS** attribute containing those qualifiers. Each of the remaining elements of the array is a **pmix\_info\_t** containing the query key and the corresponding value returned by the query. This attribute is solely for reporting purposes and cannot be used in **PMIx\_Get** or other query operations.

**PMIX\_QUERY\_QUALIFIERS** "pmix.qry.quals" (**pmix\_data\_array\_t**)  
Contains an array of qualifiers that were included in the query that produced the provided results. This attribute is solely for reporting purposes and cannot be used in **PMIx\_Get** or other query operations.

**PMIX\_QUERY\_SUPPORTED\_KEYS** "pmix.qry.keys" (**char\***)  
Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS.

**PMIX\_QUERY\_SUPPORTED\_QUALIFIERS** "pmix.qry.quals" (**char\***)  
Return comma-delimited list of qualifiers supported by a query on the provided key, instead of actually performing the query on the key. NO QUALIFIERS.

**PMIX\_QUERY\_REFRESH\_CACHE** "pmix.qry.rfsh" (**bool**)  
Retrieve updated information from server. NO QUALIFIERS.

**PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (**char\***)  
Request a comma-delimited list of active namespaces. NO QUALIFIERS.

**PMIX\_QUERY\_NAMESPACE\_INFO** "pmix.qry.nsinfo" (**pmix\_data\_array\_t\***)  
Return an array of active namespace information - each element will itself contain an array including the namespace plus the command line of the application executing within it.  
OPTIONAL QUALIFIERS: **PMIX\_NSPACE** of specific namespace whose info is being requested

. **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (**pmix\_status\_t**)  
Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the namespace whose status is being queried.

**PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (**char\***)  
Request a comma-delimited list of scheduler queues. NO QUALIFIERS.

**PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (**char\***)  
Returns status of a specified scheduler queue, expressed as a string. OPTIONAL QUALIFIERS: **PMIX\_ALLOC\_QUEUE** naming specific queue whose status is being requested.

**PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.ptable" (**char\***)

```

1 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
2 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
3 PMIX_NSPACE indicating the namespace whose process table is being queried.
4 PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
5 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
6 process in the specified namespace executing on the same node as the requester, ordered by
7 process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace
8 whose local process table is being queried. OPTIONAL QUALIFIER: PMIX_HOSTNAME
9 indicating the host whose local process table is being queried. By default, the query assumes
10 that the host upon which the request was made is to be used.
11 PMIX_QUERY_AUTHORIZATIONS "pmix.qry.auths" (bool)
12 Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.
13 PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
14 Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.
15 PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
16 Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.
17 PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
18 Return information on memory usage for the processes indicated in the qualifiers.
19 OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
20 specific process(es) whose memory usage is being requested.
21 PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
22 Constrain the query to local information only. NO QUALIFIERS.
23 PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
24 Report only average values for sampled information. NO QUALIFIERS.
25 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
26 Report minimum and maximum values. NO QUALIFIERS.
27 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
28 String identifier of the allocation whose status is being requested. NO QUALIFIERS.
29 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
30 Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
31 OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
32 requested (defaults to allocation containing the caller).
33 PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
34 Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or
35 more of PMIX_CLIENT_FUNCTIONS, PMIX_SERVER_FUNCTIONS,
36 PMIX_TOOL_FUNCTIONS, and PMIX_HOST_FUNCTIONS.
37 PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
38 Return the number of process sets defined in the specified range (defaults to
39 PMIX_RANGE_SESSION).
40 PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)
41 Return a pmix_data_array_t containing an array of strings of the process set names
42 defined in the specified range (defaults to PMIX_RANGE_SESSION).

```

```

1   PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t)
2     Return an array of pmix_proc_t containing the members of the specified process set.
3
4   PMIX_QUERY_AVAIL_SERVERS "pmix.qry.asrvrs" (pmix_data_array_t)
5     Return an array of pmix_info_t, each element itself containing a
6     PMIX_SERVER_INFO_ARRAY entry holding all available data for a server on this node to
7     which the caller might be able to connect.
8   PMIX_SERVER_INFO_ARRAY "pmix.srv.arr" (pmix_data_array_t)
9     Array of pmix_info_t about a given server, starting with its PMIX_NSPACE and
      including at least one of the rendezvous-required pieces of information.

```

10 These attributes are used to query memory available and used in the system.

```

11   PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
12     Total available physical memory on a node. OPTIONAL QUALIFERS: PMIX_HOSTNAME
13     or PMIX_NODEID (defaults to caller's node).
14
15   PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float)
16     Megabytes of memory currently used by the RM daemon on the node. OPTIONAL
17     QUALIFERS: PMIX_HOSTNAME or PMIX_NODEID (defaults to caller's node).
18
19   PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float)
20     Average Megabytes of memory used by client processes on node. OPTIONAL
21     QUALIFERS: PMIX_HOSTNAME or PMIX_NODEID (defaults to caller's node).

```

22 The following attributes are used as qualifiers in queries regarding attribute support within the  
23 PMIx implementation and/or the host environment:

```

24   PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (bool)
25     Request a list of functions supported by the PMIx client library.
26   PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
27     Request attributes supported by the PMIx client library.
28   PMIX_SERVER_FUNCTIONS "pmix.srvr.fns" (bool)
29     Request a list of functions supported by the PMIx server library.
30   PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
31     Request attributes supported by the PMIx server library.
32   PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (bool)
33     Request a list of functions supported by the host environment.
34   PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
35     Request attributes supported by the host environment.
36   PMIX_TOOL_FUNCTIONS "pmix.tool.fns" (bool)
37     Request a list of functions supported by the PMIx tool library.
PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
      Request attributes supported by the PMIx tool library functions.

```

## 5.4.7 Query Structure

The `pmix_query_t` structure is used by the `PMIx_Query_info` APIs to describe a single query operation.

PMIx v2.0

```
4     typedef struct pmix_query {
5         char **keys;
6         pmix_info_t *qualifiers;
7         size_t nqual;
8     } pmix_query_t;
```

where:

- *keys* is a **NULL**-terminated argv-style array of strings
- *qualifiers* is an array of `pmix_info_t` describing constraints on the query
- *nqual* is the number of elements in the *qualifiers* array

### 5.4.7.1 Query structure support macros

The following macros are provided to support the `pmix_query_t` structure.

#### Initialize the query structure

Initialize the `pmix_query_t` fields

PMIx v2.0

```
17     PMIX_QUERY_CONSTRUCT (m)
```

IN m

Pointer to the structure to be initialized (pointer to `pmix_query_t`)

#### Destruct the query structure

Destruct the `pmix_query_t` fields

PMIx v2.0

```
22     PMIX_QUERY_DESTRUCT (m)
```

IN m

Pointer to the structure to be destructed (pointer to `pmix_query_t`)

```

1      Create a query array
2      Allocate and initialize an array of pmix_query_t structures
3      PMIx v2.0   ▼———— C —————▶
4          PMIX_QUERY_CREATE (m, n)   ▲———— C —————▶
5          INOUT m           Address where the pointer to the array of pmix_query_t structures shall be stored (handle)
6          IN n             Number of structures to be allocated (size_t)
7
8      Free a query structure
9      Release a pmix_query_t structure
10     PMIx v4.0   ▼———— C —————▶
11     PMIX_QUERY_RELEASE (m)   ▲———— C —————▶
12     IN m           Pointer to a pmix_query_t structure (handle)
13      Free a query array
14      Release an array of pmix_query_t structures
15     PMIx v2.0   ▼———— C —————▶
16     PMIX_QUERY_FREE (m, n)   ▲———— C —————▶
17     IN m           Pointer to the array of pmix_query_t structures (handle)
18     IN n           Number of structures in the array (size_t)
19
20     Create the info array of query qualifiers
21     Create an array of pmix_info_t structures for passing query qualifiers, updating the nqual field
22     of the pmix_query_t structure.
23     PMIx v2.2   ▼———— C —————▶
24     PMIX_QUERY_QUALIFIERS_CREATE (m, n)   ▲———— C —————▶
25     IN m           Pointer to the pmix_query_t structure (handle)
26     IN n           Number of qualifiers to be allocated (size_t)
27

```

## 1 5.5 Using Get vs Query

2 Both **PMIx\_Get** and **PMIx\_Query\_info** can be used to retrieve information about the system.  
3 In general, the *get* operation should be used to retrieve:

- 4 • information provided by the host environment at time of job start. This includes information on  
5 the number of processes in the job, their location, and possibly their communication endpoints.
- 6 • information posted by processes via the **PMIx\_Put** function.

7 This information is largely considered to be *static*, although this will not necessarily be true for  
8 environments supporting dynamic programming models or fault tolerance. Note that the  
9 **PMIx\_Get** function only accesses information about execution environments - i.e., its scope is  
10 limited to values pertaining to a specific *session*, *job*, *application*, *process*, or *node*. It cannot be  
11 used to obtain information about areas such as the status of queues in the WLM.

12 In contrast, the *query* option should be used to access:

- 13 • system-level information (such as the available WLM queues) that would generally not be  
14 included in job-level information provided at job start.
- 15 • dynamic information such as application and queue status, and resource utilization statistics.  
16 Note that the **PMIX\_QUERY\_REFRESH\_CACHE** attribute must be provided on each query to  
17 ensure current data is returned.
- 18 • information created post job start, such as process tables.
- 19 • information requiring more complex search criteria than supported by the simpler **PMIx\_Get**  
20 API.
- 21 • queries focused on retrieving multi-attribute blocks of data with a single request, thus bypassing  
22 the single-key limitation of the **PMIx\_Get** API.

23 In theory, all information can be accessed via **PMIx\_Query\_info** as the local cache is typically  
24 the same datastore searched by **PMIx\_Get**. However, in practice, the overhead associated with the  
25 *query* operation may (depending upon implementation) be higher than the simpler *get* operation  
26 due to the need to construct and process the more complex **pmix\_query\_t** structure. Thus,  
27 requests for a single key value are likely to be accomplished faster with **PMIx\_Get** versus the  
28 *query* operation.

## 29 5.6 Accessing attribute support information

30 Information as to which attributes are supported by either the PMIx implementation or its host  
31 environment can be obtained via the **PMIx\_Query\_info** APIs. The  
32 **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** attribute must be listed as the first entry in the *keys* field  
33 of the **pmix\_query\_t** structure, followed by the name of the function whose attribute support is  
34 being requested - support for multiple functions can be requested simultaneously by simply adding

1 the function names to the array of *keys*. Function names *must* be given as user-level API names -  
2 e.g., “PMIx\_Get”, “PMIx\_server\_setup\_application”, or “PMIx\_tool\_connect\_to\_server”.

3 The desired levels of attribute support are provided as qualifiers. Multiple levels can be requested  
4 simultaneously by simply adding elements to the *qualifiers* array. Each qualifier should contain the  
5 desired level attribute with the boolean value set to indicate whether or not that level is to be  
6 included in the returned information. Failure to provide any levels is equivalent to a request for all  
7 levels. Supported levels include:

- 8 • **PMIX\_CLIENT\_FUNCTIONS** "pmix.client.fns" (bool)  
9 Request a list of functions supported by the PMIx client library.
- 10 • **PMIX\_CLIENT\_ATTRIBUTES** "pmix.client.attrs" (bool)  
11 Request attributes supported by the PMIx client library.
- 12 • **PMIX\_SERVER\_FUNCTIONS** "pmix.srvr.fns" (bool)  
13 Request a list of functions supported by the PMIx server library.
- 14 • **PMIX\_SERVER\_ATTRIBUTES** "pmix.srvr.attrs" (bool)  
15 Request attributes supported by the PMIx server library.
- 16 • **PMIX\_HOST\_FUNCTIONS** "pmix.srvr.fns" (bool)  
17 Request a list of functions supported by the host environment.
- 18 • **PMIX\_HOST\_ATTRIBUTES** "pmix.host.attrs" (bool)  
19 Request attributes supported by the host environment.
- 20 • **PMIX\_TOOL\_FUNCTIONS** "pmix.tool.fns" (bool)  
21 Request a list of functions supported by the PMIx tool library.
- 22 • **PMIX\_TOOL\_ATTRIBUTES** "pmix.setup.env" (bool)  
23 Request attributes supported by the PMIx tool library functions.

24 Unlike other queries, queries for attribute support can result in the number of returned  
25 **pmix\_info\_t** structures being different from the number of queries. Each element in the  
26 returned array will correspond to a pair of specified attribute level and function in the query, where  
27 the *key* is the function and the *value* contains a **pmix\_data\_array\_t** of **pmix\_info\_t**. Each  
28 element of the array is marked by a *key* indicating the requested attribute *level* with a *value*  
29 composed of a **pmix\_data\_array\_t** of **pmix\_regattr\_t**, each describing a supported  
30 attribute for that function, as illustrated in Fig. 5.1 below where the requestor asked for supported  
31 attributes of **PMIx\_Get** at the *client* and *server* levels, plus attributes of  
32 **PMIx\_Allocation\_request** at all levels.

33 The array of returned structures, and their child arrays, are subject to the return rules for the  
34 **PMIx\_Query\_info\_nb** API. For example, a request for supported attributes of the **PMIx\_Get**  
35 function that includes the *host* level will return values for the *client* and *server* levels, plus an array  
36 element with a *key* of **PMIX\_HOST\_ATTRIBUTES** and a value type of **PMIX\_UNDEF** indicating  
37 that no attributes are supported at that level.

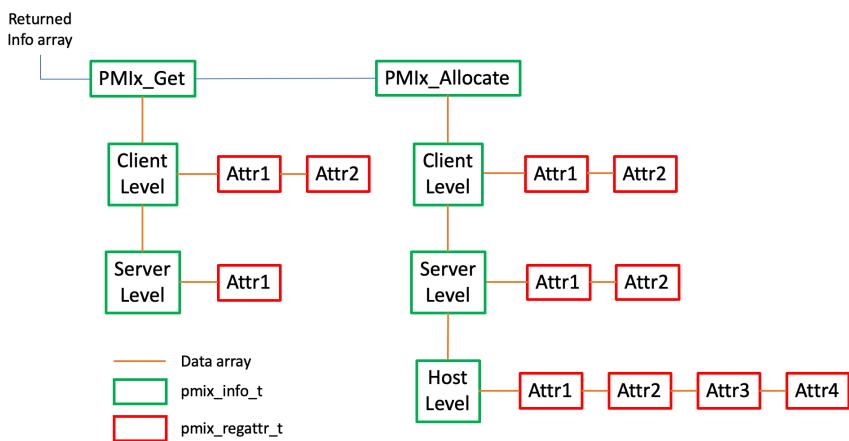


Figure 5.1.: Returned information hierarchy for attribute support request

## CHAPTER 6

# Reserved Keys

---

1     *Reserved* keys are keys whose string representation begin with a prefix of "**pmix**". By definition,  
2     reserved keys are provided by the host environment and the PMIx server, and are required to be  
3     available at client start of execution. PMIx clients and tools are therefore prohibited from posting  
4     reserved keys using the **PMIx\_Put** API.

5     PMIx implementations may choose to define their own custom-prefixed keys which may adhere to  
6     either the *reserved* or the *non-reserved* retrieval rules at the discretion of the implementation.  
7     Implementations may choose to provide such custom keys at client start of execution, but this is not  
8     required.

9     Host environments may also opt to define their own custom keys. However, PMIx implementations  
10    are unlikely to recognize such host-defined keys and will therefore treat them according to the  
11    *non-reserved* rules described in Chapter 7. Users are advised to check both the local PMIx  
12    implementation and host environment documentation for a list of any custom prefixes they must  
13    avoid, and to learn of any non-standard keys that may require special handling.

## 6.1 Data realms

15    PMIx information spans a wide range of sources. In some cases, there are multiple overlapping  
16    sources for the same type of data - e.g., the session, job, and application can each provide  
17    information on the number of nodes involved in their respective area. In order to resolve the  
18    ambiguity, a *data realm* is used to identify the scope to which the referenced data applies. Thus, a  
19    reference to an attribute that isn't specific to a realm (e.g., the **PMIX\_NUM\_NODES** attribute) must  
20    be accompanied by a corresponding attribute identifying the realm to which the request pertains if  
21    it differs from the default.

22    PMIx defines five *data realms* to resolve the ambiguities, as captured in the following attributes  
23    used in **PMIx\_Get** for retrieving information from each of the realms:

24    **PMIX\_SESSION\_INFO "pmix.ssn.info" (bool)**

25       Return information regarding the session realm of the target process.

26    **PMIX\_JOB\_INFO "pmix.job.info" (bool)**

27       Return information regarding the job realm corresponding to the namespace in the target  
28       process' identifier.

29    **PMIX\_APP\_INFO "pmix.app.info" (bool)**

1       Return information regarding the application realm to which the target process belongs - the  
2       namespace of the target process serves to identify the job containing the target application. If  
3       information about an application other than the one containing the target process is desired,  
4       then the attribute array must contain a **PMIX\_APPNUM** attribute identifying the desired  
5       target application. This is useful in cases where there are multiple applications and the  
6       mapping of processes to applications is unclear.  
7       **PMIX\_PROC\_INFO "pmix.proc.info" (bool)**

8       Return information regarding the target process. This attribute is technically not required as  
9       the **PMIx\_Get** API specifically identifies the target process in its parameters. However, it is  
10      included here for completeness.

11      **PMIX\_NODE\_INFO "pmix.node.info" (bool)**

12      Return information from the node realm regarding the node upon which the specified  
13      process is executing. If information about a node other than the one containing the specified  
14      process is desired, then the attribute array must also contain either the **PMIX\_NODEID** or  
15      **PMIX\_HOSTNAME** attribute identifying the desired target. This is useful for requesting  
16      information about a specific node even if the identity of processes running on that node are  
17      not known..

---

### Advice to users

---

18      If information about a session other than the one containing the requesting process is desired, then  
19      the attribute array must contain a **PMIX\_SESSION\_ID** attribute identifying the desired target  
20      session. This is required as many environments only guarantee unique namespaces within a  
21      session, and not across sessions.

22      The PMIx server has corresponding attributes the host can use to specify the realm of information  
23      that it provides during namespace registration (see Section 16.2.3.2).

## 24     6.1.1 Session realm attributes

25      If information about a session other than the one containing the requesting process is desired, then  
26      the *info* array passed to **PMIx\_Get** must contain a **PMIX\_SESSION\_ID** attribute identifying the the  
27      desired target session. This is required as many environments only guarantee unique namespaces  
28      within a session, and not across sessions.

29      Note that the *proc* argument of **PMIx\_Get** is ignored when referencing session-related  
30      information.

31      Session-level information includes the following attributes:

32      **PMIX\_SESSION\_ID "pmix.session.id" (uint32\_t)**  
33            Session identifier assigned by the scheduler.  
34      **PMIX\_CLUSTER\_ID "pmix.clid" (char\*)**  
35            A string name for the cluster this allocation is on.  
36      **PMIX\_UNIV\_SIZE "pmix.univ.size" (uint32\_t)**

```

1 Number of allocated slots in a session - each slot may or may not be occupied by an
2 executing process. Note that this attribute is equivalent to the PMIX_MAX_PROCS attributed
3 combined with PMIX_SESSION_INFO array - it is included in the PMIx Standard for
4 historical reasons.
5 PMIX_TMPDIR "pmix.tmpdir" (char*)
6 Full path to the top-level temporary directory assigned to the session.
7 PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)
8 Resource Manager will cleanup assigned temporary directory trees.

```

9 The following attributes are used to describe the RM - these are values assigned by the host  
10 environment to the session:

```

11 PMIX_RM_NAME "pmix.rm.name" (char*)
12 String name of the RM.
13 PMIX_RM_VERSION "pmix.rm.version" (char*)
14 RM version string.

```

15 The remaining session-related information can only be retrieved by including the  
16 **PMIX\_SESSION\_INFO** attribute in the *info* array passed to **PMIx\_Get**:

```

17 PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)
18 Comma-delimited list or regular expression of all nodes in the specified realm regardless of
19 whether or not they currently host processes.
20 PMIX_NUM_ALLOCATED_NODES "pmix.num.anodes" (uint32_t)
21 Number of nodes in the specified realm regardless of whether or not they currently host
22 processes.
23 PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
24 Maximum number of processes that can be executed in the specified realm. Typically, this is
25 a constraint imposed by a scheduler or by user settings in a hostfile or other resource
26 description.
27 PMIX_NODE_LIST "pmix.nlist" (char*)
28 Comma-delimited list of nodes currently hosting processes in the specified realm.
29 PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)
30 Number of slots allocated in the specified realm. Note that this attribute is the equivalent to
31 PMIX_MAX_PROCS - it is included in the PMIx Standard for historical reasons.
32 PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)
33 Number of nodes currently hosting processes in the specified realm.
34 PMIX_NODE_MAP "pmix.nmap" (char*)
35 Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2
36 for an explanation of its generation.
37 PMIX_PROC_MAP "pmix.pmap" (char*)
38 Regular expression describing processes on each node in the specified realm - see 16.2.3.2
39 for an explanation of its generation.
40 PMIX_ANL_MAP "pmix.anlmap" (char*)

```

2 

## 6.1.2 Job realm attributes

3 Job-related information is retrieved by including the namespace of the target job and a rank of  
 4 **PMIX\_RANK\_WILDCARD** in the *proc* argument passed to **PMIx\_Get**. If desired for code clarity,  
 5 the caller can also include the **PMIX\_JOB\_INFO** attribute in the *info* array, though this is not  
 6 required. If information is requested about a namespace in a session other than the one containing  
 7 the requesting process, then the *info* array must contain a **PMIX\_SESSION\_ID** attribute  
 8 identifying the desired target session. This is required as many environments only guarantee unique  
 9 namespaces within a session, and not across sessions.

10 Job-level information includes the following attributes:

11 **PMIX\_NSPACE "pmix.nspace" (char\*)**

12 Namespace of the job - may be a numerical value expressed as a string, but is often an  
 13 alphanumeric string carrying information solely of use to the system.

14 **PMIX\_JOBID "pmix.jobid" (char\*)**

15 Job identifier assigned by the scheduler to the specified job - may be identical to the  
 16 namespace, but is often a numerical value expressed as a string (e.g., "12345.3").

17 **PMIX\_NPROC\_OFFSET "pmix.offset" (pmix\_rank\_t)**

18 Starting global rank of the specified job.

19 **PMIX\_MAX\_PROCS "pmix.max.size" (uint32\_t)**

20 Maximum number of processes that can be executed in the specified realm. Typically, this is  
 21 a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
 22 description. In this context, this is the maximum number of processes that can be executed  
 23 in the specified job, which may be a subset of the number allocated to the overall session.

24 **PMIX\_NUM\_SLOTS "pmix.num.slots" (uint32\_t)**

25 Number of slots allocated in the specified realm. Note that this attribute is the equivalent to  
 26 **PMIX\_MAX\_PROCS** - it is included in the PMIx Standard for historical reasons. In this  
 27 context, this is the number of slots assigned to the specified job, which may be a subset of  
 28 the slots allocated to the overall session. Jobs may reserve a subset of their assigned slots for  
 29 dynamic operations such as **PMIx\_Spawn** - i.e., not all slots may be occupied by executing  
 30 processes from this job at a given point in time.

31 **PMIX\_NUM\_NODES "pmix.num.nodes" (uint32\_t)**

32 Number of nodes currently hosting processes in the specified realm. In this context, this is  
 33 the number of nodes currently hosting processes in the specified job, which may be a subset  
 34 of the nodes allocated to the overall session. Jobs may reserve a subset of their assigned  
 35 nodes for dynamic operations such as **PMIx\_Spawn** - i.e., not all nodes may have executing  
 36 processes from this job at a given point in time.

37 **PMIX\_NODE\_MAP "pmix.nmap" (char\*)**

38 Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2  
 39 for an explanation of its generation. In this context, this is the regular expression of nodes  
 40 currently hosting processes in the specified job.

```

1   PMIX_NODE_LIST "pmix.nlist" (char*)
2     Comma-delimited list of nodes currently hosting processes in the specified realm. In this
3     context, this is the comma-delimited list of nodes currently hosting processes in the specified
4     job.
5   PMIX_PROC_MAP "pmix.pmap" (char*)
6     Regular expression describing processes on each node in the specified realm - see 16.2.3.2
7     for an explanation of its generation. In this context, this is the regular expression describing
8     processes on each node in the specified job.
9   PMIX_ANL_MAP "pmix.anlmap" (char*)
10    Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation. In this context,
11    this is the process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation of the
12    processes in the specified job.
13   PMIX_CMD_LINE "pmix.cmd.line" (char*)
14    Command line used to execute the specified job (e.g., "mpirun -n 2 --map-by foo ./myapp : -n
15      4 ./myapp2").
16   PMIX_NSDIR "pmix.nsdir" (char*)
17    Full path to the temporary directory assigned to the specified job, under PMIX_TMPDIR.
18   PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)
19    Lowest rank within the specified job on the node (defaults to current node in absence of
20      PMIX_HOSTNAME or PMIX_NODEID qualifier).
21   PMIX_JOB_SIZE "pmix.job.size" (uint32_t)
22    Total number of processes in the specified job across all contained applications. Note that
23    this value can be different from PMIX_MAX_PROCS. For example, users may choose to
24    subdivide an allocation (running several jobs in parallel within it), and dynamic
25    programming models may support adding and removing processes from a running job
26    on-the-fly. In the latter case, PMIx events may be used to notify processes within the job that
27    the job size has changed.
28   PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)
29    Number of applications in the specified job.

```

### 30 6.1.3 Application realm attributes

31 Application-related information can only be retrieved by including the **PMIX\_APP\_INFO** attribute  
32 in the *info* array passed to **PMIx\_Get**. If the **PMIX\_APPNUM** qualifier is given, then the query  
33 shall return the corresponding value for the given application within the namespace specified in the  
34 *proc* argument of the query (a **NULL** value for the *proc* argument equates to the namespace of the  
35 caller). If the **PMIX\_APPNUM** qualifier is not included, then the retrieval shall default to the  
36 application containing the specified process. If the rank of the specified process is  
37 **PMIX\_RANK\_WILDCARD**, then the application number shall default to that of the calling process  
38 if the namespace is its own job, or a value of zero if the namespace is that of a different job.

39 Application-level information includes the following attributes:

40 **PMIX\_APPNUM** "pmix.appnum" (**uint32\_t**)

```

1      The application number within the job in which the specified process is a member.
2      PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)
3          Number of nodes currently hosting processes in the specified realm. In this context, this is
4          the number of nodes currently hosting processes in the specified application, which may be a
5          subset of the nodes allocated to the overall session.
6      PMIX_APPLDR "pmix.aldr" (pmix_rank_t)
7          Lowest rank in the specified application.
8      PMIX_APP_SIZE "pmix.app.size" (uint32_t)
9          Number of processes in the specified application, regardless of their execution state - i.e.,
10         this number may include processes that either failed to start or have already terminated.
11      PMIX_APP_ARGV "pmix.app.argv" (char*)
12         Consolidated argv passed to the spawn command for the given application (e.g., "./myapp
13         arg1 arg2 arg3").
14      PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
15         Maximum number of processes that can be executed in the specified realm. Typically, this is
16         a constraint imposed by a scheduler or by user settings in a hostfile or other resource
17         description. In this context, this is the maximum number of processes that can be executed
18         in the specified application, which may be a subset of the number allocated to the overall
19         session and job.
20      PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)
21         Number of slots allocated in the specified realm. Note that this attribute is the equivalent to
22         PMIX_MAX_PROCS - it is included in the PMIx Standard for historical reasons. In this
23         context, this is the number of slots assigned to the specified application, which may be a
24         subset of the slots allocated to the overall session and job.
25      PMIX_NODE_MAP "pmix.nmap" (char*)
26         Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2
27         for an explanation of its generation. In this context, this is the regular expression of nodes
28         currently hosting processes in the specified application.
29      PMIX_NODE_LIST "pmix.nlist" (char*)
30         Comma-delimited list of nodes currently hosting processes in the specified realm. In this
31         context, this is the comma-delimited list of nodes currently hosting processes in the specified
32         application.
33      PMIX_PROC_MAP "pmix.pmap" (char*)
34         Regular expression describing processes on each node in the specified realm - see 16.2.3.2
35         for an explanation of its generation. In this context, this is the regular expression describing
36         processes on each node in the specified application.
37      PMIX_APP_MAP_TYPE "pmix.apmap.type" (char*)
38         Type of mapping used to layout the application (e.g., cyclic).
39      PMIX_APP_MAP_REGEX "pmix.apmap.regex" (char*)
40         Regular expression describing the result of the process mapping.

```

## 1    6.1.4 Process realm attributes

2    Process-related information is retrieved by referencing the namespace and rank of the target process  
3    in the call to **PMIx\_Get**. If information is requested about a process in a session other than the one  
4    containing the requesting process, then an attribute identifying the target session must be provided.  
5    This is required as many environments only guarantee unique namespaces within a session, and not  
6    across sessions.

7    Process-level information includes the following attributes:

8    **PMIX\_APPNUM "pmix.appnum" (uint32\_t)**

9         The application number within the job in which the specified process is a member.

10      **PMIX\_RANK "pmix.rank" (pmix\_rank\_t)**

11         Process rank within the job, starting from zero.

12      **PMIX\_GLOBAL\_RANK "pmix.grank" (pmix\_rank\_t)**

13         Rank of the specified process spanning across all jobs in this session, starting with zero.

14         Note that no ordering of the jobs is implied when computing this value. As jobs can start and  
15         end at random times, this is defined as a continually growing number - i.e., it is not  
16         dynamically adjusted as individual jobs and processes are started or terminated.

17      **PMIX\_APP\_RANK "pmix.apprank" (pmix\_rank\_t)**

18         Rank of the specified process within its application.

19      **PMIX\_PARENT\_ID "pmix.parent" (pmix\_proc\_t)**

20         Process identifier of the parent process of the specified process - typically used to identify  
21         the application process that caused the job containing the specified process to be spawned  
22         (e.g., the process that called **PMIx\_Spawn**).

23      **PMIX\_EXIT\_CODE "pmix.exit.code" (int)**

24         Exit code returned when the specified process terminated.

25      **PMIX\_PROCID "pmix.procid" (pmix\_proc\_t)**

26         Process identifier. Used as a key in **PMIx\_Get** to retrieve the caller's own process identifier  
27         in a portion of the program that doesn't have access to the memory location in which it was  
28         originally stored (e.g., due to a call to **PMIx\_Init**). The process identifier in the  
29         **PMIx\_Get** call is ignored in this instance.

30      **PMIX\_LOCAL\_RANK "pmix.lrank" (uint16\_t)**

31         Rank of the specified process on its node - refers to the numerical location (starting from  
32         zero) of the process on its node when counting only those processes from the same job that  
33         share the node, ordered by their overall rank within that job.

34      **PMIX\_NODE\_RANK "pmix.nrank" (uint16\_t)**

35         Rank of the specified process on its node spanning all jobs- refers to the numerical location  
36         (starting from zero) of the process on its node when counting all processes (regardless of  
37         job) that share the node, ordered by their overall rank within the job. The value represents a  
38         snapshot in time when the specified process was started on its node and is not dynamically  
39         adjusted as processes from other jobs are started or terminated on the node.

40      **PMIX\_PACKAGE\_RANK "pmix.pkgrank" (uint16\_t)**

1 Rank of the specified process on the *package* where this process resides - refers to the  
 2 numerical location (starting from zero) of the process on its package when counting only  
 3 those processes from the same job that share the package, ordered by their overall rank  
 4 within that job. Note that processes that are not bound to PUs within a single specific  
 5 package cannot have a package rank.  
 6 **PMIX\_PROC\_PID** "pmix.pid" (**pid\_t**)  
 7 Operating system PID of specified process.  
 8 **PMIX\_PROCDIR** "pmix.pdir" (**char\***)  
 9 Full path to the subdirectory under **PMIX\_NSDIR** assigned to the specified process.  
 10 **PMIX\_CPUSET** "pmix.cpuset" (**char\***)  
 11 A string representation of the PU binding bitmap applied to the process upon launch. The  
 12 string shall begin with the name of the library that generated it (e.g., "hwloc") followed by a  
 13 colon and the bitmap string itself.  
 14 **PMIX\_CREDENTIAL** "pmix.cred" (**char\***)  
 15 Security credential assigned to the process.  
 16 **PMIX\_SPAWNED** "pmix.spawned" (**bool**)  
 17 **true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return  
 18 status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.  
 19 **PMIX\_REINCARNATION** "pmix.reinc" (**uint32\_t**)  
 20 Number of times this process has been re-instantiated - i.e, a value of zero indicates that the  
 21 process has never been restarted.

22 In addition, process-level information includes functional attributes directly associated with a  
 23 process - for example, the process-related fabric attributes included in Section 14.3.

## 24 6.1.5 Node realm keys

25 Information regarding the local node can be retrieved by directly requesting the node realm key in  
 26 the call to **PMIx\_Get** - the keys for node-related information are not shared across other realms.  
 27 The target process identifier will be ignored for keys that are not dependent upon it. Information  
 28 about a node other than the local node can be retrieved by specifying the **PMIX\_NODE\_INFO**  
 29 attribute in the *info* array along with either the **PMIX\_HOSTNAME** or **PMIX\_NODEID** qualifiers for  
 30 the node of interest.

31 Node-level information includes the following keys:

32 **PMIX\_HOSTNAME** "pmix.hname" (**char\***)  
 33 Name of the host, as returned by the **gethostname** utility or its equivalent.  
 34 **PMIX\_HOSTNAME\_ALIASES** "pmix.alias" (**char\***)  
 35 Comma-delimited list of names by which the target node is known.  
 36 **PMIX\_HOSTNAME\_KEEP\_FQDN** "pmix.fqdn" (**bool**)  
 37 Fully Qualified Domain Names (FQDNs) are being retained by the PMIx library.  
 38 **PMIX\_NODEID** "pmix.nodeid" (**uint32\_t**)

```

1 Node identifier expressed as the node's index (beginning at zero) in an array of nodes within
2 the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME
3 of the node - i.e., users can interchangeably reference the same location using either the
4 PMIX_HOSTNAME or corresponding PMIX_NODEID.
5 PMIX_NODE_SIZE "pmix.node.size" (uint32_t)
6 Number of processes across all jobs that are executing upon the node.
7 PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
8 Total available physical memory on a node.

9 The following attributes only return information regarding the caller's node - any node-related
10 qualifiers shall be ignored. In addition, these attributes require specification of the namespace in the
11 target process identifier except where noted - the value of the rank is ignored in all cases.
12 PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
13 Comma-delimited list of ranks that are executing on the local node within the specified
14 namespace – shortcut for PMIxResolve_peers for the local node.
15 PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)
16 Array of pmix_proc_t of all processes executing on the local node – shortcut for
17 PMIxResolve_peers for the local node and a NULL namespace argument. The process
18 identifier is ignored for this attribute.
19 PMIX_LOCAL_CPUSETS "pmix.lcpus" (pmix_data_array_t)
20 A pmix_data_array_t array of string representations of the PU binding bitmaps
21 applied to each local peer on the caller's node upon launch. Each string shall begin with the
22 name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap string
23 itself. The array shall be in the same order as the processes returned by
24 PMIX_LOCAL_PEERS for that namespace.
25 PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)
26 Number of processes in the specified job or application realm on the caller's node. Defaults
27 to job realm unless the PMIX_APP_INFO and the PMIX_APPNUM qualifiers are given.

28 In addition, node-level information includes functional attributes directly associated with a node -
29 for example, the node-related fabric attributes included in Section 14.3.

```

## 6.2 Retrieval rules for reserved keys

```

31 The retrieval rules for reserved keys are relatively simple as the keys are required, by definition, to
32 be available when the client begins execution. Accordingly, PMIxGet for a reserved key first
33 checks the local PMIx Client cache (per the data realm rules of the prior section) for the target key.
34 If the information is not found, then the PMIX_ERR_NOT_FOUND error constant is returned unless
35 the target process belongs to a different namespace from that of the requester.

```

```

36 In the case where the target and requester's namespaces differ, then the request is forwarded to the
37 local PMIx server. Upon receiving the request, the server shall check its data storage for the

```

1 specified namespace. If it already knows about this namespace, then it shall attempt to lookup the  
2 specified key, returning the value if it is found or the **PMIX\_ERR\_NOT\_FOUND** error constant.

3 If the server does not have a copy of the information for the specified namespace, then the server  
4 shall take one of the following actions:

- 5 1. If the request included the **PMIX\_IMMEDIATE** attribute, then the server will respond to the  
6 client with the **PMIX\_ERR\_NOT\_FOUND** status.
- 7 2. If the host has provided the Direct Business Card Exchange (DBCX) module function interface  
8 (**pmix\_server\_dmodex\_req\_fn\_t**), then the server shall pass the request to its host for  
9 servicing. The host is responsible for identifying a source of information on the specified  
10 namespace and retrieving it. The host is required to retrieve *all* of the information regarding the  
11 target namespace and return it to the requesting server in anticipation of follow-on requests. If  
12 the host cannot retrieve the namespace information, then it must respond with the  
13 **PMIX\_ERR\_NOT\_FOUND** error constant unless the **PMIX\_TIMEOUT** is given and reached (in  
14 which case, the host must respond with the **PMIX\_ERR\_TIMEOUT** constant).

15 Once the the PMIx server receives the namespace information, the server shall search it (again  
16 adhering to the prior data realm rules) for the requested key, returning the value if it is found or  
17 the **PMIX\_ERR\_NOT\_FOUND** error constant.

- 18 3. If the host does not support the DBCX interface, then the server will respond to the client with  
19 the **PMIX\_ERR\_NOT\_FOUND** status

## 20 **6.2.1 Accessing information: examples**

21 This section provides examples illustrating methods for accessing information from the various  
22 realms. The intent of the examples is not to provide comprehensive coding guidance, but rather to  
23 further illustrate the use of **PMIx\_Get** for obtaining information on a *session, job, application,*  
24 *process*, and *node*.

### 25 **6.2.1.1 Session-level information**

26 The **PMIx\_Get** API does not include an argument for specifying the *session* associated with the  
27 information being requested. Thus, requests for keys that are not specifically for session-level  
28 information must be accompanied by the **PMIX\_SESSION\_INFO** qualifier.

29 Example requests are shown below:

```
1 pmix_info_t info;
2 pmix_value_t *value;
3 pmix_status_t rc;
4 pmix_proc_t myproc, wildcard;
5
6 /* initialize the client library */
7 PMIx_Init(&myproc, NULL, 0);
8
9 /* get the #slots in our session */
10 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
11 rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);
12
13 /* get the #nodes in our session */
14 PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
15 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

Information regarding a different session can be requested by adding the **PMIX\_SESSION\_ID** attribute identifying the target session. In this case, the *proc* argument to **PMIx\_Get** will be ignored:

```
19 pmix_info_t info[2];
20 pmix_value_t *value;
21 pmix_status_t rc;
22 pmix_proc_t myproc;
23 uint32_t sid;
24
25 /* initialize the client library */
26 PMIx_Init(&myproc, NULL, 0);
27
28 /* get the #nodes in a different session */
29 sid = 12345;
30 PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
31 PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
32 rc = PMIx_Get(NULL, PMIX_NUM_NODES, info, 2, &value);
```

### 1 6.2.1.2 Job-level information

2 Information regarding a job can be obtained by the methods detailed in Section 6.1.2. Example  
3 requests are shown below:

```
4 pmix_info_t info;
5 pmix_value_t *value;
6 pmix_status_t rc;
7 pmix_proc_t myproc, wildcard;
8
9 /* initialize the client library */
10 PMIx_Init(&myproc, NULL, 0);
11
12 /* get the #apps in our job */
13 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
14 rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);
15
16 /* get the #nodes in our job */
17 PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
18 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

### 19 6.2.1.3 Application-level information

20 Information regarding an application can be obtained by the methods described in Section 6.1.3.  
21 Example requests are shown below:

```
22 pmix_info_t info;
23 pmix_value_t *value;
24 pmix_status_t rc;
25 pmix_proc_t myproc, otherproc;
26 uint32_t appsize, appnum;
27
28 /* initialize the client library */
29 PMIx_Init(&myproc, NULL, 0);
30
31 /* get the #processes in our application */
32 rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
33 appsize = value->data.uint32;
34
35 /* get the #nodes in an application containing "otherproc".
36 * For this use-case, assume that we are in the first application
37 * and we want the #nodes in the second application - use the
```

```

1      * rank of the first process in that application, remembering
2      * that ranks start at zero */
3      PMIX_PROC_LOAD(&otherproc, myproc.nspace, appsize);
4
5      /* Since "otherproc" refers to a process in the second application,
6      * we can simply mark that we want the info for this key from the
7      * application realm */
8      PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
9      rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);
10
11     /* alternatively, we can directly ask for the #nodes in
12     * the second application in our job, again remembering that
13     * application numbers start with zero. Since we are asking
14     * for application realm information about a specific appnum
15     * within our own namespace, the process identifier can be NULL */
16     appnum = 1;
17     PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
18     PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
19     rc = PMIx_Get(NULL, PMIX_NUM_NODES, appinfo, 2, &value);

```



## 6.2.1.4 Process-level information

Process-level information is accessed by providing the namespace and rank of the target process. In the absence of any directive as to the level of information being requested, the PMIx library will always return the process-level value. See Section 6.1.4 for details.

## 6.2.1.5 Node-level information

Information regarding a node within the system can be obtained by the methods described in Section 6.1.5. Example requests are shown below:



```

27 pmix_info_t info[2];
28 pmix_value_t *value;
29 pmix_status_t rc;
30 pmix_proc_t myproc, otherproc;
31 uint32_t nodeid;
32
33 /* initialize the client library */
34 PMIx_Init(&myproc, NULL, 0);
35
36 /* get the #procs on our node */
37 rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);
38

```

```
1  /* get the #slots on another node */
2  PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
3  PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
4  rc = PMIx_Get(NULL, PMIX_MAX_PROCS, info, 2, &value);
5
6  /* get the total #procs on the remote node - note that we don't
7   * actually need to include the "PMIX_NODE_INFO" attribute here,
8   * but (a) it does no harm and (b) it allowed us to simply reuse
9   * the prior info array
10 rc = PMIx_Get(NULL, PMIX_NODE_SIZE, info, 2, &value);
```



## CHAPTER 7

# Process-Related Non-Reserved Keys

---

1     *Non-reserved keys* are keys whose string representation begin with a prefix other than "**pmix**".  
2     Such keys are typically defined by an application when information needs to be exchanged between  
3     processes (e.g., where connection information is required and the host environment does not  
4     support the *instant on* option) or where the host environment does not provide a required piece of  
5     data. Other than the prefix, there are no restrictions on the use or content of non-reserved keys.

6     PMIx provides support for two methods of exchanging non-reserved keys:

- 7       • Global, collective exchange of the information prior to retrieval. This is accomplished by  
8           executing a barrier operation that includes collection and exchange of the data provided by each  
9           process such that each process has access to the full set of data from all participants once the  
10          operation has completed. PMIx provides the **PMIx\_Fence** function (or its non-blocking  
11          equivalent) for this purpose, accompanied by the **PMIX\_COLLECT\_DATA** qualifier.
- 12       • Direct, on-demand retrieval of the information. No barrier or global exchange is conducted in  
13           this case. Instead, information is retrieved from the host where that process is executing upon  
14           request - i.e., a call to **PMIx\_Get** results in a data exchange with the PMIx server on the remote  
15           host. Various caching strategies may be employed by the host environment and/or PMIx  
16           implementation to reduce the number of retrievals. Note that this method requires that the host  
17           environment both know the location of the posting process and support direct information  
18           retrieval.

19     Both of the above methods are based on retrieval from a specific process - i.e., the *proc* argument to  
20       **PMIx\_Get** must include both the namespace and the rank of the process that posted the  
21       information. However, in some cases, non-reserved keys are provided on a globally unique basis  
22       and the retrieving process has no knowledge of the identity of the process posting the key. This is  
23       typically found in legacy applications (where the originating process identifier is often embedded in  
24       the key itself) and in unstructured applications that lack rank-related behavior. In these cases, the  
25       key remains associated with the namespace of the process that posted it, but is retrieved by use of  
26       the **PMIX\_RANK\_UNDEF** rank. In addition, the keys must be globally exchanged prior to retrieval  
27       as there is no way for the host to otherwise locate the source for the information.

28     Note that the retrieval rules for non-reserved keys (detailed in Section 7.2) differ significantly from  
29       those used for reserved keys.

## 1 7.1 Posting Key/Value Pairs

2 PMIx clients can post non-reserved key-value pairs associated with themselves by using  
3 **PMIx\_Put**. Alternatively, PMIx clients can cache arbitrary key-value pairs accessible only by the  
4 caller via the **PMIx\_Store\_internal** API.

### 5 7.1.1 **PMIx\_Put**

#### 6 Summary

7 Post a key/value pair for distribution.

#### 8 Format

9 *PMIx v1.0*

```
10 pmix_status_t  
11 PMIx_Put (pmix_scope_t scope,  
12           const pmix_key_t key,  
13           pmix_value_t *val);
```

C

C

#### 13 IN scope

14 Distribution scope of the provided value (handle)

#### 15 IN key

16 key (**pmix\_key\_t**)

#### 17 IN value

18 Reference to a **pmix\_value\_t** structure (handle)

19 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. If a reserved  
20 key is provided in the *key* argument then **PMIx\_Put** will return **PMIX\_ERR\_BAD\_PARAM**.

#### 21 Description

22 Post a key-value pair for distribution. Depending upon the PMIx implementation, the posted value  
23 may be locally cached in the client's PMIx library until **PMIx\_Commit** is called.

24 The provided *scope* determines the ability of other processes to access the posted data, as defined in  
25 Section 7.1.1.1 on page 107. Specific implementations may support different scope values, but all  
26 implementations must support at least **PMIX\_GLOBAL**.

27 The **pmix\_value\_t** structure supports both string and binary values. PMIx implementations are  
28 required to support heterogeneous environments by properly converting binary values between host  
29 architectures, and will copy the provided *value* into internal memory prior to returning from  
30 **PMIx\_Put**.

#### 31 Advice to users

32 Note that keys starting with a string of “**pmix**” must not be used in calls to **PMIx\_Put**. Thus,  
applications should never use a defined “PMIX” attribute as the key in a call to **PMIx\_Put**.

### 7.1.1.1 Scope of Put Data

*PMIx v1.0* The `pmix_scope_t` structure is a `uint8_t` type that defines the availability of data passed to `PMIx_Put`. The following constants can be used to set a variable of the type `pmix_scope_t`. All definitions were introduced in version 1 of the standard unless otherwise marked.

Specific implementations may support different scope values, but all implementations must support at least `PMIX_GLOBAL`. If a specified scope value is not supported, then the `PMIx_Put` call must return `PMIX_ERR_NOT_SUPPORTED`.

`PMIX_SCOPE_UNDEF` Undefined scope.

`PMIX_LOCAL` The data is intended only for other application processes on the same node. Data marked in this way will not be included in data packages sent to remote requesters - i.e., it is only available to processes on the local node.

`PMIX_REMOTE` The data is intended solely for applications processes on remote nodes. Data marked in this way will not be shared with other processes on the same node - i.e., it is only available to processes on remote nodes.

`PMIX_GLOBAL` The data is to be shared with all other requesting processes, regardless of location.

*PMIx v2.0* `PMIX_INTERNAL` The data is intended solely for this process and is not shared with other processes.

### 7.1.2 PMIx\_Store\_internal

#### Summary

Store some data locally for retrieval by other areas of the process.

#### Format

*PMIx v1.0*

```
pmix_status_t  
PMIx_Store_internal(const pmix_proc_t *proc,  
                      const pmix_key_t key,  
                      pmix_value_t *val);
```

**IN proc**  
process reference (handle)  
**IN key**  
key to retrieve (string)  
**IN val**  
Value to store (handle)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant. If a reserved key is provided in the `key` argument then `PMIx_Store_internal` will return `PMIX_ERR_BAD_PARAM`.

1           **Description**

2       Store some data locally for retrieval by other areas of the process. This is data that has only internal  
3       scope - it will never be posted externally. Typically used to cache data obtained by means outside of  
4       PMIx so that it can be accessed by various areas of the process.

5           **7.1.3 PMIx\_Commit**

6           **Summary**

7       Post all previously **PMIx\_Put** values for distribution.

8           **Format**

9       *PMIx v1.0*

10      

```
pmix_status_t PMIx_Commit(void);
```

11      Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

12           **Description**

13       PMIx implementations may choose to locally cache non-reserved keys prior to submitting them for  
14       distribution. Accordingly, PMIx provides a second API specifically to stage all previously posted  
15       data for distribution - e.g., by transmitting the entire collection of data posted by the process to a  
16       server in one operation. This is an asynchronous operation that will immediately return to the caller  
     while the data is staged in the background.

17           **Advice to users**

18       Users are advised to always include the call to **PMIx\_Commit** in case the local implementation  
19       requires it. Note that posted data will not be circulated during **PMIx\_Commit**. Availability of the  
20       data by other processes upon completion of **PMIx\_Commit** therefore still relies upon the exchange  
     mechanisms described at the beginning of this chapter.

## 1    7.2 Retrieval rules for non-reserved keys

2    Since non-reserved keys cannot, by definition, have been provided by the host environment, their  
3    retrieval follows significantly different rules than those defined for reserved keys (as detailed in  
4    Section 6.2). **PMIx\_Get** for a non-reserved key will obey the following precedence search:

- 5    1. If the **PMIX\_GET\_REFRESH\_CACHE** attribute is given, then the request is first forwarded to  
6    the local PMIx server which will then update the client's cache. Note that this may not,  
7    depending upon implementation details, result in any action.
  - 8    2. Check the local PMIx client cache for the requested key - if not found and either the  
9    **PMIX\_OPTIONAL** or **PMIX\_GET\_REFRESH\_CACHE** attribute was given, the search will stop  
10   at this point and return the **PMIX\_ERR\_NOT\_FOUND** status.
  - 11   3. Request the information from the local PMIx server. The server will check its cache for the  
12   specified key. If the value still isn't found and the **PMIX\_IMMEDIATE** attribute was given, then  
13   the library shall return the **PMIX\_ERR\_NOT\_FOUND** error constant to the requester. Otherwise,  
14   the PMIx server library will take one of the following actions:
    - 15   • If the target process has a rank of **PMIX\_RANK\_UNDEF**, then this indicates that the key being  
16   requested is globally unique and *not* associated with a specific process. In this case, the server  
17   shall hold the request until either the data appears at the server or, if given, the  
18   **PMIX\_TIMEOUT** is reached. In the latter case, the server will return the  
19   **PMIX\_ERR\_TIMEOUT** status. Note that the server may, depending on PMIx implementation,  
20   never respond if the caller failed to specify a **PMIX\_TIMEOUT** and the requested key fails to  
21   arrive at the server.
    - 22   • If the target process is *local* (i.e., attached to the same PMIx server), then the server will hold  
23   the request until either the target process provides the data or, if given, the **PMIX\_TIMEOUT**  
24   is reached. In the latter case, the server will return the **PMIX\_ERR\_TIMEOUT** status. Note  
25   that data which is posted via **PMIx\_Put** but not staged with **PMIx\_Commit** may, depending  
26   upon implementation, never appear at the server.
    - 27   • If the target process is *remote* (i.e., not attached to the same PMIx server), the server will  
28   either:
      - 29   – If the host has provided the **pmix\_server\_dmodex\_req\_fn\_t** module function  
30   interface, then the server shall pass the request to its host for servicing. The host is  
31   responsible for determining the location of the target process and passing the request to the  
32   PMIx server at that location.
- 33   When the remote data request is received, the target PMIx server will check its cache for  
34   the specified key. If the key is not present, the request shall be held until either the target  
35   process provides the data or, if given, the **PMIX\_TIMEOUT** is reached. In the latter case,  
36   the server will return the **PMIX\_ERR\_TIMEOUT** status. The host shall convey the result  
37   back to the originating PMIx server, which will reply to the requesting client with the result  
38   of the request when the host provides it.

1 Note that the target server may, depending on PMIx implementation, never respond if the  
2 caller failed to specify a **PMIX\_TIMEOUT** and the target process fails to post the requested  
3 key.

- 4 – if the host does not support the **pmix\_server\_dmode\_x\_req\_fn\_t** interface, then the  
5 server will immediately respond to the client with the **PMIX\_ERR\_NOT\_FOUND** status

---

**Advice to PMIx library implementers**

---

6 While there is no requirement that all PMIx implementations follow the client-server paradigm  
7 used in the above description, implementers are required to provide behaviors consistent with the  
8 described search pattern.

---

**Advice to users**

---

9 Users are advised to always specify the **PMIX\_TIMEOUT** value when retrieving non-reserved keys  
10 to avoid potential deadlocks should the specified key not become available.

## CHAPTER 8

# Publish/Lookup Operations

Chapter 6 and Chapter 7 discussed how reserved and non-reserved keys dealt with information that either was associated with a specific process (i.e., the retrieving process knew the identifier of the process that posted it) or required a synchronization operation prior to retrieval (e.g., the case of globally unique non-reserved keys). However, another requirement exists for an asynchronous exchange of data where neither the posting nor the retrieving process is known in advance. For example, two separate namespaces may need to rendezvous with each other without knowing in advance the identity of the other namespace or when that namespace might become active.

The APIs defined in this section focus on resolving that specific situation by allowing processes to publish data that can subsequently be retrieved solely by referral to its key. Mechanisms for constraining availability of the information are also provided as a means for better targeting of the eventual recipient(s).

Note that no presumption is made regarding how the published information is to be stored, nor as to the entity (host environment or PMIx implementation) that shall act as the datastore. The descriptions in the remainder of this chapter shall simply refer to that entity as the *datastore*.

## 8.1 PMIx\_Publish

### Summary

Publish data for later access via [PMIx\\_Lookup](#).

### Format

PMIx v1.0

```
pmix_status_t  
PMIx_Publish(const pmix_info_t info[], size_t ninfo);
```

#### IN info

Array of info structures containing both data to be published and directives (array of handles)

#### IN ninfo

Number of elements in the *info* array (integer)

Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

## Required Attributes

There are no required attributes for this API. PMIx implementations that do not directly support the operation but are hosted by environments that do support it must pass any attributes that are provided by the client to the host environment for processing. In addition, the PMIx library is required to add the **PMIX\_USERID** and the **PMIX\_GRP\_ID** attributes of the client process that published the information to the *info* array passed to the host environment.

## Optional Attributes

The following attributes are optional for host environments that support this operation:

- PMIX\_TIMEOUT** "pmix.timeout" (int)  
Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.
- PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)  
Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.
- PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)  
Declare how long the datastore shall retain the provided data. The datastore is to delete the data upon reaching the persistence criterion.
- PMIX\_ACCESS\_PERMISSIONS** "pmix.aperms" (pmix\_data\_array\_t)  
Define access permissions for the published data. The value shall contain an array of **pmix\_info\_t** structs containing the specified permissions.

## Description

Publish the data in the *info* array for subsequent lookup. By default, the data will be published into the **PMIX\_RANGE\_SESSION** range and with **PMIX\_PERSIST\_APP** persistence. Changes to those values, and any additional directives, can be included in the **pmix\_info\_t** array. Attempts to access the data by processes outside of the provided data range shall be rejected. The **PMIX\_PERSISTENCE** attribute instructs the datastore holding the published information as to how long that information is to be retained.

The blocking form of this call will block until it has obtained confirmation from the datastore that the data is available for lookup. The *info* array can be released upon return from the blocking function call.

Publishing duplicate keys is permitted provided they are published to different ranges. Duplicate keys being published on the same data range shall return the **PMIX\_ERR\_DUPLICATE\_KEY** error.

## 1 8.2 PMIx\_Publish\_nb

### 2 Summary

3 Nonblocking [PMIx\\_Publish](#) routine.

### 4 Format

5 *PMIx v1.0*

C

```
6     pmix_status_t  
7     PMIx_Publish_nb(const pmix_info_t info[], size_t ninfo,  
8                         pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

#### 9 IN **info**

10 Array of info structures containing both data to be published and directives (array of handles)

#### 11 IN **ninfo**

12 Number of elements in the *info* array (integer)

#### 13 IN **cbfunc**

14 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

#### 15 IN **cbdata**

16 Data to be passed to the callback function (memory reference)

17 Returns one of the following:

- 18 • [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
19 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
function prior to returning from the API.
- 20 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
21 returned *success* - the *cbfunc* will *not* be called.
- 22 • a PMIx error constant indicating either an error in the input or that the request was immediately  
23 processed and failed - the *cbfunc* will *not* be called.

### ▼----- Required Attributes -----▼

24 There are no required attributes for this API. PMIx implementations that do not directly support the  
25 operation but are hosted by environments that do support it must pass any attributes that are  
26 provided by the client to the host environment for processing. In addition, the PMIx library is  
27 required to add the [PMIX\\_USERID](#) and the [PMIX\\_GRP\\_ID](#) attributes of the client process that  
28 published the information to the *info* array passed to the host environment.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (zero indicating infinite) and  
4 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
5 caused by multiple layers (client, server, and host) simultaneously timing the operation.

6 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

7 Define constraints on the processes that can access the provided data. Only processes that  
8 meet the constraints are allowed to access it.

9 **PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)

10 Declare how long the datastore shall retain the provided data. The datastore is to delete the  
11 data upon reaching the persistence criterion.

12 **PMIX\_ACCESS\_PERMISSIONS** "pmix.aperms" (pmix\_data\_array\_t)

13 Define access permissions for the published data. The value shall contain an array of  
14 **pmix\_info\_t** structs containing the specified permissions.

### 15 **Description**

16 Nonblocking **PMIx\_Publish** routine.

## 17 **8.3 Publish-specific constants**

18 The following constants are defined for use with the **PMIx\_Publish** APIs:

19 **PMIX\_ERR\_DUPLICATE\_KEY** The provided key has already been published on the same  
20 data range.

## 21 **8.4 Publish-specific attributes**

22 The following attributes are defined for use with the **PMIx\_Publish** APIs:

23 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

24 Define constraints on the processes that can access the provided data. Only processes that  
25 meet the constraints are allowed to access it.

26 **PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)

27 Declare how long the datastore shall retain the provided data. The datastore is to delete the  
28 data upon reaching the persistence criterion.

29 **PMIX\_ACCESS\_PERMISSIONS** "pmix.aperms" (pmix\_data\_array\_t)

30 Define access permissions for the published data. The value shall contain an array of  
31 **pmix\_info\_t** structs containing the specified permissions.

```
1 PMIX_ACCESS_USERIDS "pmix.auids" (pmix_data_array_t)
2     Array of effective User IDs (UIDs) that are allowed to access the published data.
3 PMIX_ACCESS_GRPIDS "pmix.agids" (pmix_data_array_t)
4     Array of effective Group IDs (GIDs) that are allowed to access the published data.
```

## 5 **8.5 Publish-Lookup Datatypes**

6 The following data types are defined for use with the **PMIx\_Publish** APIs.

### 7 **8.5.1 Range of Published Data**

8 *PMIx v1.0* The **pmix\_data\_range\_t** structure is a **uint8\_t** type that defines a range for both data  
9 *published* via the **PMIx\_Publish** API and generated events. The following constants can be used  
10 to set a variable of the type **pmix\_data\_range\_t**.

11 **PMIX\_RANGE\_UNDEF** Undefined range.

12 **PMIX\_RANGE\_RM** Data is intended for the host environment, or lookup is restricted to data  
13 published by the host environment.

14 **PMIX\_RANGE\_LOCAL** Data is only available to processes on the local node, or lookup is  
15 restricted to data published by processes on the local node of the requester.

16 **PMIX\_RANGE\_NAMESPACE** Data is only available to processes in the same namespace, or  
17 lookup is restricted to data published by processes in the same namespace as the requester.

18 **PMIX\_RANGE\_SESSION** Data is only available to all processes in the session, or lookup is  
19 restricted to data published by other processes in the same session as the requester.

20 **PMIX\_RANGE\_GLOBAL** Data is available to all processes, or lookup is open to data published  
21 by anyone.

22 **PMIX\_RANGE\_CUSTOM** Data is available only to processes as specified in the  
23 **pmix\_info\_t** associated with this call, or lookup is restricted to data published by  
24 processes as specified in the **pmix\_info\_t**.

25 **PMIX\_RANGE\_PROC\_LOCAL** Data is only available to this process, or lookup is restricted to  
26 data published by this process.

27 **PMIX\_RANGE\_INVALID** Invalid value - typically used to indicate that a range has not yet  
28 been set.

### 29 **8.5.2 Data Persistence Structure**

30 *PMIx v1.0* The **pmix\_persistence\_t** structure is a **uint8\_t** type that defines the policy for data  
31 *published* by clients via the **PMIx\_Publish** API. The following constants can be used to set a  
32 variable of the type **pmix\_persistence\_t**.

33 **PMIX\_PERSIST\_INDEF** Retain data until specifically deleted.

34 **PMIX\_PERSIST\_FIRST\_READ** Retain data until the first access, then the data is deleted.

35 **PMIX\_PERSIST\_PROC** Retain data until the publishing process terminates.

36 **PMIX\_PERSIST\_APP** Retain data until the application terminates.

37 **PMIX\_PERSIST\_SESSION** Retain data until the session/allocation terminates.

38 **PMIX\_PERSIST\_INVALID** Invalid value - typically used to indicate that a persistence has  
39 not yet been set.

## 1 8.6 PMIx\_Lookup

### 2 Summary

3 Lookup information published by this or another process with [PMIx\\_Publish](#) or  
4 [PMIx\\_Publish\\_nb](#).

### 5 Format

6 *PMIx v1.0*

C

```
7     pmix_status_t
8     PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
9                     const pmix_info_t info[], size_t ninfo);
```

C

### 9 INOUT data

10 Array of publishable data structures (array of [pmix\\_pdata\\_t](#))

### 11 IN ndata

12 Number of elements in the *data* array (integer)

### 13 IN info

14 Array of info structures (array of [pmix\\_info\\_t](#))

### 15 IN ninfo

16 Number of elements in the *info* array (integer)

17 Returns one of the following:

- 18 • [PMIX\\_SUCCESS](#) All data was found and has been returned.
- 19 • [PMIX\\_ERR\\_NOT\\_FOUND](#) None of the requested data could be found within the requester's  
20 range.
- 21 • [PMIX\\_ERR\\_PARTIAL\\_SUCCESS](#) Some of the requested data was found. Any key that cannot  
22 be found will return with a data type of [PMIX\\_UNDEF](#) in the associated *value* struct. Note that  
23 the specific reason for a particular piece of missing information (e.g., lack of permissions) cannot  
24 be communicated back to the requester in this situation.
- 25 • [PMIX\\_ERR\\_NOT\\_SUPPORTED](#) There is no available datastore (either at the host environment  
26 or PMIx implementation level) on this system that supports this function.
- 27 • [PMIX\\_ERR\\_NO\\_PERMISSIONS](#) All of the requested data was found and range restrictions  
28 were met for each specified key, but none of the matching data could be returned due to lack of  
29 access permissions.
- 30 • a non-zero PMIx error constant indicating a reason for the request's failure.

## Required Attributes

1 PMIx libraries are not required to directly support any attributes for this function. However, any  
2 provided attributes must be passed to the host environment for processing, and the PMIx library is  
3 required to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process that is  
4 requesting the info.

## Optional Attributes

5 The following attributes are optional for host environments that support this operation:

6 **PMIX\_TIMEOUT** "pmix.timeout" (int)

7 Time in seconds before the specified operation should time out (zero indicating infinite) and  
8 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
9 caused by multiple layers (client, server, and host) simultaneously timing the operation.

10 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

11 Define constraints on the processes that can access the provided data. Only processes that  
12 meet the constraints are allowed to access it.

13 **PMIX\_WAIT** "pmix.wait" (int)

14 Caller requests that the PMIx server wait until at least the specified number of values are  
15 found (a value of zero indicates *all* and is the default).

## Description

16 Lookup information published by this or another process. By default, the search will be constrained  
17 to publishers that fall within the **PMIX\_RANGE\_SESSION** range in case duplicate keys exist on  
18 different ranges. Changes to the range (e.g., expanding the search to all potential publishers via the  
20 **PMIX\_RANGE\_GLOBAL** constant), and any additional directives, can be provided in the  
21 **pmix\_info\_t** array. Data is returned per the retrieval rules of Section 8.8.

22 The *data* parameter consists of an array of **pmix\_pdata\_t** structures with the keys specifying the  
23 requested information. Data will be returned for each **key** field in the associated **value** field of  
24 this structure as per the above description of return values. The **proc** field in each  
25 **pmix\_pdata\_t** structure will contain the namespace/rank of the process that published the data.

## Advice to users

26 Although this is a blocking function, it will not wait by default for the requested data to be  
27 published. Instead, it will block for the time required by the datastore to lookup its current data and  
28 return any found items. Thus, the caller is responsible for either ensuring that data is published  
29 prior to executing a lookup, using **PMIX\_WAIT** to instruct the datastore to wait for the data to be  
30 published, or retrying until the requested data is found.

## 1 8.7 PMIx\_Lookup\_nb

### 2 Summary

3 Nonblocking version of [PMIx\\_Lookup](#).

### 4 Format

5 *PMIx v1.0*

C

```
6     pmix_status_t  
7     PMIx_Lookup_nb(char **keys,  
8                         const pmix_info_t info[], size_t ninfo,  
9                         pmix_lookup_cbfunc_t cbfunc, void *cbdata);
```

C

#### 9 IN keys

10 NULL-terminated array of keys (array of strings)

#### 11 IN info

12 Array of info structures (array of handles)

#### 13 IN ninfo

14 Number of elements in the *info* array (integer)

#### 15 IN cbfunc

16 Callback function (handle)

#### 17 IN cbdata

18 Callback data to be provided to the callback function (pointer)

19 Returns one of the following:

- 20 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
21 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
22 function prior to returning from the API.
- 23 • a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called.

24 If executed, the status returned in the provided callback function will be one of the following  
25 constants:

- 26 • **PMIX\_SUCCESS** All data was found and has been returned.
- 27 • **PMIX\_ERR\_NOT\_FOUND** None of the requested data was available within the requester's range.  
28 The *pdata* array in the callback function shall be **NULL** and the *npdata* parameter set to zero.
- 29 • **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the requested data was found. Only found data will  
30 be included in the returned *pdata* array. Note that the specific reason for a particular piece of  
31 missing information (e.g., lack of permissions) cannot be communicated back to the requester in  
32 this situation.
- 33 • **PMIX\_ERR\_NOT\_SUPPORTED** There is no available datastore (either at the host environment  
34 or PMIx implementation level) on this system that supports this function.

- **PMIX\_ERR\_NO\_PERMISSIONS** All of the requested data was found and range restrictions were met for each specified key, but none of the matching data could be returned due to lack of access permissions.
  - a non-zero PMIx error constant indicating a reason for the request's failure.

## Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host environment for processing, and the PMIx library is required to add the `PMIX_USERID` and the `PMIX_GRP_ID` attributes of the client process that is requesting the info.

## Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX\_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

**PMIX\_WAIT** "pmix.wait" (int)

Caller requests that the PMIx server wait until at least the specified number of values are found (a value of zero indicates *all* and is the default).

## Description

Non-blocking form of the **PMIx Lookup** function.

## 8.7.1 Lookup Returned Data Structure

The `pmix_pdata_t` structure is used by `PMIx_Lookup` to describe the data being accessed.

*PMIx v1.0*

```
3     typedef struct pmix_pdata {
4         pmix_proc_t proc;
5         pmix_key_t key;
6         pmix_value_t value;
7     } pmix_pdata_t;
```

where:

- *proc* is the process identifier of the data publisher.
- *key* is the string key of the published data.
- *value* is the value associated with the *key*.

### 8.7.1.1 Lookup data structure support macros

The following macros are provided to support the `pmix_pdata_t` structure.

#### Initialize the pdata structure

Initialize the `pmix_pdata_t` fields

*PMIx v1.0*

```
16 PMIX_PDATA_CONSTRUCT (m)
```

**IN** *m*

Pointer to the structure to be initialized (pointer to `pmix_pdata_t`)

#### Destruct the pdata structure

Destruct the `pmix_pdata_t` fields

*PMIx v1.0*

```
21 PMIX_PDATA_DESTRUCT (m)
```

**IN** *m*

Pointer to the structure to be destructed (pointer to `pmix_pdata_t`)

```

1      Create a pdata array
2      Allocate and initialize an array of pmix_pdata_t structures
3      PMIx v1.0   ▼———— C —————▶
4          PMIX_PDATA_CREATE (m, n)   ▲———— C —————▶
5          INOUT m           Address where the pointer to the array of pmix_pdata_t structures shall be stored (handle)
6          IN n             Number of structures to be allocated (size_t)
7
8      Free a pdata structure
9      Release a pmix_pdata_t structure
10     PMIx v4.0   ▼———— C —————▶
11        PMIX_PDATA_RELEASE (m)   ▲———— C —————▶
12        IN m           Pointer to a pmix_pdata_t structure (handle)
13      Free a pdata array
14      Release an array of pmix_pdata_t structures
15     PMIx v1.0   ▼———— C —————▶
16        PMIX_PDATA_FREE (m, n)   ▲———— C —————▶
17        IN m           Pointer to the array of pmix_pdata_t structures (handle)
18        IN n             Number of structures in the array (size_t)
19
20      Load a lookup data structure
21      This macro simplifies the loading of key, process identifier, and data into a pmix_pdata_t by
22      correctly assigning values to the structure's fields.

```

*PMIx v1.0*

```
1 PMIX_PDATA_LOAD(m, p, k, d, t); C  
2 IN m  
3 Pointer to the pmix_pdata_t structure into which the key and data are to be loaded  
4 (pointer to pmix_pdata_t)  
5 IN p  
6 Pointer to the pmix_proc_t structure containing the identifier of the process being  
7 referenced (pointer to pmix_proc_t)  
8 IN k  
9 String key to be loaded - must be less than or equal to PMIX_MAX_KEYLEN in length  
10 (handle)  
11 IN d  
12 Pointer to the data value to be loaded (handle)  
13 IN t  
14 Type of the provided data value (pmix_data_type_t)
```

#### Advice to users

Key, process identifier, and data will all be copied into the pmix\_pdata\_t - thus, the source information can be modified or free'd without affecting the copied data once the macro has completed.

### Transfer a lookup data structure

This macro simplifies the transfer of key, process identifier, and data value between two pmix\_pdata\_t structures.

```
PMIx v2.0  
1 PMIX_PDATA_XFER(d, s); C  
2 IN d  
3 Pointer to the destination pmix_pdata_t (pointer to pmix_pdata_t)  
4 IN s  
5 Pointer to the source pmix_pdata_t (pointer to pmix_pdata_t)
```

#### Advice to users

Key, process identifier, and data will all be copied into the destination pmix\_pdata\_t - thus, the source pmix\_pdata\_t may free'd without affecting the copied data once the macro has completed.

## 1 8.7.2 Lookup Callback Function

### 2 Summary

3 The `pmix_lookup_cbfunc_t` is used by `PMIx_Lookup_nb` to return data.

4 *PMIx v1.0*

C

```
5     typedef void (*pmix_lookup_cbfunc_t)
6         (pmix_status_t status,
7          pmix_pdata_t data[], size_t ndata,
8          void *cbdata);
```

C

#### 9 IN **status**

10 Status associated with the operation (handle)

#### 11 IN **data**

12 Array of data returned (`pmix_pdata_t`)

#### 13 IN **ndata**

14 Number of elements in the *data* array (`size_t`)

#### 15 IN **cbdata**

16 Callback data passed to original API call (memory reference)

### 17 Description

18 A callback function for calls to `PMIx_Lookup_nb`. The function will be called upon completion  
19 of the `PMIx_Lookup_nb` API with the *status* indicating the success or failure of the request. Any  
20 retrieved data will be returned in an array of `pmix_pdata_t` structs. The namespace and rank of  
the process that provided each data element is also returned.

21 Note that the `pmix_pdata_t` structures will be released upon return from the callback function,  
22 so the receiver must copy/protect the data prior to returning if it needs to be retained.

## 23 8.8 Retrieval rules for published data

24 The retrieval rules for published data primarily revolve around enforcing data access permissions  
25 and range constraints. The datastore shall search its stored information for each specified key  
26 according to the following precedence logic:

- 27 1. If the requester specified the range, then the search shall be constrained to data where the  
28 publishing process falls within the specified range.
- 29 2. If the key of the stored information does not match the specified key, then the search will  
30 continue.
- 31 3. If the requester's identifier does not fall within the range specified by the publisher, then the  
32 search will continue.

1     4. If the publisher specified access permissions, the effective UID and GID of the requester shall be  
2        checked against those permissions, with the datastore rejecting the match if the requester fails to  
3        meet the requirements.

4     5. If all of the above checks pass, then the value is added to the information that is to be returned.

5       The status returned by the datastore shall be set to:

- 6
  - **PMIX\_SUCCESS** All data was found and is included in the returned information.
  - **PMIX\_ERR\_NOT\_FOUND** None of the requested data could be found within a requester's range.
  - **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the requested data was found. Only found data will be included in the returned information. Note that the specific reason for a particular piece of missing information (e.g., lack of permissions) cannot be communicated back to the requester in this situation.
  - a non-zero PMIx error constant indicating a reason for the request's failure.

13      In the case where data was found and range restrictions were met for each specified key, but none of  
14       the matching data could be returned due to lack of access permissions, the datastore must return the  
15       **PMIX\_ERR\_NO\_PERMISSIONS** error.

---

### Advice to users

---

16      Note that duplicate keys are allowed to exist on different ranges, and that ranges do overlap each  
17       other. Thus, if duplicate keys are published on overlapping ranges, it is possible for the datastore to  
18       successfully find multiple responses for a given key should publisher and requester specify  
19       sufficiently broad ranges. In this situation, the choice of resolving the duplication is left to the  
20       datastore implementation - e.g., it may return the first value found in its search, or the value  
21       corresponding to the most limited range of the found values, or it may choose to simply return an  
22       error.

23      Users are advised to avoid this ambiguity by careful selection of key values and ranges - e.g., by  
24       creating range-specific keys where necessary.

---

## 25     8.9 PMIx\_Unpublish

### 26       Summary

27       Unpublish data posted by this process using the given keys.

1           **Format**

2        *pmix\_status\_t*  
3        *PMIx\_Unpublish(char \*\*keys,*  
4                    *const pmix\_info\_t info[], size\_t ninfo);*

- 5     **IN keys**  
6        NULL-terminated array of keys (array of strings)  
7     **IN info**  
8        Array of info structures (array of handles)  
9     **IN ninfo**  
10      Number of elements in the *info* array (integer)

11     Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

▼----- **Required Attributes** -----▼

12     PMIx libraries are not required to directly support any attributes for this function. However, any  
13     provided attributes must be passed to the host environment for processing, and the PMIx library is  
14     required to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process that is  
15     requesting the operation.

▼----- **Optional Attributes** -----▼

16     The following attributes are optional for host environments that support this operation:

17     **PMIX\_TIMEOUT "pmix.timeout" (int)**  
18        Time in seconds before the specified operation should time out (zero indicating infinite) and  
19        return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
20        caused by multiple layers (client, server, and host) simultaneously timing the operation.

21     **PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t)**  
22        Define constraints on the processes that can access the provided data. Only processes that  
23        meet the constraints are allowed to access it.

24           **Description**

25     Unpublish data posted by this process using the given *keys*. The function will block until the data  
26     has been removed by the server (i.e., it is safe to publish that key again within the specified range).  
27     A value of **NULL** for the *keys* parameter instructs the server to remove all data published by this  
28     process.

29     By default, the range is assumed to be **PMIX\_RANGE\_SESSION**. Changes to the range, and any  
30     additional directives, can be provided in the *info* array.

## 1 8.10 PMIx\_Unpublish\_nb

### 2 Summary

3 Nonblocking version of [PMIx\\_Unpublish](#).

### 4 Format

5 *PMIx v1.0*

C

```
6     pmix_status_t  
7     PMIx_Unpublish_nb(char **keys,  
8                         const pmix_info_t info[], size_t ninfo,  
9                         pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

#### 9 IN keys

10 NULL-terminated array of keys (array of strings)

#### 11 IN info

12 Array of info structures (array of handles)

#### 13 IN ninfo

14 Number of elements in the *info* array (integer)

#### 15 IN cbfunc

16 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

#### 17 IN cbdata

18 Data to be passed to the callback function (memory reference)

19 Returns one of the following:

- 20 • [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
21      will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
22      function prior to returning from the API.
- 23 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
24      returned *success* - the *cbfunc* will *not* be called.
- 25 • a PMIx error constant indicating either an error in the input or that the request was immediately  
26      processed and failed - the *cbfunc* will *not* be called.

### Required Attributes

27 PMIx libraries are not required to directly support any attributes for this function. However, any  
28 provided attributes must be passed to the host environment for processing, and the PMIx library is  
29 required to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process that is  
30 requesting the operation.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (zero indicating infinite) and  
4 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
5 caused by multiple layers (client, server, and host) simultaneously timing the operation.

6 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

7 Define constraints on the processes that can access the provided data. Only processes that  
8 meet the constraints are allowed to access it.

## 9 Description

10 Non-blocking form of the **PMIx\_Unpublish** function. The callback function will be executed  
11 once the server confirms removal of the specified data. The *info* array must be maintained until the  
12 callback is provided.

## CHAPTER 9

# Event Notification

---

This chapter defines the PMIx event notification system. These interfaces are designed to support the reporting of events to/from clients and servers, and between library layers within a single process.

## 9.1 Notification and Management

PMIx event notification provides an asynchronous out-of-band mechanism for communicating events between application processes and/or elements of the SMS. Its uses span a wide range including fault notification, coordination between multiple programming libraries within a single process, and workflow orchestration for non-synchronous programming models. Events can be divided into two distinct classes:

- *Job-specific events* directly relate to a job executing within the session, such as a debugger attachment, process failure within a related job, or events generated by an application process. Events in this category are to be immediately delivered to the PMIx server library for relay to the related local processes.
- *Environment events* indirectly relate to a job but do not specifically target the job itself. This category includes SMS-generated events such as Error Check and Correction (ECC) errors, temperature excursions, and other non-job conditions that might directly affect a session's resources, but would never include an event generated by an application process. Note that although these do potentially impact the session's jobs, they are not directly tied to those jobs. Thus, events in this category are to be delivered to the PMIx server library only upon request.

Both SMS elements and applications can register for events of either type.

### Advice to PMIx library implementers

Race conditions can cause the registration to come after events of possible interest (e.g., a memory ECC event that occurs after start of execution but prior to registration, or an application process generating an event prior to another process registering to receive it). SMS vendors are *requested* to cache environment events for some time to mitigate this situation, but are not *required* to do so. However, PMIx implementers are *required* to cache all events received by the PMIx server library and to deliver them to registering clients in the same order in which they were received

## Advice to users

1 Applications must be aware that they may not receive environment events that occur prior to  
2 registration, depending upon the capabilities of the host SMS.

3 The generator of an event can specify the *target range* for delivery of that event. Thus, the generator  
4 can choose to limit notification to processes on the local node, processes within the same job as the  
5 generator, processes within the same allocation, other threads within the same process, only the  
6 SMS (i.e., not to any application processes), all application processes, or to a custom range based  
7 on specific process identifiers. Only processes within the given range that register for the provided  
8 event code will be notified. In addition, the generator can use attributes to direct that the event not  
9 be delivered to any default event handlers, or to any multi-code handler (as defined below).

10 Event notifications provide the process identifier of the source of the event plus the event code and  
11 any additional information provided by the generator. When an event notification is received by a  
12 process, the registered handlers are scanned for their event code(s), with matching handlers  
13 assembled into an *event chain* for servicing. Note that users can also specify a *source range* when  
14 registering an event (using the same range designators described above) to further limit when they  
15 are to be invoked. When assembled, PMIx event chains are ordered based on both the specificity of  
16 the event handler and user directives at time of handler registration. By default, handlers are  
17 grouped into three categories based on the number of event codes that can trigger the callback:

- 18 • *single-code* handlers are serviced first as they are the most specific. These are handlers that are  
19 registered against one specific event code.
- 20 • *multi-code* handlers are serviced once all single-code handlers have completed. The handler will  
21 be included in the chain upon receipt of an event matching any of the provided codes.
- 22 • *default* handlers are serviced once all multi-code handlers have completed. These handlers are  
23 always included in the chain unless the generator specifically excludes them.

24 Users can specify the callback order of a handler within its category at the time of registration.  
25 Ordering can be specified either by providing the relevant returned event handler registration ID or  
26 using event handler names, if the user specified an event handler name when registering the  
27 corresponding event. Thus, users can specify that a given handler be executed before or after  
28 another handler should both handlers appear in an event chain (the ordering is ignored if the other  
29 handler isn't included). Note that ordering does not imply immediate relationships. For example,  
30 multiple handlers registered to be serviced after event handler *A* will all be executed after *A*, but are  
31 not guaranteed to be executed in any particular order amongst themselves.

32 In addition, one event handler can be declared as the *first* handler to be executed in the chain. This  
33 handler will *always* be called prior to any other handler, regardless of category, provided the  
34 incoming event matches both the specified range and event code. Only one handler can be so  
35 designated — attempts to designate additional handlers as *first* will return an error. Dereistration  
36 of the declared *first* handler will re-open the position for subsequent assignment.

1      Similarly, one event handler can be declared as the *last* handler to be executed in the chain. This  
2      handler will *always* be called after all other handlers have executed, regardless of category,  
3      provided the incoming event matches both the specified range and event code. Note that this  
4      handler will not be called if the chain is terminated by an earlier handler. Only one handler can be  
5      designated as *last* — attempts to designate additional handlers as *last* will return an error.  
6      Deregistration of the declared *last* handler will re-open the position for subsequent assignment.

### Advice to users

7      Note that the *last* handler is called *after* all registered default handlers that match the specified  
8      range of the incoming event unless a handler prior to it terminates the chain. Thus, if the application  
9      intends to define a *last* handler, it should ensure that no default handler aborts the process before it.

10     Upon completing its work and prior to returning, each handler *must* call the event handler  
11    completion function provided when it was invoked (including a status code plus any information to  
12    be passed to later handlers) so that the chain can continue being progressed. PMIx automatically  
13    aggregates the status and any results of each handler (as provided in the completion callback) with  
14    status from all prior handlers so that each step in the chain has full knowledge of what preceded it.  
15    An event handler can terminate all further progress along the chain by passing the  
16    [PMIX\\_EVENT\\_ACTION\\_COMPLETE](#) status to the completion callback function.

## 9.1.1 Events versus status constants

18    Return status constants (see Section 3.1.1) represent values that can be returned from or passed into  
19    PMIx APIs. These are distinct from PMIx *events* in that they are not values that can be registered  
20    against event handlers. In general, the two types of constants are distinguished by inclusion of an  
21    "ERR" in the name of error constants versus an "EVENT" in events, though there are exceptions  
22    (e.g, the [PMIX\\_SUCCESS](#) constant).

## 9.1.2 PMIx\_Register\_event\_handler

### Summary

Register an event handler.

## Format

```
pmix_status_t  
PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,  
                            pmix_info_t info[], size_t ninfo,  
                            pmix_notification_fn_t evhdlr,  
                            pmix_hdlr_reg_cbfunc_t cbfunc,  
                            void *cbdata);
```

- IN codes**  
Array of status codes (array of [pmix\\_status\\_t](#))
- IN ncodes**  
Number of elements in the *codes* array ([size\\_t](#))
- IN info**  
Array of info structures (array of handles)
- IN ninfo**  
Number of elements in the *info* array ([size\\_t](#))
- IN evhdlr**  
Event handler to be called [pmix\\_notification\\_fn\\_t](#) (function reference)
- IN cbfunc**  
Callback function [pmix\\_hdlr\\_req\\_cbfunc\\_t](#) (function reference)
- IN cbdata**  
Data to be passed to the *cbfunc* callback function (memory reference)

If `cbfunc` is **NULL**, the function call will be treated as a *blocking* call. In this case, the returned status will be either (a) the event handler reference identifier if the value is greater than or equal to zero, or (b) a negative error code indicative of the reason for the failure.

If the `cbfunc` is non-**NULL**, the function call will be treated as a *non-blocking* call and will return the following:

- **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API. The result of the registration operation shall be returned in the provided callback function along with the assigned event handler identifier.
  - **PMIX\_ERR\_EVENT\_REGISTRATION** indicating that the registration has failed for an undetermined reason.
  - a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed.

1      The callback function must not be executed prior to returning from the API, and no events  
2      corresponding to this registration may be delivered prior to the completion of the registration  
3      callback function (*cbfunc*).

## Required Attributes

4      The following attributes are required to be supported by all PMIx libraries:

5      **PMIX\_EVENT\_HDLR\_NAME** "pmix.evname" (**char\***)  
6          String name identifying this handler.

7      **PMIX\_EVENT\_HDLR\_FIRST** "pmix.evfist" (**bool**)  
8          Invoke this event handler before any other handlers.

9      **PMIX\_EVENT\_HDLR\_LAST** "pmix.evlst" (**bool**)  
10        Invoke this event handler after all other handlers have been called.

11     **PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY** "pmix.evfistcat" (**bool**)  
12        Invoke this event handler before any other handlers in this category.

13     **PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY** "pmix.evlstcat" (**bool**)  
14        Invoke this event handler after all other handlers in this category have been called.

15     **PMIX\_EVENT\_HDLR\_BEFORE** "pmix.evbefore" (**char\***)  
16        Put this event handler immediately before the one specified in the (**char\***) value.

17     **PMIX\_EVENT\_HDLR\_AFTER** "pmix.evafter" (**char\***)  
18        Put this event handler immediately after the one specified in the (**char\***) value.

19     **PMIX\_EVENT\_HDLR-prepend** "pmix.evprepend" (**bool**)  
20        Prepend this handler to the precedence list within its category.

21     **PMIX\_EVENT\_HDLR\_APPEND** "pmix.evappend" (**bool**)  
22        Append this handler to the precedence list within its category.

23     **PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrang" (**pmix\_data\_array\_t\***)  
24        Array of **pmix\_proc\_t** defining range of event notification.

25     **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)  
26        Define constraints on the processes that can access the provided data. Only processes that  
27        meet the constraints are allowed to access it.

28     **PMIX\_EVENT\_RETURN\_OBJECT** "pmix.evobject" (**void \***)  
29        Object to be returned whenever the registered callback function **cbfunc** is invoked. The  
30        object will only be returned to the process that registered it.

1  
2 Host environments that implement support for PMIx event notification are required to support the  
3 following attributes when registering handlers - these attributes are used to direct that the handler  
4 should be invoked only when the event affects the indicated process(es):

5 **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (**pmix\_proc\_t**)  
6 The single process that was affected.

7 **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaaffected" (**pmix\_data\_array\_t\***)  
8 Array of **pmix\_proc\_t** defining affected processes.  
▲-----▲

9 **Description**

10 Register an event handler to report events. Note that the codes being registered do *not* need to be  
11 PMIx error constants — any integer value can be registered. This allows for registration of  
12 non-PMIx events such as those defined by a particular SMS vendor or by an application itself.

▼-----Advice to users-----▼

13 In order to avoid potential conflicts, users are advised to only define codes that lie outside the range  
14 of the PMIx standard's error codes. Thus, SMS vendors and application developers should  
15 constrain their definitions to positive values or negative values beyond the  
16 **PMIX\_EXTERNAL\_ERR\_BASE** boundary.  
▲-----▲

▼-----Advice to users-----▼

17 As previously stated, upon completing its work, and prior to returning, each handler *must* call the  
18 event handler completion function provided when it was invoked (including a status code plus any  
19 information to be passed to later handlers) so that the chain can continue being progressed. An  
20 event handler can terminate all further progress along the chain by passing the  
21 **PMIX\_EVENT\_ACTION\_COMPLETE** status to the completion callback function. Note that the  
22 parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once  
23 the completion function has been called - thus, any information in the incoming parameters that  
24 will be referenced following the call to the completion function must be copied.  
▲-----▲

25 **9.1.3 Event registration constants**

26 **PMIX\_ERR\_EVENT\_REGISTRATION** Error in event registration.

## 1 9.1.4 System events

2   **PMIX\_EVENT\_SYS\_BASE**    Mark the beginning of a dedicated range of constants for system  
3    event reporting.  
4   **PMIX\_EVENT\_NODE\_DOWN**    A node has gone down - the identifier of the affected node will  
5    be included in the notification.  
6   **PMIX\_EVENT\_NODE\_OFFLINE**    A node has been marked as *offline* - the identifier of the  
7    affected node will be included in the notification.  
8   **PMIX\_EVENT\_SYS\_OTHER**    Mark the end of a dedicated range of constants for system event  
9    reporting.

### 10 Detect system event constant

11 Test a given event constant to see if it falls within the dedicated range of constants for system event  
12 reporting.

13   *PMIx v2.2*

C

14   **PMIX\_SYSTEM\_EVENT (a)**

C

15   **IN a**

16       Error constant to be checked ([pmix\\_status\\_t](#))

17       Returns **true** if the provided values falls within the dedicated range of events for system event  
reporting.

## 18 9.1.5 Event handler registration and notification attributes

19       Attributes to support event registration and notification.

20   **PMIX\_EVENT\_HDLR\_NAME "pmix.evname" (char\*)**  
21       String name identifying this handler.  
22   **PMIX\_EVENT\_HDLR\_FIRST "pmix.evfirst" (bool)**  
23       Invoke this event handler before any other handlers.  
24   **PMIX\_EVENT\_HDLR\_LAST "pmix.evlast" (bool)**  
25       Invoke this event handler after all other handlers have been called.  
26   **PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY "pmix.evfirstcat" (bool)**  
27       Invoke this event handler before any other handlers in this category.  
28   **PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY "pmix.evlastcat" (bool)**  
29       Invoke this event handler after all other handlers in this category have been called.  
30   **PMIX\_EVENT\_HDLR\_BEFORE "pmix.evbefore" (char\*)**  
31       Put this event handler immediately before the one specified in the **(char\*)** value.  
32   **PMIX\_EVENT\_HDLR\_AFTER "pmix.evafter" (char\*)**  
33       Put this event handler immediately after the one specified in the **(char\*)** value.  
34   **PMIX\_EVENT\_HDLR-prepend "pmix.evprepend" (bool)**  
35       Prepend this handler to the precedence list within its category.  
36   **PMIX\_EVENT\_HDLR\_APPEND "pmix.evappend" (bool)**

```

1      Append this handler to the precedence list within its category.
2  PMIX_EVENT_CUSTOM_RANGE "pmix.evrangle" (pmix_data_array_t*)
3      Array of pmix_proc_t defining range of event notification.
4  PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)
5      The single process that was affected.
6  PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)
7      Array of pmix_proc_t defining affected processes.
8  PMIX_EVENT_NON_DEFAULT "pmix.evnonddef" (bool)
9      Event is not to be delivered to default event handlers.
10 PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
11      Object to be returned whenever the registered callback function cbfunc is invoked. The
12      object will only be returned to the process that registered it.
13 PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)
14      Instruct the PMIx server not to cache the event.
15 PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)
16      PMIx server that sourced the event.
17 PMIX_EVENT_TEXT_MESSAGE "pmix.evtext" (char*)
18      Text message suitable for output by recipient - e.g., describing the cause of the event.
19 PMIX_EVENT_TIMESTAMP "pmix.evtstamp" (time_t)
20      System time when the associated event occurred.

```

### 9.1.5.1 Fault tolerance event attributes

The following attributes may be used by the host environment when providing an event notification as qualifiers indicating the action it intends to take in response to the event:

```

24 PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)
25      The RM intends to terminate this session.
26 PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)
27      The RM intends to terminate this job.
28 PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)
29      The RM intends to terminate all processes on this node.
30 PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool)
31      The RM intends to terminate just this process.
32 PMIX_EVENT_ACTION_TIMEOUT "pmix.evttimeout" (int)
33      The time in seconds before the RM will execute the indicated operation.

```

### 9.1.6 Notification Function

#### Summary

The **pmix\_notification\_fn\_t** is called by PMIx to deliver notification of an event.

## Advice to users

1 The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but  
2 different signature than the v2.0 Standard described below. The *ad hoc* v1.0 version was removed  
3 from the v2.0 Standard is not included in this document to avoid confusion.

PMIx v2.0

C

```
4     typedef void (*pmix_notification_fn_t)
5         (size_t evhdlr_registration_id,
6          pmix_status_t status,
7          const pmix_proc_t *source,
8          pmix_info_t info[], size_t ninfo,
9          pmix_info_t results[], size_t nresults,
10         pmix_event_notification_cbfunc_fn_t cbfunc,
11         void *cbdata);
```

C

12 **IN evhdlr\_registration\_id**

13 Registration number of the handler being called (**size\_t**)

14 **IN status**

15 Status associated with the operation (**pmix\_status\_t**)

16 **IN source**

17 Identifier of the process that generated the event (**pmix\_proc\_t**). If the source is the SMS,  
18 then the nspace will be empty and the rank will be PMIX\_RANK\_UNDEF

19 **IN info**

20 Information describing the event (**pmix\_info\_t**). This argument will be NULL if no  
21 additional information was provided by the event generator.

22 **IN ninfo**

23 Number of elements in the info array (**size\_t**)

24 **IN results**

25 Aggregated results from prior event handlers servicing this event (**pmix\_info\_t**). This  
26 argument will be **NULL** if this is the first handler servicing the event, or if no prior handlers  
27 provided results.

28 **IN nresults**

29 Number of elements in the results array (**size\_t**)

30 **IN cbfunc**

31 **pmix\_event\_notification\_cbfunc\_fn\_t** callback function to be executed upon  
32 completion of the handler's operation and prior to handler return (function reference).

33 **IN cbdata**

34 Callback data to be passed to cbfunc (memory reference)

1           **Description**

2       Note that different RMs may provide differing levels of support for event notification to application  
3       processes. Thus, the *info* array may be **NULL** or may contain detailed information of the event. It is  
4       the responsibility of the application to parse any provided info array for defined key-values if it so  
5       desires.

6            **Advice to users** 

7       Possible uses of the *info* array include:

- 8
  - for the host RM to alert the process as to planned actions, such as aborting the session, in  
9           response to the reported event
  - provide a timeout for alternative action to occur, such as for the application to request an  
10          alternate response to the event

11      For example, the RM might alert the application to the failure of a node that resulted in termination  
12      of several processes, and indicate that the overall session will be aborted unless the application  
13      requests an alternative behavior in the next 5 seconds. The application then has time to respond  
14      with a checkpoint request, or a request to recover from the failure by obtaining replacement nodes  
15      and restarting from some earlier checkpoint.

16      Support for these options is left to the discretion of the host RM. Info keys are included in the  
17      common definitions above but may be augmented by environment vendors.  


18            **Advice to PMIx server hosts** 

19      On the server side, the notification function is used to inform the PMIx server library's host of a  
20      detected event in the PMIx server library. Events generated by PMIx clients are communicated to  
21      the PMIx server library, but will be relayed to the host via the  
**pmix\_server\_notify\_event\_fn\_t** function pointer, if provided.  


22      **9.1.7 PMIx\_Deregister\_event\_handler**

23           **Summary**

24      Deregister an event handler.

## Format

C

## IN evhd1r ref

Event handler ID returned by registration (**size t**)

IN cbfunc

Callback function to be executed upon completion of operation `pmix_op_cbfunc_t` (function reference)

IN cbdata

Data to be passed to the cbfunc callback function (memory reference)

If `cbfunc` is **NULL**, the function will be treated as a *blocking* call and the result of the operation returned in the status code.

If `cbfunc` is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
  - **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

The returned status code will be one of the following:

- **PMIX\_SUCCESS** The event handler was successfully deregistered.
  - **PMIX\_ERR\_BAD\_PARAM** The provided *evhdlr\_ref* was unrecognized.
  - **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support event notification.

## Description

Deregister an event handler. Note that no events corresponding to the referenced registration may be delivered following completion of the deregistration operation (either return from the API with **PMIX\_OPERATION\_SUCCEEDED** or execution of the *cbfunc*).

## PMIx Notify event

## Summary

Report an event for notification via any registered event handler.

1           **Format**

2        `pmix_status_t`  
3        `PMIx_Notify_event(pmix_status_t status,`  
4                            `const pmix_proc_t *source,`  
5                            `pmix_data_range_t range,`  
6                            `pmix_info_t info[], size_t ninfo,`  
7                            `pmix_op_cbfunc_t cbfunc, void *cbdata);`

C

8        **IN    status**

9        Status code of the event (`pmix_status_t`)

10      **IN    source**

11      Pointer to a `pmix_proc_t` identifying the original reporter of the event (handle)

12      **IN    range**

13      Range across which this notification shall be delivered (`pmix_data_range_t`)

14      **IN    info**

15      Array of `pmix_info_t` structures containing any further info provided by the originator of  
16      the event (array of handles)

17      **IN    ninfo**

18      Number of elements in the *info* array (`size_t`)

19      **IN    cbfunc**

20      Callback function to be executed upon completion of operation `pmix_op_cbfunc_t`  
21      (function reference)

22      **IN    cbdata**

23      Data to be passed to the *cbfunc* callback function (memory reference)

24      If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation  
25      returned in the status code.

26      If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the  
27      following:

- **PMIX\_SUCCESS** The notification request is valid and is being processed. The callback function  
will be called when the process-local operation is complete and will provide the resulting status  
of that operation. Note that this does *not* reflect the success or failure of delivering the event to  
any recipients. The callback function must not be executed prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
returned *success* - the *cbfunc* will *not* be called
- **PMIX\_ERR\_BAD\_PARAM** The request contains at least one incorrect entry that prevents it from  
being processed. The callback function will *not* be called.

- **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support event notification, or in the case of a PMIx server calling the API, the range extended beyond the local node and the host SMS environment does not support event notification. The callback function will *not* be called.

## Required Attributes

The following attributes are required to be supported by all PMIx libraries:

**PMIX\_EVENT\_NON\_DEFAULT** "pmix.evnondef" (bool)

Event is not to be delivered to default event handlers.

**PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrangle" (pmix\_data\_array\_t\*)

Array of `pmix_proc_t` defining range of event notification.

**PMIX\_EVENT\_DO\_NOT\_CACHE** "pmix.evnocache" (bool)

Instruct the PMIx server not to cache the event.

**PMIX\_EVENT\_PROXY** "pmix.evproxy" (pmix\_proc\_t\*)

PMIx server that sourced the event.

**PMIX\_EVENT\_TEXT\_MESSAGE** "pmix.evtext" (char\*)

Text message suitable for output by recipient - e.g., describing the cause of the event.

Host environments that implement support for PMIx event notification are required to provide the following attributes for all events generated by the environment:

**PMIX EVENT AFFECTED PROC "pmix.evproc" (pmix proc t)**

The single process that was affected.

**PMIX EVENT AFFECTED PROCS** "pmix.evaaffected" (pmix data array t\*)

Array of `pmix_proc_t` defining affected processes.

## Optional Attributes

Host environments that support PMIx event notification may offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes may optionally be included to indicate the host environment's intended response to the event:

**PMIX\_EVENT\_TERMINATE\_SESSION** "pmix.evterm.sess" (bool)

The RM intends to terminate this session.

**PMIX\_EVENT\_TERMINATE\_JOB** "pmix.evterm.job" (bool)

The RM intends to terminate this job.

**PMIX\_EVENT\_TERMINATE\_NODE** "pmix.evterm.node" (bool)

The RM intends to terminate all processes on this node.

**PMIX\_EVENT\_TERMINATE\_PROC** "pmix.evterm.proc" (bool)

1        The RM intends to terminate just this process.

2        **PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evttimeout" (int)

3        The time in seconds before the RM will execute the indicated operation.



4        **Description**

5        Report an event for notification via any registered event handler. This function can be called by any  
6        PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server  
7        calls this API to report events it detected itself so that the host SMS daemon distribute and handle  
8        them, and to pass events given to it by its host down to any attached client processes for processing.  
9        Examples might include notification of the failure of another process, detection of an impending  
10      node failure due to rising temperatures, or an intent to preempt the application. Events may be  
11      locally generated or come from anywhere in the system.

12      Host SMS daemons call the API to pass events down to its embedded PMIx server both for  
13      transmittal to local client processes and for the host's own internal processing where the host has  
14      registered its own event handlers. The PMIx server library is not allowed to echo any event given to  
15      it by its host via this API back to the host through the **pmix\_server\_notify\_event\_fn\_t**  
16      server module function. The host is required to deliver the event to all PMIx servers where the  
17      targeted processes either are currently running, or (if they haven't started yet) might be running at  
18      some point in the future as the events are required to be cached by the PMIx server library.

19      Client application processes can call this function to notify the SMS and/or other application  
20      processes of an event it encountered. Note that processes are not constrained to report status values  
21      defined in the official PMIx standard — any integer value can be used. Thus, applications are free  
22      to define their own internal events and use the notification system for their own internal purposes.



Advice to users

23      The callback function will be called upon completion of the **notify\_event** function's actions.  
24      At that time, any messages required for executing the operation (e.g., to send the notification to the  
25      local PMIx server) will have been queued, but may not yet have been transmitted. The caller is  
26      required to maintain the input data until the callback function has been executed — the sole purpose  
27      of the callback function is to indicate when the input data is no longer required.



## 1 9.1.9 Notification Handler Completion Callback Function

### 2 Summary

3 The `pmix_event_notification_cbfunc_fn_t` is called by event handlers to indicate  
4 completion of their operations.

PMIx v2.0

```
5     typedef void (*pmix_event_notification_cbfunc_fn_t)
6         (pmix_status_t status,
7          pmix_info_t *results, size_t nresults,
8          pmix_op_cbfunc_t cbfunc, void *thiscbdata,
9          void *notification_cbdata);
```

C

C

#### 10 IN `status`

11 Status returned by the event handler's operation (`pmix_status_t`)

#### 12 IN `results`

13 Results from this event handler's operation on the event (`pmix_info_t`)

#### 14 IN `nresults`

15 Number of elements in the results array (`size_t`)

#### 16 IN `cbfunc`

17 `pmix_op_cbfunc_t` function to be executed when PMIx completes processing the  
18 callback (function reference)

#### 19 IN `thiscbdata`

20 Callback data that was passed in to the handler (memory reference)

#### 21 IN `cbdata`

22 Callback data to be returned when PMIx executes cbfunc (memory reference)

### 23 Description

24 Define a callback by which an event handler can notify the PMIx library that it has completed its  
25 response to the notification. The handler is *required* to execute this callback so the library can  
26 determine if additional handlers need to be called. The handler shall return

27 `PMIX_EVENT_ACTION_COMPLETE` if no further action is required. The return status of each  
28 event handler and any returned `pmix_info_t` structures will be added to the *results* array of  
29 `pmix_info_t` passed to any subsequent event handlers to help guide their operation.

30 If non-`NULL`, the provided callback function will be called to allow the event handler to release the  
31 provided info array and execute any other required cleanup operations.

## 32 9.1.9.1 Completion Callback Function Status Codes

33 The following status code may be returned indicating various actions taken by other event handlers.

34 `PMIX_EVENT_NO_ACTION_TAKEN` Event handler: No action taken.

35 `PMIX_EVENT_PARTIAL_ACTION_TAKEN` Event handler: Partial action taken.

36 `PMIX_EVENT_ACTION_DEFERRED` Event handler: Action deferred.

37 `PMIX_EVENT_ACTION_COMPLETE` Event handler: Action complete.

## CHAPTER 10

# Data Packing and Unpacking

1 PMIx intentionally does not include support for internode communications in the standard, instead  
2 relying on its host SMS environment to transfer any needed data and/or requests between nodes.  
3 These operations frequently involve PMIx-defined public data structures that include binary data.  
4 Many HPC clusters are homogeneous, and so transferring the structures can be done rather simply.  
5 However, greater effort is required in heterogeneous environments to ensure binary data is correctly  
6 transferred. PMIx buffer manipulation functions are provided for this purpose via standardized  
7 interfaces to ease adoption.

## 8 10.1 Data Buffer Type

9 The `pmix_data_buffer_t` structure describes a data buffer used for packing and unpacking.

PMIx v2.0

C

```
10     typedef struct pmix_data_buffer {
11         /** Start of my memory */
12         char *base_ptr;
13         /** Where the next data will be packed to
14             (within the allocated memory starting
15             at base_ptr) */
16         char *pack_ptr;
17         /** Where the next data will be unpacked
18             from (within the allocated memory
19             starting as base_ptr) */
20         char *unpack_ptr;
21         /** Number of bytes allocated (starting
22             at base_ptr) */
23         size_t bytes_allocated;
24         /** Number of bytes used by the buffer
25             (i.e., amount of data -- including
26             overhead -- packed in the buffer) */
27         size_t bytes_used;
28     } pmix_data_buffer_t;
```

C

## 10.2 Support Macros

2 PMIx provides a set of convenience macros for creating, initiating, and releasing data buffers.

3 **PMIX\_DATA\_BUFFER\_CREATE**

4 Allocate memory for a `pmix_data_buffer_t` object and initialize it. This macro uses `calloc` to  
5 allocate memory for the buffer and initialize all fields in it

PMIx v2.0

6 `PMIX_DATA_BUFFER_CREATE(buffer);`

C

C

7 **OUT buffer**

8 Variable to be assigned the pointer to the allocated `pmix_data_buffer_t` (handle)

9 **PMIX\_DATA\_BUFFER\_RELEASE**

10 Free a `pmix_data_buffer_t` object and the data it contains. Free's the data contained in the  
11 buffer, and then free's the buffer itself

PMIx v2.0

12 `PMIX_DATA_BUFFER_RELEASE(buffer);`

C

C

13 **IN buffer**

14 Pointer to the `pmix_data_buffer_t` to be released (handle)

15 **PMIX\_DATA\_BUFFER\_CONSTRUCT**

16 Initialize a statically declared `pmix_data_buffer_t` object.

PMIx v2.0

17 `PMIX_DATA_BUFFER_CONSTRUCT(buffer);`

C

C

18 **IN buffer**

19 Pointer to the allocated `pmix_data_buffer_t` that is to be initialized (handle)

20 **PMIX\_DATA\_BUFFER\_DESTRUCT**

21 Release the data contained in a `pmix_data_buffer_t` object.

PMIx v2.0

22 `PMIX_DATA_BUFFER_DESTRUCT(buffer);`

C

C

23 **IN buffer**

24 Pointer to the `pmix_data_buffer_t` whose data is to be released (handle)

```
1 PMIX_DATA_BUFFER_LOAD  
2 Load a blob into a pmix_data_buffer_t object. Load the given data into the provided  
3 pmix_data_buffer_t object, usually done in preparation for unpacking the provided data.  
4 Note that the data is not copied into the buffer - thus, the blob must not be released until after  
5 operations on the buffer have completed.
```

PMIx v2.0

C

```
6 PMIX_DATA_BUFFER_LOAD(buffer, data, size);
```

C

```
7 IN buffer  
8 Pointer to a pre-allocated pmix_data_buffer_t (handle)  
9 IN data  
10 Pointer to a blob (char*)  
11 IN size  
12 Number of bytes in the blob size_t
```

```
13 PMIX_DATA_BUFFER_UNLOAD
```

```
14 Unload the data from a pmix_data_buffer_t object. Extract the data in a buffer, assigning the  
15 pointer to the data (and the number of bytes in the blob) to the provided variables, usually done to  
16 transmit the blob to a remote process for unpacking. The buffer's internal pointer will be set to  
17 NULL to protect the data upon buffer destruct or release - thus, the user is responsible for releasing  
18 the blob when done with it.
```

PMIx v2.0

C

```
19 PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

C

```
20 IN buffer  
21 Pointer to the pmix_data_buffer_t whose data is to be extracted (handle)  
22 OUT data  
23 Variable to be assigned the pointer to the extracted blob (void*)  
24 OUT size  
25 Variable to be assigned the number of bytes in the blob size_t
```

## 26 10.3 General Routines

27 The following routines are provided to support internode transfers in heterogeneous environments.

### 28 10.3.1 PMIx\_Data\_pack

#### 29 Summary

30 Pack one or more values of a specified type into a buffer, usually for transmission to another process.

1           **Format**

2            pmix\_status\_t  
3            **PMIx\_Data\_pack**(const pmix\_proc\_t \*target,  
4                            pmix\_data\_buffer\_t \*buffer,  
5                            void \*src, int32\_t num\_vals,  
6                            pmix\_data\_type\_t type);

C

7           **IN target**

8           Pointer to a **pmix\_proc\_t** containing the nspace/rank of the process that will be unpacking  
9           the final buffer. A NULL value may be used to indicate that the target is based on the same  
10          PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

11          **IN buffer**

12          Pointer to a **pmix\_data\_buffer\_t** where the packed data is to be stored (handle)

13          **IN src**

14          Pointer to a location where the data resides. Strings are to be passed as (char \*\*)— i.e., the  
15          caller must pass the address of the pointer to the string as the (void\*). This allows the caller to  
16          pass multiple strings in a single call. (memory reference)

17          **IN num\_vals**

18          Number of elements pointed to by the *src* pointer. A string value is counted as a single value  
19          regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,  
20          string arrays) should be contiguous, although the data pointed to need not be contiguous  
21          across array entries. (**int32\_t**)

22          **IN type**

23          The type of the data to be packed (**pmix\_data\_type\_t**)

24          Returns one of the following:

25           **PMIX\_SUCCESS** The data has been packed as requested

26           **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.

27           **PMIX\_ERR\_BAD\_PARAM** The provided buffer or src is **NULL**

28           **PMIX\_ERR\_UNKNOWN\_DATA\_TYPE** The specified data type is not known to this  
29            implementation

30           **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation

31           **PMIX\_ERROR** General error

32          **Description**

33          The pack function packs one or more values of a specified type into the specified buffer. The buffer  
34          must have already been initialized via the **PMIX\_DATA\_BUFFER\_CREATE** or  
35          **PMIX\_DATA\_BUFFER\_CONSTRUCT** macros — otherwise, **PMIx\_Data\_pack** will return an  
36          error. Providing an unsupported type flag will likewise be reported as an error.

37          Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may  
38          lose precision when unpacked by a non-homogeneous recipient. The **PMIx\_Data\_pack** function

1 will do its best to deal with heterogeneity issues between the packer and unpacker in such cases.  
2 Sending a number larger than can be handled by the recipient will return an error code (generated  
3 upon unpacking) — the error cannot be detected during packing.

4 The namespace of the intended recipient of the packed buffer (i.e., the process that will be  
5 unpacking it) is used solely to resolve any data type differences between PMIx versions. The  
6 recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx  
7 library is aware of the version the recipient is using. Note that all processes in a given namespace  
8 are *required* to use the same PMIx version — thus, the caller must only know at least one process  
9 from the target’s namespace.

## 10 10.3.2 PMIx\_Data\_unpack

### 11 Summary

12 Unpack values from a [pmix\\_data\\_buffer\\_t](#)

### 13 Format

14 *PMIx v2.0*

15     [pmix\\_status\\_t](#)  
16     [PMIx\\_Data\\_unpack](#)([const pmix\\_proc\\_t](#) \*[source](#),  
17                         [pmix\\_data\\_buffer\\_t](#) \*[buffer](#), [void](#) \*[dest](#),  
18                         [int32\\_t](#) \*[max\\_num\\_values](#),  
19                         [pmix\\_data\\_type\\_t](#) [type](#));

20     **IN source**  
21         Pointer to a [pmix\\_proc\\_t](#) structure containing the nspace/rank of the process that packed  
22         the provided buffer. A NULL value may be used to indicate that the source is based on the  
23         same PMIx version as the caller. Note that only the source’s nspace is relevant. (handle)  
24     **IN buffer**  
25         A pointer to the buffer from which the value will be extracted. (handle)  
26     **INOUT dest**  
27         A pointer to the memory location into which the data is to be stored. Note that these values  
28         will be stored contiguously in memory. For strings, this pointer must be to ([char\\*\\*](#)) to provide  
29         a means of supporting multiple string operations. The unpack function will allocate memory  
30         for each string in the array - the caller must only provide adequate memory for the array of  
31         pointers. ([void\\*](#))  
32     **INOUT max\_num\_values**  
33         The number of values to be unpacked — upon completion, the parameter will be set to the  
34         actual number of values unpacked. In most cases, this should match the maximum number  
35         provided in the parameters — but in no case will it exceed the value of this parameter. Note  
36         that unpacking fewer values than are actually available will leave the buffer in an unpackable  
37         state — the function will return an error code to warn of this condition. ([int32\\_t](#))

1   **IN type**  
2   The type of the data to be unpacked — must be one of the PMIx defined data types  
3   ([pmix\\_data\\_type\\_t](#))  
4   Returns one of the following:  
5   **PMIX\_SUCCESS** The data has been unpacked as requested  
6   **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.  
7   **PMIX\_ERR\_BAD\_PARAM** The provided buffer or dest is **NULL**  
8   **PMIX\_ERR\_UNKNOWN\_DATA\_TYPE** The specified data type is not known to this  
9   implementation  
10   **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation  
11   **PMIX\_ERROR** General error

12   **Description**  
13   The unpack function unpacks the next value (or values) of a specified type from the given buffer.  
14   The buffer must have already been initialized via an **PMIX\_DATA\_BUFFER\_CREATE** or  
15   **PMIX\_DATA\_BUFFER\_CONSTRUCT** call (and assumedly filled with some data) — otherwise, the  
16   unpack\_value function will return an error. Providing an unsupported type flag will likewise be  
17   reported as an error, as will specifying a data type that *does not* match the type of the next item in  
18   the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an  
19   error.

20   Note that it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper  
21   variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte  
22   field in a string array that so happens to have a value that matches the specified data type flag).  
23   Therefore, the data type error check is *not* completely safe.

24   Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer.  
25   It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the  
26   unpack\_ptr.

27   Warning: The caller is responsible for providing adequate memory storage for the requested data.  
28   The user must provide a parameter indicating the maximum number of values that can be unpacked  
29   into the allocated memory. If more values exist in the buffer than can fit into the memory storage,  
30   then the function will unpack what it can fit into that location and return an error code indicating  
31   that the buffer was only partially unpacked.

32   Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may  
33   lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with  
34   heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than  
35   can be handled by the recipient will return an error code generated upon unpacking — these errors  
36   cannot be detected during packing.

37   The namespace of the process that packed the buffer is used solely to resolve any data type  
38   differences between PMIx versions. The packer must, therefore, be known to the user prior to  
39   calling the pack function so that the PMIx library is aware of the version the packer is using. Note

1 that all processes in a given namespace are *required* to use the same PMIx version — thus, the  
2 caller must only know at least one process from the packer's namespace.

### 3 10.3.3 PMIx\_Data\_copy

#### 4 Summary

5 Copy a data value from one location to another.

#### 6 Format

PMIx v2.0

```
7     pmix_status_t
8     PMIx_Data_copy(void **dest, void *src,
9                      pmix_data_type_t type);
```

10 IN dest

11 The address of a pointer into which the address of the resulting data is to be stored. (**void\*\***)

12 IN src

13 A pointer to the memory location from which the data is to be copied (handle)

14 IN type

15 The type of the data to be copied — must be one of the PMIx defined data types.

16 (**pmix\_data\_type\_t**)

17 Returns one of the following:

18 **PMIX\_SUCCESS** The data has been copied as requested

19 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.

20 **PMIX\_ERR\_BAD\_PARAM** The provided src or dest is **NULL**

21 **PMIX\_ERR\_UNKNOWN\_DATA\_TYPE** The specified data type is not known to this  
22 implementation

23 **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation

24 **PMIX\_ERROR** General error

#### 25 Description

26 Since registered data types can be complex structures, the system needs some way to know how to  
27 copy the data from one location to another (e.g., for storage in the registry). This function, which  
28 can call other copy functions to build up complex data types, defines the method for making a copy  
29 of the specified data type.

### 30 10.3.4 PMIx\_Data\_print

#### 31 Summary

32 Pretty-print a data value.

1           **Format**

2        `pmix_status_t`  
3        `PMIx_Data_print(char **output, char *prefix,`  
4                    `void *src, pmix_data_type_t type);`

C

C

5           **IN    output**

6        The address of a pointer into which the address of the resulting output is to be stored.  
7            (`char**`)

8           **IN    prefix**

9        String to be prepended to the resulting output (`char*`)

10          **IN    src**

11        A pointer to the memory location of the data value to be printed (handle)

12          **IN    type**

13        The type of the data value to be printed — must be one of the PMIx defined data types.  
14            (`pmix_data_type_t`)

15        Returns one of the following:

16            `PMIX_SUCCESS` The data has been printed as requested

17            `PMIX_ERR_BAD_PARAM` The provided data type is not recognized.

18            `PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

19           **Description**

20        Since registered data types can be complex structures, the system needs some way to know how to  
21        print them (i.e., convert them to a string representation). Primarily for debug purposes.

22      **10.3.5 PMIx\_Data\_copy\_payload**

23           **Summary**

24        Copy a payload from one buffer to another

25           **Format**

PMIx v2.0

```
1 pmix_status_t  
2 PMIx_Data_copy_payload(pmix_data_buffer_t *dest,  
3                            pmix_data_buffer_t *src);
```

4 **IN dest**

5     Pointer to the destination **pmix\_data\_buffer\_t** (handle)

6 **IN src**

7     Pointer to the source **pmix\_data\_buffer\_t** (handle)

8 Returns one of the following:

9     **PMIX\_SUCCESS** The data has been copied as requested

10    **PMIX\_ERR\_BAD\_PARAM** The src and dest **pmix\_data\_buffer\_t** types do not match

11    **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.

12 **Description**

13 This function will append a copy of the payload in one buffer into another buffer. Note that this is  
14 *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing  
15 payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

## CHAPTER 11

# Process Management

---

This chapter defines functionality processes can use to abort processes, spawn processes, and determine the relative locality of local processes.

### 11.1 Abort

PMIx provides a dedicated API by which an application can request that specified processes be aborted by the system.

#### 11.1.1 PMIx\_Abort

##### Summary

Abort the specified processes

##### Format

PMIx v1.0

```
pmix_status_t  
PMIx_Abort(int status, const char msg[],  
            pmix_proc_t procs[], size_t nprocs)
```

IN **status**  
Error code to return to invoking environment (integer)  
IN **msg**  
String message to be returned to user (string)  
IN **procs**  
Array of **pmix\_proc\_t** structures (array of handles)  
IN **nprocs**  
Number of elements in the *procs* array (integer)

Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

1           **Description**

2         Request that the host resource manager print the provided message and abort the provided array of  
3         *procs*. A Unix or POSIX environment should handle the provided status as a return error code from  
4         the main program that launched the application. A **NULL** for the *procs* array indicates that all  
5         processes in the caller's namespace are to be aborted, including itself. Passing a **NULL** *msg*  
6         parameter is allowed.

7         The function shall not return until the host environment has carried out the operation on the  
8         specified processes. If the caller is included in the array of targets, then the function will not return  
9         unless the host is unable to execute the operation.

10           **Advice to users**

11         The response to this request is somewhat dependent on the specific Resource Manager and its  
12         configuration (e.g., some resource managers will not abort the application if the provided status is  
13         zero unless specifically configured to do so, and some cannot abort subsets of processes in an  
14         application), and thus lies outside the control of PMIx itself. However, the PMIx client library shall  
15         inform the RM of the request that the specified *procs* be aborted, regardless of the value of the  
provided status.

16         Note that race conditions caused by multiple processes calling **PMIx\_Abort** are left to the server  
17         implementation to resolve with regard to which status is returned and what messages (if any) are  
18         printed.

19           **11.2 Process Creation**

20         The **PMIx\_Spawn** commands spawn new processes and/or applications in the PMIx universe.  
21         This may include requests to extend the existing resource allocation or obtain a new one, depending  
22         upon provided and supported attributes.

23           **11.2.1 PMIx\_Spawn**

24           **Summary**

25         Spawn a new job.

1           **Format**

2        **pmix\_status\_t**  
3        **PMIx\_Spawn**(const pmix\_info\_t job\_info[], size\_t ninfo,  
4                            const pmix\_app\_t apps[], size\_t napps,  
5                            char nspace[])

6   **IN    job\_info**  
7      Array of info structures (array of handles)  
8   **IN    ninfo**  
9      Number of elements in the *job\_info* array (integer)  
10   **IN    apps**  
11     Array of **pmix\_app\_t** structures (array of handles)  
12   **IN    napps**  
13     Number of elements in the *apps* array (integer)  
14   **OUT   nspace**  
15     Namespace of the new job (string)

16   Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

▼----- Required Attributes -----▼

17   PMIx libraries are not required to directly support any attributes for this function. However, any  
18   provided attributes must be passed to the host environment for processing.

19   Host environments are required to support the following attributes when present in either the  
20   *job\_info* or the *info* array of an element of the *apps* array:

21   **PMIX\_WDIR "pmix.wdir" (char\*)**  
22     Working directory for spawned processes.  
  
23   **PMIX\_SET\_SESSION\_CWD "pmix.ssncwd" (bool)**  
24     Set the current working directory to the session working directory assigned by the RM - can  
25     be assigned to the entire job (by including attribute in the *job\_info* array) or on a  
26     per-application basis in the *info* array for each **pmix\_app\_t**.  
  
27   **PMIX\_PREFIX "pmix.prefix" (char\*)**  
28     Prefix to use for starting spawned processes - i.e., the directory where the executables can be  
29     found.  
  
30   **PMIX\_HOST "pmix.host" (char\*)**  
31     Comma-delimited list of hosts to use for spawned processes.  
  
32   **PMIX\_HOSTFILE "pmix.hostfile" (char\*)**  
33     Hostfile to use for spawned processes.

## Optional Attributes

The following attributes are optional for host environments that support this operation:

1           **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (**char\***)  
2           Hostfile containing hosts to add to existing allocation.  
3  
4           **PMIX\_ADD\_HOST** "pmix.addhost" (**char\***)  
5           Comma-delimited list of hosts to add to the allocation.  
6           **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (**bool**)  
7           Preload executables onto nodes prior to executing launch procedure.  
8           **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (**char\***)  
9           Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.  
10          **PMIX\_PERSONALITY** "pmix.pers" (**char\***)  
11          Name of personality corresponding to programming model used by application - supported  
12          values depend upon PMIx implementation.  
13          **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (**bool**)  
14          Display process mapping upon spawn.  
15          **PMIX\_PPR** "pmix.ppr" (**char\***)  
16          Number of processes to spawn on each identified resource.  
17          **PMIX\_MAPBY** "pmix.mapby" (**char\***)  
18          Process mapping policy - when accessed using **PMIx\_Get**, use the  
19          **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
20          provided namespace. Supported values are launcher specific.  
21          **PMIX\_RANKBY** "pmix.rankby" (**char\***)  
22          Process ranking policy - when accessed using **PMIx\_Get**, use the  
23          **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
24          provided namespace. Supported values are launcher specific.  
25          **PMIX\_BINDTO** "pmix.bindto" (**char\***)  
26          Process binding policy - when accessed using **PMIx\_Get**, use the  
27          **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
28          provided namespace. Supported values are launcher specific.  
29          **PMIX\_STDIN\_TGT** "pmix.stdin" (**uint32\_t**)  
30          Spawned process rank that is to receive any forwarded **stdin**.  
31          **PMIX\_TAG\_OUTPUT** "pmix.tagout" (**bool**)  
32          Tag **stdout/stderr** with the identity of the source process - can be assigned to the entire  
33          job (by including attribute in the *job\_info* array) or on a per-application basis in the *info*  
34          array for each **pmix\_app\_t**.  
35          **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (**bool**)

1      Timestamp output - can be assigned to the entire job (by including attribute in the *job\_info*  
2      array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.

3      **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)

4      Merge **stdout** and **stderr** streams - can be assigned to the entire job (by including  
5      attribute in the *job\_info* array) or on a per-application basis in the *info* array for each  
6      **pmix\_app\_t**.

7      **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)

8      Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be  
9      assigned to the entire job (by including attribute in the *job\_info* array) or on a per-application  
10     basis in the *info* array for each **pmix\_app\_t**.

11     **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)

12     Mark the **argv** with the rank of the process.

13     **PMIX\_CPUS\_PER\_PROC** "pmix.cpuperproc" (uint32\_t)

14     Number of PUs to assign to each rank - when accessed using **PMIx\_Get**, use the  
15     **PMIX\_RANK\_WILDCARD** value for the rank to discover the PUs/process assigned to the  
16     provided namespace.

17     **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)

18     Do not place processes on the head node.

19     **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)

20     Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a  
21     node.

22     **PMIX\_REPORT\_BINDINGS** "pmix.repbind" (bool)

23     Report bindings of the individual processes.

24     **PMIX\_CPU\_LIST** "pmix.cpulist" (char\*)

25     List of PUs to use for this job - when accessed using **PMIx\_Get**, use the  
26     **PMIX\_RANK\_WILDCARD** value for the rank to discover the PU list used for the provided  
27     namespace.

28     **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)

29     Application supports recoverable operations.

30     **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)

31     Application is continuous, all failed processes should be immediately restarted.

32     **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)

33     Maximum number of times to restart a process - when accessed using **PMIx\_Get**, use the  
34     **PMIX\_RANK\_WILDCARD** value for the rank to discover the max restarts for the provided  
35     namespace.

36     **PMIX\_SET\_ENVAR** "pmix.environ.set" (pmix\_envar\_t\*)

37     Set the envar to the given value, overwriting any pre-existing one

```

1   PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
2     Unset the environment variable specified in the string.
3   PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
4     Add the environment variable, but do not overwrite any pre-existing one
5   PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
6     Prepend the given value to the specified environmental value using the given separator
7     character, creating the variable if it doesn't already exist
8   PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
9     Append the given value to the specified environmental value using the given separator
10    character, creating the variable if it doesn't already exist
11  PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
12    Ensure the given value appears first in the specified envar using the separator character,
13    creating the envar if it doesn't already exist
14  PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
15    Name of the WLM queue to which the allocation request is to be directed, or the queue being
16    referenced in a query.
17  PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
18    Total session time (in seconds) being requested in an allocation request.
19  PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
20    The number of nodes being requested in an allocation request.
21  PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
22    Regular expression of the specific nodes being requested in an allocation request.
23  PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
24    Number of PUs being requested in an allocation request.
25  PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
26    Regular expression of the number of PUs for each node being requested in an allocation
27    request.
28  PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
29    Regular expression of the specific PUs being requested in an allocation request.
30  PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
31    Number of Megabytes[base2] of memory (per process) being requested in an allocation
32    request.
33  PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
34    Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
35    request.
36  PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)

```

1                   Fabric quality of service level for the job being requested in an allocation request.  
2     **PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (**char\***)  
3       Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.  
4     **PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (**char\***)  
5       ID string for the *fabric plane* to be used for the requested allocation.  
6     **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (**size\_t**)  
7       Number of endpoints to allocate per *process* in the job.  
8     **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (**size\_t**)  
9       Number of endpoints to allocate per *node* for the job.



## 10                  **Description**

11      Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspace*  
12     parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the  
13     namespace returned. The *nspace* array must be at least of size one more than **PMIX\_MAX\_NSLEN**.

14      By default, the spawned processes will be PMIx "connected" to the parent process upon successful  
15     launch (see Section 11.3 for details). This includes that (a) the parent process will be given a copy  
16     of the new job's information so it can query job-level info without incurring any communication  
17     penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level  
18     info, and (c) both the parent process and members of the child job will receive notification of errors  
19     from processes in their combined assemblage.

### Advice to users

20      Behavior of individual resource managers may differ, but it is expected that failure of any  
21     application process to start will result in termination/cleanup of all processes in the newly spawned  
22     job and return of an error code to the caller.



### Advice to PMIx library implementers

23      Tools may utilize **PMIx\_Spawn** to start intermediate launchers as described in Section 17.2.2. For  
24     times where the tool is not attached to a PMIx server, internal support for fork/exec of the specified  
25     applications would allow the tool to maintain a single code path for both the connected and  
26     disconnected cases. Inclusion of such support is recommended, but not required.



## 27    **11.2.2 PMIx\_Spawn\_nb**

### 28    **Summary**

29    Nonblocking version of the **PMIx\_Spawn** routine.

1           **Format**

2            *pmix\_status\_t*  
3        **PMIx\_Spawn\_nb**(*const pmix\_info\_t job\_info[]*, *size\_t ninfo*,  
4            *const pmix\_app\_t apps[]*, *size\_t napps*,  
5            *pmix\_spawn\_cbfunc\_t cbfunc*, *void \*cbdata*)

C

C

6        **IN    job\_info**

7            Array of info structures (array of handles)

8        **IN    ninfo**

9            Number of elements in the *job\_info* array (integer)

10      **IN    apps**

11        Array of *pmix\_app\_t* structures (array of handles)

12      **IN    cbfunc**

13        Callback function *pmix\_spawn\_cbfunc\_t* (function reference)

14      **IN    cbdata**

15        Data to be passed to the callback function (memory reference)

16        Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the request - the *cbfunc* will *not* be called

▼----- **Required Attributes** -----▼

21        PMIx libraries are not required to directly support any attributes for this function. However, any  
22        provided attributes must be passed to the host SMS daemon for processing.

23        Host environments are required to support the following attributes when present in either the  
24        *job\_info* or the *info* array of an element of the *apps* array:

25        **PMIX\_WDIR "pmix.wdir" (char\*)**

26            Working directory for spawned processes.

27        **PMIX\_SET\_SESSION\_CWD "pmix.ssncwd" (bool)**

28            Set the current working directory to the session working directory assigned by the RM - can  
29            be assigned to the entire job (by including attribute in the *job\_info* array) or on a  
30            per-application basis in the *info* array for each *pmix\_app\_t*.

31        **PMIX\_PREFIX "pmix.prefix" (char\*)**

32            Prefix to use for starting spawned processes - i.e., the directory where the executables can be  
33            found.

34        **PMIX\_HOST "pmix.host" (char\*)**

1 Comma-delimited list of hosts to use for spawned processes.  
2 **PMIX\_HOSTFILE** "pmix.hostfile" (**char\***)  
3 Hostfile to use for spawned processes.

▲-----▲  
▼-----▼

### Optional Attributes

4 The following attributes are optional for host environments that support this operation:  
5 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (**char\***)  
6 Hostfile containing hosts to add to existing allocation.  
7 **PMIX\_ADD\_HOST** "pmix.addhost" (**char\***)  
8 Comma-delimited list of hosts to add to the allocation.  
9 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (**bool**)  
10 Preload executables onto nodes prior to executing launch procedure.  
11 **PMIX\_PRELOAD\_FILES** "pmix.loadfiles" (**char\***)  
12 Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.  
13 **PMIX\_PERSONALITY** "pmix.pers" (**char\***)  
14 Name of personality corresponding to programming model used by application - supported  
15 values depend upon PMIx implementation.  
16 **PMIX\_DISPLAY\_MAP** "pmix.dispmapping" (**bool**)  
17 Display process mapping upon spawn.  
18 **PMIX\_PPR** "pmix.ppr" (**char\***)  
19 Number of processes to spawn on each identified resource.  
20 **PMIX\_MAPBY** "pmix.mapby" (**char\***)  
21 Process mapping policy - when accessed using **PMIx\_Get**, use the  
22 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
23 provided namespace. Supported values are launcher specific.  
24 **PMIX\_RANKBY** "pmix.rankby" (**char\***)  
25 Process ranking policy - when accessed using **PMIx\_Get**, use the  
26 **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
27 provided namespace. Supported values are launcher specific.  
28 **PMIX\_BINDTO** "pmix.bindto" (**char\***)  
29 Process binding policy - when accessed using **PMIx\_Get**, use the  
30 **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
31 provided namespace. Supported values are launcher specific.  
32 **PMIX\_STDIN\_TGT** "pmix.stdin" (**uint32\_t**)  
33 Spawning process rank that is to receive any forwarded **stdin**.  
34 **PMIX\_TAG\_OUTPUT** "pmix.tagout" (**bool**)

1 Tag **stdout/stderr** with the identity of the source process - can be assigned to the entire  
2 job (by including attribute in the *job\_info* array) or on a per-application basis in the *info*  
3 array for each **pmix\_app\_t**.

4 **PMIX\_TIMESTAMP\_OUTPUT "pmix.tsout" (bool)**

5 Timestamp output - can be assigned to the entire job (by including attribute in the *job\_info*  
6 array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.

7 **PMIX\_MERGE\_STDERR\_STDOUT "pmix.mergeerrout" (bool)**

8 Merge **stdout** and **stderr** streams - can be assigned to the entire job (by including  
9 attribute in the *job\_info* array) or on a per-application basis in the *info* array for each  
10 **pmix\_app\_t**.

11 **PMIX\_OUTPUT\_TO\_FILE "pmix.outfile" (char\*)**

12 Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be  
13 assigned to the entire job (by including attribute in the *job\_info* array) or on a per-application  
14 basis in the *info* array for each **pmix\_app\_t**.

15 **PMIX\_INDEX\_ARGV "pmix.indxargv" (bool)**

16 Mark the **argv** with the rank of the process.

17 **PMIX\_CPU\_PER\_PROC "pmix.cpuperproc" (uint32\_t)**

18 Number of PUs to assign to each rank - when accessed using **PMIx\_Get**, use the  
19 **PMIX\_RANK\_WILDCARD** value for the rank to discover the PUs/process assigned to the  
20 provided namespace.

21 **PMIX\_NO\_PROCS\_ON\_HEAD "pmix.nolocal" (bool)**

22 Do not place processes on the head node.

23 **PMIX\_NO\_OVERSUBSCRIBE "pmix.noover" (bool)**

24 Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a  
25 node.

26 **PMIX\_REPORT\_BINDINGS "pmix.repbind" (bool)**

27 Report bindings of the individual processes.

28 **PMIX\_CPU\_LIST "pmix.cpulist" (char\*)**

29 List of PUs to use for this job - when accessed using **PMIx\_Get**, use the  
30 **PMIX\_RANK\_WILDCARD** value for the rank to discover the PU list used for the provided  
31 namespace.

32 **PMIX\_JOB\_RECOVERABLE "pmix.recover" (bool)**

33 Application supports recoverable operations.

34 **PMIX\_JOB\_CONTINUOUS "pmix.continuous" (bool)**

35 Application is continuous, all failed processes should be immediately restarted.

36 **PMIX\_MAX\_RESTARTS "pmix.maxrestarts" (uint32\_t)**

```

1 Maximum number of times to restart a process - when accessed using PMIx_Get, use the
2 PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
3 namespace.

4 PMIX_SET_ENVAR "pmix.environ.set" (pmix_envar_t*)
5 Set the envar to the given value, overwriting any pre-existing one

6 PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
7 Unset the environment variable specified in the string.

8 PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
9 Add the environment variable, but do not overwrite any pre-existing one

10 PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
11 Prepend the given value to the specified environmental value using the given separator
12 character, creating the variable if it doesn't already exist

13 PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
14 Append the given value to the specified environmental value using the given separator
15 character, creating the variable if it doesn't already exist

16 PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
17 Ensure the given value appears first in the specified envar using the separator character,
18 creating the envar if it doesn't already exist

19 PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
20 Name of the WLM queue to which the allocation request is to be directed, or the queue being
21 referenced in a query.

22 PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
23 Total session time (in seconds) being requested in an allocation request.

24 PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
25 The number of nodes being requested in an allocation request.

26 PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
27 Regular expression of the specific nodes being requested in an allocation request.

28 PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
29 Number of PUs being requested in an allocation request.

30 PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
31 Regular expression of the number of PUs for each node being requested in an allocation
32 request.

33 PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
34 Regular expression of the specific PUs being requested in an allocation request.

35 PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)

```

1           Number of Megabytes[base2] of memory (per process) being requested in an allocation  
2           request.

3       **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (**float**)  
4           Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
5           request.

6       **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (**char\***)  
7           Fabric quality of service level for the job being requested in an allocation request.

8       **PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (**char\***)  
9           Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

10      **PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (**char\***)  
11       ID string for the *fabric plane* to be used for the requested allocation.

12      **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (**size\_t**)  
13       Number of endpoints to allocate per *process* in the job.

14      **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (**size\_t**)  
15       Number of endpoints to allocate per *node* for the job.

## 16           **Description**

17           Nonblocking version of the **PMIx\_Spawn** routine. The provided callback function will be  
18           executed upon successful start of *all* specified application processes.

### Advice to users

19           Behavior of individual resource managers may differ, but it is expected that failure of any  
20           application process to start will result in termination/cleanup of all processes in the newly spawned  
21           job and return of an error code to the caller.

## 22    **11.2.3 Spawn-specific constants**

23           In addition to the generic error constants, the following spawn-specific error constants may be  
24           returned by the spawn APIs:

25      **PMIX\_ERR\_JOB\_ALLOC\_FAILED**     The job request could not be executed due to failure to  
26           obtain the specified allocation  
27      **PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE**    The specified application executable either  
28           could not be found, or lacks execution privileges.  
29      **PMIX\_ERR\_JOB\_NO\_EXE\_SPECIFIED**    The job request did not specify an executable.  
30      **PMIX\_ERR\_JOB\_FAILED\_TO\_MAP**     The launcher was unable to map the processes for the  
31           specified job request.  
32      **PMIX\_ERR\_JOB\_FAILED\_TO\_LAUNCH**   One or more processes in the job request failed to  
33           launch

## 11.2.4 Spawn attributes

Attributes used to describe **PMIx\_Spawn** behavior - they are values passed to the **PMIx\_Spawn** API and therefore are not accessed using the **PMIx\_Get** APIs when used in that context. However, some of the attributes defined in this section can be provided by the host environment for other purposes - e.g., the host might provide the **PMIX\_MAPPER** attribute in the job-related information so that an application can use **PMIx\_Get** to discover the layout algorithm used for determining process locations. Multi-use attributes and their respective access reference rank are denoted below.

```
PMIX_PERSONALITY "pmix.pers" (char*)
    Name of personality corresponding to programming model used by application - supported
    values depend upon PMIx implementation.

PMIX_HOST "pmix.host" (char*)
    Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE "pmix.hostfile" (char*)
    Hostfile to use for spawned processes.

PMIX_ADD_HOST "pmix.addhost" (char*)
    Comma-delimited list of hosts to add to the allocation.

PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)
    Hostfile containing hosts to add to existing allocation.

PMIX_PREFIX "pmix.prefix" (char*)
    Prefix to use for starting spawned processes - i.e., the directory where the executables can be
    found.

PMIX_WDIR "pmix.wdir" (char*)
    Working directory for spawned processes.

PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
    Display process mapping upon spawn.

PMIX_PPR "pmix.ppr" (char*)
    Number of processes to spawn on each identified resource.

PMIX_MAPBY "pmix.mapby" (char*)
    Process mapping policy - when accessed using PMIx_Get, use the
    PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
    provided namespace. Supported values are launcher specific.

PMIX_RANKBY "pmix.rankby" (char*)
    Process ranking policy - when accessed using PMIx_Get, use the
    PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the
    provided namespace. Supported values are launcher specific.

PMIX_BINDTO "pmix.bindto" (char*)
    Process binding policy - when accessed using PMIx_Get, use the
    PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the
    provided namespace. Supported values are launcher specific.

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
    Preload executables onto nodes prior to executing launch procedure.

PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
```

```

1      Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.
2      PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
3          Spawned process rank that is to receive any forwarded stdin.
4      PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
5          Set the current working directory to the session working directory assigned by the RM - can
6          be assigned to the entire job (by including attribute in the job_info array) or on a
7          per-application basis in the info array for each pmix_app_t.
8      PMIX_TAG_OUTPUT "pmix.tagout" (bool)
9          Tag stdout/stderr with the identity of the source process - can be assigned to the entire
10         job (by including attribute in the job_info array) or on a per-application basis in the info
11         array for each pmix_app_t.
12      PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
13         Timestamp output - can be assigned to the entire job (by including attribute in the job_info
14         array) or on a per-application basis in the info array for each pmix_app_t.
15      PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
16         Merge stdout and stderr streams - can be assigned to the entire job (by including
17         attribute in the job_info array) or on a per-application basis in the info array for each
18         pmix_app_t.
19      PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
20         Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be
21         assigned to the entire job (by including attribute in the job_info array) or on a per-application
22         basis in the info array for each pmix_app_t.
23      PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
24         Direct output into files of form "<directory>/<jobid>/rank.<rank>/
25             stdout [err] " - can be assigned to the entire job (by including attribute in the job_info
26             array) or on a per-application basis in the info array for each pmix_app_t.
27      PMIX_INDEX_ARGV "pmix.indxargv" (bool)
28         Mark the argv with the rank of the process.
29      PMIX_CPUUS_PER_PROC "pmix.cpuperproc" (uint32_t)
30         Number of PUs to assign to each rank - when accessed using PMIx_Get, use the
31             PMIX_RANK_WILDCARD value for the rank to discover the PUs/process assigned to the
32             provided namespace.
33      PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
34         Do not place processes on the head node.
35      PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
36         Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
37             node.
38      PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
39         Report bindings of the individual processes.
40      PMIX_CPU_LIST "pmix.cpulist" (char*)
41         List of PUs to use for this job - when accessed using PMIx_Get, use the
42             PMIX_RANK_WILDCARD value for the rank to discover the PU list used for the provided
43             namespace.

```

```

1   PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
2       Application supports recoverable operations.
3   PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
4       Application is continuous, all failed processes should be immediately restarted.
5   PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
6       Maximum number of times to restart a process - when accessed using PMIx_Get, use the
7       PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
8       namespace.
9   PMIX_SPAWN_TOOL "pmix.spwn.tool" (bool)
10      Indicate that the job being spawned is a tool.
11   PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)
12      Include process stacktraces in timeout report from a job.
13   PMIX_TIMEOUT_REPORT_STATE "pmix.tim.state" (bool)
14      Report process states in timeout report from a job.
15   PMIX_NOTIFY_JOB_EVENTS "pmix.note.jev" (bool)
16      Requests that the launcher generate the PMIX_EVENT_JOB_START,
17      PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events. Each event is to
18      include at least the namespace of the corresponding job and a PMIX_EVENT_TIMESTAMP
19      indicating the time the event occurred. Note that the requester must register for these
20      individual events, or capture and process them by registering a default event handler instead
21      of individual handlers and then process the events based on the returned status code.
22      Another common method is to register one event handler for all job-related events, with a
23      separate handler for non-job events - see PMIx_Register_event_handler for details.
24   PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)
25      Requests that the launcher generate the PMIX_EVENT_JOB_END event for normal or
26      abnormal termination of the spawned job. The event shall include the returned status code
27      (PMIX_JOB_TERM_STATUS) for the corresponding job; the identity (PMIX_PROCID)
28      and exit status (PMIX_EXIT_CODE) of the first failed process, if applicable; and a
29      PMIX_EVENT_TIMESTAMP indicating the time the termination occurred. Note that the
30      requester must register for the event or capture and process it within a default event handler.
31   PMIX_NOTIFY_PROC_TERMINATION "pmix.noteproc" (bool)
32      Requests that the launcher generate the PMIX_EVENT_PROC_TERMINATED event
33      whenever a process either normally or abnormally terminates.
34   PMIX_NOTIFY_PROC_ABNORMAL_TERMINATION "pmix.noteabproc" (bool)
35      Requests that the launcher generate the PMIX_EVENT_PROC_TERMINATED event only
36      when a process abnormally terminates.
37   PMIX_LOG_PROC_TERMINATION "pmix.logproc" (bool)
38      Requests that the launcher log the PMIX_EVENT_PROC_TERMINATED event whenever a
39      process either normally or abnormally terminates.
40   PMIX_LOG_PROC_ABNORMAL_TERMINATION "pmix.logabproc" (bool)
41      Requests that the launcher log the PMIX_EVENT_PROC_TERMINATED event only when a
42      process abnormally terminates.
43   PMIX_LOG_JOB_EVENTS "pmix.log.jev" (bool)

```

```

1 Requests that the launcher log the PMIX_EVENT_JOB_START,
2 PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events using PMIx_Log,
3 subject to the logging attributes of Section 12.4.3.
4 PMIX_LOG_COMPLETION "pmix.logcomp" (bool)
5 Requests that the launcher log the PMIX_EVENT_JOB_END event for normal or abnormal
6 termination of the spawned job using PMIx_Log, subject to the logging attributes of
7 Section 12.4.3. The event shall include the returned status code
8 (PMIX_JOB_TERM_STATUS) for the corresponding job; the identity (PMIX_PROCID)
9 and exit status (PMIX_EXIT_CODE) of the first failed process, if applicable; and a
10 PMIX_EVENT_TIMESTAMP indicating the time the termination occurred.
11 PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
12 Do not generate an event when this job normally terminates.

13 Attributes used to adjust remote environment variables prior to spawning the specified application
14 processes.

15 PMIX_SET_ENVAR "pmix.environ.set" (pmix_envar_t*)
16 Set the envar to the given value, overwriting any pre-existing one
17 PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
18 Unset the environment variable specified in the string.
19 PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
20 Add the environment variable, but do not overwrite any pre-existing one
21 PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
22 Prepend the given value to the specified environmental value using the given separator
23 character, creating the variable if it doesn't already exist
24 PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
25 Append the given value to the specified environmental value using the given separator
26 character, creating the variable if it doesn't already exist
27 PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
28 Ensure the given value appears first in the specified envar using the separator character,
29 creating the envar if it doesn't already exist

```

### 11.2.5 Application Structure

31 The **pmix\_app\_t** structure describes the application context for the **PMIx\_Spawn** and  
32 **PMIx\_Spawn\_nb** operations.

*PMIx v1.0*

```
1  typedef struct pmix_app {
2      /** Executable */
3      char *cmd;
4      /** Argument set, NULL terminated */
5      char **argv;
6      /** Environment set, NULL terminated */
7      char **env;
8      /** Current working directory */
9      char *cwd;
10     /** Maximum processes with this profile */
11     int maxprocs;
12     /** Array of info keys describing this application*/
13     pmix_info_t *info;
14     /** Number of info keys in 'info' array */
15     size_t ninfo;
16 } pmix_app_t;
```

### 11.2.5.1 App structure support macros

18      The following macros are provided to support the `pmix_app_t` structure.

#### 19      Initialize the app structure

20      Initialize the `pmix_app_t` fields

PMIx v1.0

```
21      PMIX_APP_CONSTRUCT (m)
```

22      **IN m**

23      Pointer to the structure to be initialized (pointer to `pmix_app_t`)

#### 24      Destruct the app structure

25      Destruct the `pmix_app_t` fields

PMIx v1.0

```
26      PMIX_APP_DESTRUCT (m)
```

27      **IN m**

28      Pointer to the structure to be destructed (pointer to `pmix_app_t`)

```

1      Create an app array
2      Allocate and initialize an array of pmix_app_t structures
3      PMIx v1.0   ▼———— C —————▶
4          PMIX_APP_CREATE (m, n)   ▲———— C —————▶
5          INOUT m           Address where the pointer to the array of pmix_app_t structures shall be stored (handle)
6          IN n             Number of structures to be allocated (size_t)
7
8      Free an app structure
9      Release a pmix_app_t structure
10     PMIx v4.0   ▼———— C —————▶
11     PMIX_APP_RELEASE (m)   ▲———— C —————▶
12     IN m           Pointer to a pmix_app_t structure (handle)
13
14      Free an app array
15      Release an array of pmix_app_t structures
16     PMIx v1.0   ▼———— C —————▶
17     PMIX_APP_FREE (m, n)   ▲———— C —————▶
18     IN m           Pointer to the array of pmix_app_t structures (handle)
19     IN n             Number of structures in the array (size_t)
20
21      Create the info array of application directives
22      Create an array of pmix_info_t structures for passing application-level directives, updating the
23      ninfo field of the pmix_app_t structure.
24     PMIx v2.2   ▼———— C —————▶
25     PMIX_APP_INFO_CREATE (m, n)   ▲———— C —————▶
26     IN m           Pointer to the pmix_app_t structure (handle)
27     IN n             Number of directives to be allocated (size_t)

```

### 11.2.5.2 Spawn Callback Function

#### Summary

The `pmix_spawn_cbfunc_t` is used on the PMIx client side by `PMIx_Spawn_nb` and on the PMIx server side by `pmix_server_spawn_fn_t`.

PMIx v1.0

```
5     typedef void (*pmix_spawn_cbfunc_t)
6         (pmix_status_t status,
7          pmix_nspace_t nspace, void *cbdata);
```

C

C

8 **IN status**  
9 Status associated with the operation (handle)

10 **IN nspace**  
11 Namespace string (`pmix_nspace_t`)

12 **IN cbdata**  
13 Callback data passed to original API call (memory reference)

#### Description

The callback will be executed upon launch of the specified applications in `PMIx_Spawn_nb`, or upon failure to launch any of them.

The `status` of the callback will indicate whether or not the spawn succeeded. The `nspace` of the spawned processes will be returned, along with any provided callback data. Note that the returned `nspace` value will not be protected upon return from the callback function, so the receiver must copy it if it needs to be retained.

## 11.3 Connecting and Disconnecting Processes

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of *connected* solely implies that the host environment should treat the failure of any process in the assemblage as a reportable event, taking action on the assemblage as if it were a single application. For example, if the environment defaults (in the absence of any application directives) to terminating an application upon failure of any process in that application, then the environment should terminate all processes in the connected assemblage upon failure of any member.

The host environment may choose to assign a new namespace to the connected assemblage and/or assign new ranks for its members for its own internal tracking purposes. However, it is not required to communicate such assignments to the participants (e.g., in response to an appropriate call to `PMIx_Query_info_nb`). The host environment is required to generate a `PMIX_ERR_PROC_TERM_WO_SYNC` event should any process in the assemblage terminate or call `PMIx_Finalize` without first *disconnecting* from the assemblage. If the job including the process is terminated as a result of that action, then the host environment is required to also generate the `PMIX_ERR_JOB_TERM_WO_SYNC` for all jobs that were terminated as a result.

## Advice to PMIx server hosts

The *connect* operation does not require the exchange of job-level information nor the inclusion of information posted by participating processes via [PMIx\\_Put](#). Indeed, the callback function utilized in `pmix_server_connect_fn_t` cannot pass information back into the PMIx server library. However, host environments are advised that collecting such information at the participating daemons represents an optimization opportunity as participating processes are likely to request such information after the connect operation completes.

## Advice to users

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation. While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

Neither the PMIx implementation nor host environment are required to provide any tracking support for the assemblage. Thus, the application is responsible for maintaining the membership list of the assemblage.

### 11.3.1 PMIx\_Connect

#### Summary

Connect namespaces.

1           **Format**

2        **pmix\_status\_t**  
3        **PMIx\_Connect**(const pmix\_proc\_t procs[], size\_t nprocs,  
4                            const pmix\_info\_t info[], size\_t ninfo)

C

C

- 5        **IN    procs**  
6        Array of proc structures (array of handles)  
7        **IN    nprocs**  
8        Number of elements in the *procs* array (integer)  
9        **IN    info**  
10      Array of info structures (array of handles)  
11      **IN    ninfo**  
12      Number of elements in the *info* array (integer)

13     Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

14           **Required Attributes**

15     PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

16           **Optional Attributes**

17     The following attributes are optional for PMIx implementations:

18       **PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)  
19        All *clones* of the calling process must participate in the collective operation.

20     The following attributes are optional for host environments that support this operation:

21       **PMIX\_TIMEOUT** "pmix.timeout" (int)  
22        Time in seconds before the specified operation should time out (zero indicating infinite) and  
23        return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
          caused by multiple layers (client, server, and host) simultaneously timing the operation.

1           **Description**

2       Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The  
3       function will return once all processes identified in *procs* have called either **PMIx\_Connect** or its  
4       non-blocking version, *and* the host environment has completed any supporting operations required  
5       to meet the terms of the PMIx definition of *connected* processes.

6       A process can only engage in one connect operation involving the identical *procs* array at a time.  
7       However, a process can be simultaneously engaged in multiple connect operations, each involving a  
8       different *procs* array.

9       As in the case of the **PMIx\_Fence** operation, the *info* array can be used to pass user-level  
10      directives regarding timeout constraints and other options available from the host RM.

11            **Advice to users** 

12       All processes engaged in a given **PMIx\_Connect** operation must provide the identical *procs* array  
13       as ordering of entries in the array and the method by which those processes are identified (e.g., use  
14       of **PMIX\_RANK\_WILDCARD** versus listing the individual processes) *may* impact the host  
environment's algorithm for uniquely identifying an operation.

15            **Advice to PMIx library implementers** 

16       **PMIx\_Connect** and its non-blocking form are both *collective* operations. Accordingly, the PMIx  
17       server library is required to aggregate participation by local clients, passing the request to the host  
environment once all local participants have executed the API.

18            **Advice to PMIx server hosts** 

19       The host will receive a single call for each collective operation. It is the responsibility of the host to  
20       identify the nodes containing participating processes, execute the collective across all participating  
nodes, and notify the local PMIx server library upon completion of the global collective.

21       **11.3.2 PMIx\_Connect\_nb**

22           **Summary**

23       Nonblocking **PMIx\_Connect\_nb** routine.

1      **Format**

C

```
2      pmix_status_t  
3      PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,  
4                    const pmix_info_t info[], size_t ninfo,  
5                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

6      **IN    procs**

7      Array of proc structures (array of handles)

8      **IN    nprocs**

9      Number of elements in the *procs* array (integer)

10     **IN    info**

11     Array of info structures (array of handles)

12     **IN    ninfo**

13     Number of elements in the *info* array (integer)

14     **IN    cbfunc**

15     Callback function **pmix\_op\_cbfunc\_t** (function reference)

16     **IN    cbdata**

17     Data to be passed to the callback function (memory reference)

18     Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

▼----- Required Attributes -----▼

26     PMIx libraries are not required to directly support any attributes for this function. However, any  
27     provided attributes must be passed to the host SMS daemon for processing.

## Optional Attributes

1 The following attributes are optional for PMIx implementations:

2 **PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)

3 All *clones* of the calling process must participate in the collective operation.

4 The following attributes are optional for host environments that support this operation:

5 **PMIX\_TIMEOUT** "pmix.timeout" (int)

6 Time in seconds before the specified operation should time out (zero indicating infinite) and  
7 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
8 caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

10 Nonblocking version of **PMIx\_Connect**. The callback function is called once all processes  
11 identified in *procs* have called either **PMIx\_Connect** or its non-blocking version, *and* the host  
12 environment has completed any supporting operations required to meet the terms of the PMIx  
13 definition of *connected* processes. See the advice provided in the description for **PMIx\_Connect**  
14 for more information.

### 11.3.3 PMIx\_Disconnect

#### Summary

Disconnect a previously connected set of processes.

#### Format

PMIx v1.0

```
19     pmix_status_t
20     PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
21                       const pmix_info_t info[], size_t ninfo);
```

C

C

#### IN procs

23 Array of proc structures (array of handles)

#### IN nprocs

25 Number of elements in the *procs* array (integer)

#### IN info

27 Array of info structures (array of handles)

#### IN ninfo

29 Number of elements in the *info* array (integer)

30 Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request was successfully executed

- the **PMIX\_ERR\_INVALID\_OPERATION** error indicating that the specified set of *procs* was not previously *connected* via a call to **PMIx\_Connect** or its non-blocking form.
  - a PMIx error constant indicating either an error in the input or that the request failed

## Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

## Optional Attributes

The following attributes are optional for PMIx implementations:

**PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)  
All *clones* of the calling process must participate in the collective operation.

The following attributes are optional for host environments that support this operation:

**PMIX\_TIMEOUT** "pmix.timeout" (int)  
Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

Disconnect a previously connected set of processes. The function will return once all processes identified in *procs* have called either **PMIx\_Disconnect** or its non-blocking version, *and* the host environment has completed any required supporting operations.

A process can only engage in one disconnect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple disconnect operations, each involving a different *procs* array.

As in the case of the [PMIx\\_Fence](#) operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

## Advice to users

All processes engaged in a given **PMIx\_Disconnect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX\_RANK\_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

## Advice to PMIx library implementers

**PMIx\_Disconnect** and its non-blocking form are both *collective* operations. Accordingly, the PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

## Advice to PMIx server hosts

The host will receive a single call for each collective operation. The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

### 11.3.4 PMIx\_Disconnect\_nb

#### Summary

Nonblocking **PMIx\_Disconnect** routine.

#### Format

PMIx v1.0

C

```
pmix_status_t  
PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,  
                    const pmix_info_t info[], size_t ninfo,  
                    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

IN **procs**  
Array of proc structures (array of handles)  
IN **nprocs**  
Number of elements in the *procs* array (integer)  
IN **info**  
Array of info structures (array of handles)  
IN **ninfo**  
Number of elements in the *info* array (integer)  
IN **cbfunc**  
Callback function **pmix\_op\_cbfunc\_t** (function reference)  
IN **cbdata**  
Data to be passed to the callback function (memory reference)

Returns one of the following:

- 1     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
2       will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
3       function prior to returning from the API.  
4     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
5       returned *success* - the *cbfunc* will *not* be called  
6     • a PMIx error constant indicating either an error in the input or that the request was immediately  
7       processed and failed - the *cbfunc* will *not* be called

### Required Attributes

8     PMIx libraries are not required to directly support any attributes for this function. However, any  
9       provided attributes must be passed to the host SMS daemon for processing.

### Optional Attributes

10    The following attributes are optional for PMIx implementations:

11    **PMIX\_ALL\_CLONES\_PARTICIPATE** "pmix.clone.part" (bool)  
12       All *clones* of the calling process must participate in the collective operation.

13    The following attributes are optional for host environments that support this operation:

14    **PMIX\_TIMEOUT** "pmix.timeout" (int)  
15       Time in seconds before the specified operation should time out (zero indicating infinite) and  
16       return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
17       caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

19    Nonblocking **PMIx\_Disconnect** routine. The callback function is called either:

- 20    • to return the **PMIX\_ERR\_INVALID\_OPERATION** error indicating that the specified set of  
21       *procs* was not previously *connected* via a call to **PMIx\_Connect** or its non-blocking form;  
22    • to return a PMIx error constant indicating that the operation failed; or  
23    • once all processes identified in *procs* have called either **PMIx\_Disconnect\_nb** or its  
24       blocking version, *and* the host environment has completed any required supporting operations.

25    See the advice provided in the description for **PMIx\_Disconnect** for more information.

## 11.4 Process Locality

The relative locality of processes is often used to optimize their interactions with the hardware and other processes. PMIx provides a means by which the host environment can communicate the locality of a given process using the `PMIx_server_generate_locality_string` to generate an abstracted representation of that value. This provides a human-readable format and allows the client to parse the locality string with a method of its choice that may differ from the one used by the server that generated it.

There are times, however, when relative locality and other PMIx-provided information doesn't include some element required by the application. In these instances, the application may need access to the full description of the local hardware topology. PMIx does not itself generate such descriptions - there are multiple third-party libraries that fulfill that role. Instead, PMIx offers an abstraction method by which users can obtain a pointer to the description. This transparently enables support for different methods of sharing the topology between the host environment (which may well have already generated it prior to local start of application processes) and the clients - e.g., through passing of a shared memory region.

### 11.4.1 `PMIx_Load_topology`

#### Summary

Load the local hardware topology description

#### Format

PMIx v4.0

```
pmix_status_t  
PMIx_Load_topology(pmix_topology_t *topo);
```

#### INOUT `topo`

Address of a `pmix_topology_t` structure where the topology information is to be loaded (handle)

Returns `PMIX_SUCCESS`, indicating that the *topo* was successfully loaded, or an appropriate PMIx error constant.

#### Description

Obtain a pointer to the topology description of the local node. If the *source* field of the provided `pmix_topology_t` is set, then the PMIx library must return a description from the specified implementation or else indicate that the implementation is not available by returning the `PMIX_ERR_NOT_SUPPORTED` error constant.

The returned pointer may point to a shared memory region or an actual instance of the topology description. In either case, the description shall be treated as a "read-only" object - attempts to modify the object are likely to fail and return an error. The PMIx library is responsible for performing any required cleanup when the client library finalizes.

## Advice to users

1 It is the responsibility of the user to ensure that the *topo* argument is properly initialized prior to  
2 calling this API, and to check the returned *source* to verify that the returned topology description is  
3 compatible with the user's code.

### 4 11.4.2 PMIx\_Get\_relative\_locality

#### 5 Summary

6 Get the relative locality of two local processes given their locality strings.

#### 7 Format

PMIx v4.0

```
8 pmix_status_t
9 PMIx_Get_relative_locality(const char *locality1,
10                      const char *locality2,
11                      pmix_locality_t *locality);
```

C

C

12 IN locality1

13 String returned by the [PMIx\\_server\\_generate\\_locality\\_string](#) API (handle)

14 IN locality2

15 String returned by the [PMIx\\_server\\_generate\\_locality\\_string](#) API (handle)

16 INOUT locality

17 Location where the relative locality bitmask is to be constructed (memory reference)

18 Returns [PMIX\\_SUCCESS](#), indicating that the *locality* was successfully loaded, or an appropriate  
19 PMIx error constant.

#### 20 Description

21 Parse the locality strings of two processes (as returned by [PMIx\\_Get](#) using the  
22 [PMIX\\_LOCALITY\\_STRING](#) key) and set the appropriate [pmix\\_locality\\_t](#) locality bits in  
23 the provided memory location.

### 24 11.4.2.1 Topology description

25 The [pmix\\_topology\\_t](#) structure contains a (case-insensitive) string identifying the source of  
26 the topology (e.g., "hwloc") and a pointer to the corresponding implementation-specific topology  
27 description.

PMIx v4.0

```
28     typedef struct pmix_topology {
29         char *source;
30         void *topology;
31     } pmix_topoology_t;
```

C

C

### 11.4.2.2 Initialize the topology description structure

2 Initialize the `pmix_topology_t` fields to `NULL`

PMIx v4.0

C

3 `PMIX_TOPOLOGY_CONSTRUCT (m)`

C

4 IN m

5 Pointer to the structure to be initialized (pointer to `pmix_topology_t`)

### 11.4.2.3 Relative locality of two processes

7 PMIx v4.0 The `pmix_locality_t` datatype is a `uint16_t` bitmask that defines the relative locality of  
8 two processes on a node. The following constants represent specific bits in the mask and can be  
9 used to test a locality value using standard bit-test methods.

10	<code>PMIX_LOCALITY_UNKNOWN</code>	All bits are set to zero, indicating that the relative locality of the
11		two processes is unknown
12	<code>PMIX_LOCALITY_NONLOCAL</code>	The two processes do not share any common locations
13	<code>PMIX_LOCALITY_SHARE_HWTHREAD</code>	The two processes share at least one hardware thread
14	<code>PMIX_LOCALITY_SHARE_CORE</code>	The two processes share at least one core
15	<code>PMIX_LOCALITY_SHARE_L1CACHE</code>	The two processes share at least an L1 cache
16	<code>PMIX_LOCALITY_SHARE_L2CACHE</code>	The two processes share at least an L2 cache
17	<code>PMIX_LOCALITY_SHARE_L3CACHE</code>	The two processes share at least an L3 cache
18	<code>PMIX_LOCALITY_SHARE_PACKAGE</code>	The two processes share at least a package
19	<code>PMIX_LOCALITY_SHARE_NUMA</code>	The two processes share at least one Non-Uniform
20		Memory Access (NUMA) region
21	<code>PMIX_LOCALITY_SHARE_NODE</code>	The two processes are executing on the same node

22 Implementers and vendors may choose to extend these definitions as needed to describe a particular  
23 system.

### 11.4.2.4 Locality keys

25 `PMIX_LOCALITY_STRING "pmix.locstr" (char*)`

26 String describing a process's bound location - referenced using the process's rank. The string  
27 is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon. The  
28 remainder of the string represents the corresponding locality as expressed by the underlying  
29 implementation. The entire string must be passed to `PMIx_Get_relative_locality`  
30 for processing. Note that hosts are only required to provide locality strings for local client  
31 processes - thus, a call to `PMIx_Get` for the locality string of a process that returns  
32 `PMIX_ERR_NOT_FOUND` indicates that the process is not executing on the same node.

### 11.4.3 PMIx\_Get\_cpuset

#### Summary

Get the PU binding bitmap from its string representation.

1           **Format**

2        pmix\_status\_t  
3        PMIx\_Get\_cpuset(const char \*cpuset\_string,  
4                            pmix\_cpuset\_t \*cpuset);

C

5           **IN cpuset\_string**

6           String returned by the [PMIx\\_server\\_generate\\_cpuset\\_string](#) API (handle)

7           **INOUT cpuset**

8           Address of an object where the bitmap is to be stored (memory reference)

9           Returns [PMIX\\_SUCCESS](#), indicating that the *cpuset* was successfully loaded, or an appropriate  
10          PMIx error constant.

11          **Description**

12          Parse the string representation of the binding bitmap (as returned by [PMIx\\_Get](#) using the  
13            [PMIX\\_CPUSET](#) key) and set the appropriate PU binding location information in the provided  
14          memory location.

## CHAPTER 12

# Job Management and Reporting

---

The job management APIs provide an application with the ability to orchestrate its operation in partnership with the SMS. Members of this category include the `PMIx_Allocation_request`, `PMIx_Job_control`, and `PMIx_Process_monitor` APIs.

## 12.1 Allocation Requests

This section defines functionality to request new allocations from the RM, and request modifications to existing allocations. These are primarily used in the following scenarios:

- *Evolving* applications that dynamically request and return resources as they execute.
- *Malleable* environments where the scheduler redirects resources away from executing applications for higher priority jobs or load balancing.
- *Resilient* applications that need to request replacement resources in the face of failures.
- *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time, but realizes that they underestimated their required time while executing.

PMIx attempts to address this range of use-cases with a flexible API.

### 12.1.1 PMIx\_Allocation\_request

#### Summary

Request an allocation operation from the host resource manager.

#### Format

*PMIx v3.0*

C

```
pmix_status_t  
PMIx_Allocation_request(pmix_alloc_directive_t directive,  
                         pmix_info_t info[], size_t ninfo,  
                         pmix_info_t *results[], size_t *nresults);
```

```

1 IN directive          Allocation directive (pmix_alloc_directive_t)
2 IN info              Array of pmix_info_t structures (array of handles)
3 IN ninfo             Number of elements in the info array (integer)
4 INOUT results        Address where a pointer to an array of pmix_info_t containing the results of the request
5                           can be returned (memory reference)
6 INOUT nresults       Address where the number of elements in results can be returned (handle)
7
8 Returns one of the following:
9
10   • PMIX_SUCCESS, indicating that the request was processed and returned success
11
12   • a PMIx error constant indicating either an error in the input or that the request was refused

```

### Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process making the request.

Host environments that implement support for this operation are required to support the following attributes:

```

21 PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
22   User-provided string identifier for this allocation request which can later be used to query
23   status of the request.
24
25 PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
26   The number of nodes being requested in an allocation request.
27
28 PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
29   Number of PUs being requested in an allocation request.
30
31 PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
32   Total session time (in seconds) being requested in an allocation request.

```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (**char\***)  
Regular expression of the specific nodes being requested in an allocation request.

**PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (**char\***)  
Regular expression of the number of PUs for each node being requested in an allocation request.

**PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (**char\***)  
Regular expression of the specific PUs being requested in an allocation request.

**PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (**float**)  
Number of Megabytes[base2] of memory (per process) being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC** "pmix.alloc.net" (**array**)  
Array of **pmix\_info\_t** describing requested fabric resources. This must include at least:  
**PMIX\_ALLOC\_FABRIC\_ID**, **PMIX\_ALLOC\_FABRIC\_TYPE**, and  
**PMIX\_ALLOC\_FABRIC\_ENDPTS**, plus whatever other descriptors are desired.

**PMIX\_ALLOC\_FABRIC\_ID** "pmix.alloc.netid" (**char\***)  
The key to be used when accessing this requested fabric allocation. The fabric allocation will be returned/stored as a **pmix\_data\_array\_t** of **pmix\_info\_t** whose first element is composed of this key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a Transmission Control Protocol (TCP) allocation might consist of a comma-delimited string of socket ranges such as "32000-32100,33005,38123-38146". Additional array entries will consist of any provided resource request directives, along with their assigned values. Examples include: **PMIX\_ALLOC\_FABRIC\_TYPE** - the type of resources provided; **PMIX\_ALLOC\_FABRIC\_PLANE** - if applicable, what plane the resources were assigned from; **PMIX\_ALLOC\_FABRIC\_QOS** - the assigned QoS; **PMIX\_ALLOC\_BANDWIDTH** - the allocated bandwidth; **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** - a security key for the requested fabric allocation. NOTE: the array contents may differ from those requested, especially if **PMIX\_INFO\_REQD** was not set in the request.

**PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (**float**)  
Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (**char\***)  
Fabric quality of service level for the job being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (**char\***)  
Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (**char\***)

1 ID string for the *fabric plane* to be used for the requested allocation.  
2 **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (**size\_t**)  
3 Number of endpoints to allocate per *process* in the job.  
4 **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (**size\_t**)  
5 Number of endpoints to allocate per *node* for the job.  
6 **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** "pmix.alloc.nsec" (**pmix\_byte\_object\_t**)  
7 Request that the allocation include a fabric security key for the spawned job.



8 **Description**

9 Request an allocation operation from the host resource manager. Several broad categories are  
10 envisioned, including the ability to:

- 11 • Request allocation of additional resources, including memory, bandwidth, and compute. This  
12 should be accomplished in a non-blocking manner so that the application can continue to  
13 progress while waiting for resources to become available. Note that the new allocation will be  
14 disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one  
15 allocation will not impact the other.
- 16 • Extend the reservation on currently allocated resources, subject to scheduling availability and  
17 priorities. This includes extending the time limit on current resources, and/or requesting  
18 additional resources be allocated to the requesting job. Any additional allocated resources will be  
19 considered as part of the current allocation, and thus will be released at the same time.
- 20 • Return no-longer-required resources to the scheduler. This includes the “loan” of resources back  
21 to the scheduler with a promise to return them upon subsequent request.

22 If successful, the returned results for a request for additional resources must include the host  
23 resource manager’s identifier (**PMIX\_ALLOC\_ID**) that the requester can use to specify the  
24 resources in, for example, a call to **PMIx\_Spawn**.

25 **12.1.2 PMIx\_Allocation\_request\_nb**

26 **Summary**

27 Request an allocation operation from the host resource manager.

## Format

```
pmix_status_t  
PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,  
                           pmix_info_t info[], size_t ninfo,  
                           pmix_info_cbfunc_t cbfunc, void *cbdata);
```

<b>IN</b>	<b>directive</b>	Allocation directive ( <a href="#">pmix_alloc_directive_t</a> )
<b>IN</b>	<b>info</b>	Array of <a href="#">pmix_info_t</a> structures (array of handles)
<b>IN</b>	<b>ninfo</b>	Number of elements in the <i>info</i> array (integer)
<b>IN</b>	<b>cbfunc</b>	Callback function <a href="#">pmix_info_cbfunc_t</a> (function reference)
<b>IN</b>	<b>cbdata</b>	Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
  - **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

## Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process making the request.

Host environments that implement support for this operation are required to support the following attributes:

**PMIX\_ALLOC\_REQ\_ID** "pmix.alloc.reqid" (char\*)

User-provided string identifier for this allocation request which can later be used to query status of the request.

**PMIX ALLOC NUM NODES** "pmix.alloc.nnodes" (uint64\_t)

The number of nodes being requested in an allocation request.

```
1 PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
2 Number of PUs being requested in an allocation request.
3 PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
4 Total session time (in seconds) being requested in an allocation request.
```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

```
5 PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
6 Regular expression of the specific nodes being requested in an allocation request.
7 PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
8 Regular expression of the number of PUs for each node being requested in an allocation
9 request.
10 PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
11 Regular expression of the specific PUs being requested in an allocation request.
12 PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
13 Number of Megabytes[base2] of memory (per process) being requested in an allocation
14 request.
15 PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
16 Array of pmix_info_t describing requested fabric resources. This must include at least:
17 PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
18 PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
19 PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
20 The key to be used when accessing this requested fabric allocation. The fabric allocation
21 will be returned/stored as a pmix_data_array_t of pmix_info_t whose first
22 element is composed of this key and the allocated resource description. The type of the
23 included value depends upon the fabric support. For example, a TCP allocation might
24 consist of a comma-delimited string of socket ranges such as "32000-32100,
25 33005,38123-38146". Additional array entries will consist of any provided resource
26 request directives, along with their assigned values. Examples include:
27 PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
28 PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
29 from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
30 the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
31 requested fabric allocation. NOTE: the array contents may differ from those requested,
32 especially if PMIX_INFO_REQD was not set in the request.
33 PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
34 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
35 request.
```

```

1   PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
2     Fabric quality of service level for the job being requested in an allocation request.
3
4   PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
5     Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
6
7   PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
8     ID string for the fabric plane to be used for the requested allocation.
9
10  PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
11    Number of endpoints to allocate per process in the job.
12
13  PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
14    Number of endpoints to allocate per node for the job.
15
16  PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
17    Request that the allocation include a fabric security key for the spawned job.

```



### 13 Description

14 Non-blocking form of the **PMIx\_Allocation\_request** API.

## 15 12.1.3 Job Allocation attributes

16 Attributes used to describe the job allocation - these are values passed to and/or returned by the  
17 **PMIx\_Allocation\_request\_nb** and **PMIx\_Allocation\_request** APIs and are not  
18 accessed using the **PMIx\_Get** API.

```

19   PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
20     User-provided string identifier for this allocation request which can later be used to query
21     status of the request.
22   PMIX_ALLOC_ID "pmix.alloc.id" (char*)
23     A string identifier (provided by the host environment) for the resulting allocation which can
24     later be used to reference the allocated resources in, for example, a call to PMIx_Spawn.
25   PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
26     Name of the WLM queue to which the allocation request is to be directed, or the queue being
27     referenced in a query.
28   PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
29     The number of nodes being requested in an allocation request.
30   PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
31     Regular expression of the specific nodes being requested in an allocation request.
32   PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
33     Number of PUs being requested in an allocation request.
34   PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
35     Regular expression of the number of PUs for each node being requested in an allocation
36     request.

```

```

1   PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
2     Regular expression of the specific PUs being requested in an allocation request.
3   PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
4     Number of Megabytes[base2] of memory (per process) being requested in an allocation
5     request.
6   PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
7     Array of pmix_info_t describing requested fabric resources. This must include at least:
8       PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
9       PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
10  PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
11    The key to be used when accessing this requested fabric allocation. The fabric allocation
12    will be returned/stored as a pmix_data_array_t of pmix_info_t whose first
13    element is composed of this key and the allocated resource description. The type of the
14    included value depends upon the fabric support. For example, a TCP allocation might
15    consist of a comma-delimited string of socket ranges such as "32000-32100,
16    33005, 38123-38146". Additional array entries will consist of any provided resource
17    request directives, along with their assigned values. Examples include:
18      PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
19      PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
20      from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
21      the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
22      requested fabric allocation. NOTE: the array contents may differ from those requested,
23      especially if PMIX_INFO REQD was not set in the request.
24  PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
25    Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
26    request.
27  PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
28    Fabric quality of service level for the job being requested in an allocation request.
29  PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
30    Total session time (in seconds) being requested in an allocation request.
31  PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
32    Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
33  PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
34    ID string for the fabric plane to be used for the requested allocation.
35  PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
36    Number of endpoints to allocate per process in the job.
37  PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
38    Number of endpoints to allocate per node for the job.
39  PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
40    Request that the allocation include a fabric security key for the spawned job.

```

## 1 12.1.4 Job Allocation Directives

2 *PMIx v2.0* The `pmix_alloc_directive_t` structure is a `uint8_t` type that defines the behavior of  
3 allocation requests. The following constants can be used to set a variable of the type  
4 `pmix_alloc_directive_t`. All definitions were introduced in version 2 of the standard  
5 unless otherwise marked.

6 **PMIX\_ALLOC\_NEW** A new allocation is being requested. The resulting allocation will be  
7 disjoint (i.e., not connected in a job sense) from the requesting allocation.

8 **PMIX\_ALLOC\_EXTEND** Extend the existing allocation, either in time or as additional  
9 resources.

10 **PMIX\_ALLOC\_RELEASE** Release part of the existing allocation. Attributes in the  
11 accompanying `pmix_info_t` array may be used to specify permanent release of the  
12 identified resources, or “lending” of those resources for some period of time.

13 **PMIX\_ALLOC\_REAQUIRE** Reacquire resources that were previously “lent” back to the  
14 scheduler.

15 **PMIX\_ALLOC\_EXTERNAL** A value boundary above which implementers are free to define  
16 their own directive values.

## 17 12.2 Job Control

18 This section defines APIs that enable the application and host environment to coordinate the  
19 response to failures and other events. This can include requesting termination of the entire job or a  
20 subset of processes within a job, but can also be used in combination with other PMIx capabilities  
21 (e.g., allocation support and event notification) for more nuanced responses. For example, an  
22 application notified of an incipient over-temperature condition on a node could use the  
23 `PMIx_Allocation_request_nb` interface to request replacement nodes while  
24 simultaneously using the `PMIx_Job_control_nb` interface to direct that a checkpoint event be  
25 delivered to all processes in the application. If replacement resources are not available, the  
26 application might use the `PMIx_Job_control_nb` interface to request that the job continue at a  
27 lower power setting, perhaps sufficient to avoid the over-temperature failure.

28 The job control APIs can also be used by an application to register itself as available for preemption  
29 when operating in an environment such as a cloud or where incentives, financial or otherwise, are  
30 provided to jobs willing to be preempted. Registration can include attributes indicating how many  
31 resources are being offered for preemption (e.g., all or only some portion), whether the application  
32 will require time to prepare for preemption, etc. Jobs that request a warning will receive an event  
33 notifying them of an impending preemption (possibly including information as to the resources that  
34 will be taken away, how much time the application will be given prior to being preempted, whether  
35 the preemption will be a suspension or full termination, etc.) so they have an opportunity to save  
36 their work. Once the application is ready, it calls the provided event completion callback function to  
37 indicate that the SMS is free to suspend or terminate it, and can include directives regarding any  
38 desired restart.

## 12.2.1 PMIx\_Job\_control

### 2 Summary

3 Request a job control action.

### 4 Format

PMIx v3.0

```
5     pmix_status_t
6     PMIx_Job_Control(const pmix_proc_t targets[], size_t ntargs,
7                         const pmix_info_t directives[], size_t ndirs,
8                         pmix_info_t *results[], size_t *nresults);
```

#### 9 IN targets

10 Array of proc structures (array of handles)

#### 11 IN ntargs

12 Number of elements in the *targets* array (integer)

#### 13 IN directives

14 Array of info structures (array of handles)

#### 15 IN ndirs

16 Number of elements in the *directives* array (integer)

#### 17 INOUT results

18 Address where a pointer to an array of [pmix\\_info\\_t](#) containing the results of the request  
19 can be returned (memory reference)

#### 20 INOUT nresults

21 Address where the number of elements in *results* can be returned (handle)

22 Returns one of the following:

- 23 • [PMIX\\_SUCCESS](#), indicating that the request was processed by the host environment and  
24 returned *success*. Details of the result will be returned in the *results* array
- 25 • a PMIx error constant indicating either an error in the input or that the request was refused

### Required Attributes

26 PMIx libraries are not required to directly support any attributes for this function. However, any  
27 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
28 required to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process making  
29 the request.

30 Host environments that implement support for this operation are required to support the following  
31 attributes:

32 [PMIX\\_JOB\\_CTRL\\_ID](#) "pmix.jctrl.id" (char\*)

```

1      Provide a string identifier for this request. The user can provide an identifier for the
2      requested operation, thus allowing them to later request status of the operation or to
3      terminate it. The host, therefore, shall track it with the request for future reference.
4      PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
5          Pause the specified processes.
6      PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
7          Resume ("un-pause") the specified processes.
8      PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
9          Forcibly terminate the specified processes and cleanup.
10     PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
11     Send given signal to specified processes.
12     PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
13     Politely terminate the specified processes.
14     PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
15     Comma-delimited list of files to be removed upon process termination.
16     PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)
17     Comma-delimited list of directories to be removed upon process termination.
18     PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)
19     Recursively cleanup all subdirectories under the specified one(s).
20     PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)
21     Only remove empty subdirectories.
22     PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)
23     Comma-delimited list of filenames that are not to be removed.
24     PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
25     When recursively cleaning subdirectories, do not remove the top-level directory (the one
26     given in the cleanup request).

```

### Optional Attributes

The following attributes are optional for host environments that support this operation:

```

28     PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
29     Cancel the specified request - the provided request ID must match the
30     PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control. An ID of
31     NULL implies cancel all requests from this requestor.
32     PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
33     Restart the specified processes using the given checkpoint ID.

```

```

1   PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
2     Checkpoint the specified processes and assign the given ID to it.
3   PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
4     Use event notification to trigger a process checkpoint.
5   PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
6     Use the given signal to trigger a process checkpoint.
7   PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
8     Time in seconds to wait for a checkpoint to complete.
9   PMIX_JOB_CTRL_CHECKPOINT_METHOD
10  "pmix.jctrl.ckmethod" (pmix_data_array_t)
11  Array of pmix_info_t declaring each method and value supported by this application.
12  PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
13  Regular expression identifying nodes that are to be provisioned.
14  PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
15  Name of the image that is to be provisioned.
16  PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
17  Indicate that the job can be pre-empted.

```



## 18 Description

19 Request a job control action. The *targets* array identifies the processes to which the requested job  
20 control action is to be applied. All *clones* of an identified process are to have the requested action  
21 applied to them. A **NULL** value can be used to indicate all processes in the caller's namespace. The  
22 use of **PMIX\_RANK\_WILDCARD** can also be used to indicate that all processes in the given  
23 namespace are to be included.

24 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The returned  
25 *status* indicates whether or not the request was granted, and information as to the reason for any  
26 denial of the request shall be returned in the *results* array.

## 27 12.2.2 PMIx\_Job\_control\_nb

### 28 Summary

29 Request a job control action.

1           **Format**

C

```
2       pmix_status_t  
3       PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargs,  
4                            const pmix_info_t directives[], size_t ndirs,  
5                            pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

6       **IN    targets**

7       Array of proc structures (array of handles)

8       **IN    ntargs**

9       Number of elements in the *targets* array (integer)

10      **IN    directives**

11      Array of info structures (array of handles)

12      **IN    ndirs**

13      Number of elements in the *directives* array (integer)

14      **IN    cbfunc**

15      Callback function **pmix\_info\_cbfunc\_t** (function reference)

16      **IN    cbdata**

17      Data to be passed to the callback function (memory reference)

18      Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

▼----- Required Attributes -----▼

26      PMIx libraries are not required to directly support any attributes for this function. However, any  
27      provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
28      *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process making  
29      the request.

30      Host environments that implement support for this operation are required to support the following  
31      attributes:

32      **PMIX\_JOB\_CTRL\_ID "pmix.jctrl.id" (char\*)**

33          Provide a string identifier for this request. The user can provide an identifier for the  
34          requested operation, thus allowing them to later request status of the operation or to  
35          terminate it. The host, therefore, shall track it with the request for future reference.

```

1   PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
2     Pause the specified processes.

3   PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
4     Resume ("un-pause") the specified processes.

5   PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
6     Forcibly terminate the specified processes and cleanup.

7   PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
8     Send given signal to specified processes.

9   PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
10    Politely terminate the specified processes.

11  PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
12    Comma-delimited list of files to be removed upon process termination.

13  PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)
14    Comma-delimited list of directories to be removed upon process termination.

15  PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)
16    Recursively cleanup all subdirectories under the specified one(s).

17  PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)
18    Only remove empty subdirectories.

19  PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)
20    Comma-delimited list of filenames that are not to be removed.

21  PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
22    When recursively cleaning subdirectories, do not remove the top-level directory (the one
23    given in the cleanup request).

```



### Optional Attributes

The following attributes are optional for host environments that support this operation:

```

25  PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
26    Cancel the specified request - the provided request ID must match the
27    PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control. An ID of
28    NULL implies cancel all requests from this requestor.

29  PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
30    Restart the specified processes using the given checkpoint ID.

31  PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
32    Checkpoint the specified processes and assign the given ID to it.

33  PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)

```

```
1      Use event notification to trigger a process checkpoint.  
2      PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)  
3          Use the given signal to trigger a process checkpoint.  
4      PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)  
5          Time in seconds to wait for a checkpoint to complete.  
6      PMIX_JOB_CTRL_CHECKPOINT_METHOD  
7      "pmix.jctrl.ckmethod" (pmix_data_array_t)  
8          Array of pmix_info_t declaring each method and value supported by this application.  
9      PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)  
10         Regular expression identifying nodes that are to be provisioned.  
11     PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)  
12         Name of the image that is to be provisioned.  
13     PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)  
14         Indicate that the job can be pre-empted.
```

## 15 **Description**

16 Non-blocking form of the **PMIx\_Job\_control** API. The *targets* array identifies the processes to  
17 which the requested job control action is to be applied. All *clones* of an identified process are to  
18 have the requested action applied to them. A **NULL** value can be used to indicate all processes in  
19 the caller's namespace. The use of **PMIX\_RANK\_WILDCARD** can also be used to indicate that all  
20 processes in the given namespace are to be included.

21 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The callback  
22 function provides a *status* to indicate whether or not the request was granted, and to provide some  
23 information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
24 **pmix\_info\_t** structures.

## 25 **12.2.3 Job control constants**

26 The following constants are specifically defined for return by the job control APIs:

27 **PMIX\_ERR\_CONFLICTING\_CLEANUP\_DIRECTIVES** Conflicting directives given for  
28 job/process cleanup.

## 12.2.4 Job control events

The following job control events may be available for registration, depending upon implementation and host environment support:

**PMIX\_JCTRL\_CHECKPOINT** Monitored by PMIx client to trigger a checkpoint operation.  
**PMIX\_JCTRL\_CHECKPOINT\_COMPLETE** Sent by a PMIx client and monitored by a PMIx server to notify that requested checkpoint operation has completed.  
**PMIX\_JCTRL\_PREEMPT\_ALERT** Monitored by a PMIx client to detect that an RM intends to preempt the job.  
**PMIX\_ERR\_PROC\_RESTART** Error in process restart.  
**PMIX\_ERR\_PROC\_CHECKPOINT** Error in process checkpoint.  
**PMIX\_ERR\_PROC\_MIGRATE** Error in process migration.

## 12.2.5 Job control attributes

Attributes used to request control operations on an executing application - these are values passed to the job control APIs and are not accessed using the [PMIx\\_Get](#) API.

**PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (**char\***)  
Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.  
**PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (**bool**)  
Pause the specified processes.  
**PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (**bool**)  
Resume ("un-pause") the specified processes.  
**PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (**char\***)  
Cancel the specified request - the provided request ID must match the **PMIX\_JOB\_CTRL\_ID** provided to a previous call to [PMIx\\_Job\\_control](#). An ID of **NULL** implies cancel all requests from this requestor.  
**PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (**bool**)  
Forcibly terminate the specified processes and cleanup.  
**PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (**char\***)  
Restart the specified processes using the given checkpoint ID.  
**PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (**char\***)  
Checkpoint the specified processes and assign the given ID to it.  
**PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (**bool**)  
Use event notification to trigger a process checkpoint.  
**PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (**int**)  
Use the given signal to trigger a process checkpoint.  
**PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (**int**)  
Time in seconds to wait for a checkpoint to complete.  
**PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
"pmix.jctrl.ckmethod" (**pmix\_data\_array\_t**)

```

1      Array of pmix_info_t declaring each method and value supported by this application.
2      PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
3          Send given signal to specified processes.
4      PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
5          Regular expression identifying nodes that are to be provisioned.
6      PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
7          Name of the image that is to be provisioned.
8      PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
9          Indicate that the job can be pre-empted.
10     PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
11     Polite terminate the specified processes.
12     PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
13     Comma-delimited list of files to be removed upon process termination.
14     PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)
15     Comma-delimited list of directories to be removed upon process termination.
16     PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)
17     Recursively cleanup all subdirectories under the specified one(s).
18     PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)
19     Only remove empty subdirectories.
20     PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)
21     Comma-delimited list of filenames that are not to be removed.
22     PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
23     When recursively cleaning subdirectories, do not remove the top-level directory (the one
24     given in the cleanup request).

```

## 12.3 Process and Job Monitoring

In addition to external faults, a common problem encountered in HPC applications is a failure to make progress due to some internal conflict in the computation. These situations can result in a significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the job. Various watchdog methods have been developed for detecting this situation, including requiring a periodic “heartbeat” from the application and monitoring a specified file for changes in size and/or modification time.

The following APIs allow applications to request monitoring, directing what is to be monitored, the frequency of the associated check, whether or not the application is to be notified (via the event notification subsystem) of stall detection, and other characteristics of the operation.

### 12.3.1 PMIx\_Process\_monitor

#### Summary

Request that application processes be monitored.

## Format

```
1 PMIx v3.0
2 pmix_status_t
3 PMIx_Process_monitor(const pmix_info_t *monitor,
4                         pmix_status_t error,
5                         const pmix_info_t directives[], size_t ndirs,
6                         pmix_info_t *results[], size_t *nresults);
```

7     **IN monitor**  
8         info (handle)  
9     **IN error**  
10        status (integer)  
11     **IN directives**  
12        Array of info structures (array of handles)  
13     **IN ndirs**  
14        Number of elements in the *directives* array (integer)  
15     **INOUT results**  
16        Address where a pointer to an array of **pmix\_info\_t** containing the results of the request  
17        can be returned (memory reference)  
18     **INOUT nresults**  
19        Address where the number of elements in *results* can be returned (handle)  
20        Returns one of the following:  
21           • **PMIX\_SUCCESS**, indicating that the request was processed and returned *success*. Details of the  
22            result will be returned in the *results* array  
23           • a PMIx error constant indicating either an error in the input or that the request was refused

## Optional Attributes

24       The following attributes may be implemented by a PMIx library or by the host environment. If  
25       supported by the PMIx server library, then the library must not pass the supported attributes to the  
26       host environment. All attributes not directly supported by the server library must be passed to the  
27       host environment if it supports this operation, and the library is *required* to add the  
28       **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the requesting process:

```
29 PMIX_MONITOR_ID "pmix.monitor.id" (char*)
30       Provide a string identifier for this request.

31 PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)
32       Identifier to be canceled (NULL means cancel all monitoring for this process).

33 PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)
```

```

1   The application desires to control the response to a monitoring event - i.e., the application is
2   requesting that the host environment not take immediate action in response to the event (e.g.,
3   terminating the job).
4   PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)
5       Register to have the PMIx server monitor the requestor for heartbeats.
6   PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
7       Time in seconds before declaring heartbeat missed.
8   PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
9       Number of heartbeats that can be missed before generating the event.
10  PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
11      Register to monitor file for signs of life.
12  PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
13      Monitor size of given file is growing to determine if the application is running.
14  PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
15      Monitor time since last access of given file to determine if the application is running.
16  PMIX_MONITOR_FILE MODIFY "pmix.monitor.fmod" (char*)
17      Monitor time since last modified of given file to determine if the application is running.
18  PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
19      Time in seconds between checking the file.
20  PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
21      Number of file checks that can be missed before generating the event.

```



## 22      Description

23      Request that application processes be monitored via several possible methods. For example, that  
24      the server monitor this process for periodic heartbeats as an indication that the process has not  
25      become “wedged”. When a monitor detects the specified alarm condition, it will generate an event  
26      notification using the provided error code and passing along any available relevant information. It  
27      is up to the caller to register a corresponding event handler.

28      The *monitor* argument is an attribute indicating the type of monitor being requested. For example,  
29      **PMIX\_MONITOR\_FILE** to indicate that the requestor is asking that a file be monitored.

30      The *error* argument is the status code to be used when generating an event notification alerting that  
31      the monitor has been triggered. The range of the notification defaults to  
32      **PMIX\_RANGE\_NAMESPACE**. This can be changed by providing a **PMIX\_RANGE** directive.

33      The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency  
34      of checking to be done

1 The returned *status* indicates whether or not the request was granted, and information as to the  
2 reason for any denial of the request shall be returned in the *results* array.

### 3 12.3.2 PMIx\_Process\_monitor\_nb

#### 4 Summary

5 Request that application processes be monitored.

#### 6 Format

PMIx v2.0

```
7     pmix_status_t
8     PMIx_Process_monitor_nb(const pmix_info_t *monitor,
9                             pmix_status_t error,
10                            const pmix_info_t directives[],
11                            size_t ndirs,
12                            pmix_info_cbfunc_t cbfunc, void *cbdata);
```

13 **IN monitor**  
14 info (handle)  
15 **IN error**  
16 status (integer)  
17 **IN directives**  
18 Array of info structures (array of handles)  
19 **IN ndirs**  
20 Number of elements in the *directives* array (integer)  
21 **IN cbfunc**  
22 Callback function [pmix\\_info\\_cbfunc\\_t](#) (function reference)  
23 **IN cbdata**  
24 Data to be passed to the callback function (memory reference)

25 Returns one of the following:

- 26 • [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
27 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
28 function prior to returning from the API.
- 29 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
30 returned *success* - the *cbfunc* will *not* be called.
- 31 • a PMIx error constant indicating either an error in the input or that the request was immediately  
32 processed and failed - the *cbfunc* will *not* be called.

## Optional Attributes

1 The following attributes may be implemented by a PMIx library or by the host environment. If  
2 supported by the PMIx server library, then the library must not pass the supported attributes to the  
3 host environment. All attributes not directly supported by the server library must be passed to the  
4 host environment if it supports this operation, and the library is *required* to add the  
5 **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the requesting process:

6     **PMIX\_MONITOR\_ID** "pmix.monitor.id" (**char\***)

7         Provide a string identifier for this request.

8     **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (**char\***)

9         Identifier to be canceled (**NULL** means cancel all monitoring for this process).

10    **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.apctrl" (**bool**)

11         The application desires to control the response to a monitoring event - i.e., the application is  
12         requesting that the host environment not take immediate action in response to the event (e.g.,  
13         terminating the job).

14    **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (**void**)

15         Register to have the PMIx server monitor the requestor for heartbeats.

16    **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (**uint32\_t**)

17         Time in seconds before declaring heartbeat missed.

18    **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (**uint32\_t**)

19         Number of heartbeats that can be missed before generating the event.

20    **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (**char\***)

21         Register to monitor file for signs of life.

22    **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (**bool**)

23         Monitor size of given file is growing to determine if the application is running.

24    **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (**char\***)

25         Monitor time since last access of given file to determine if the application is running.

26    **PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (**char\***)

27         Monitor time since last modified of given file to determine if the application is running.

28    **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (**uint32\_t**)

29         Time in seconds between checking the file.

30    **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (**uint32\_t**)

31         Number of file checks that can be missed before generating the event.

1           **Description**

2       Non-blocking form of the [PMIx\\_Process\\_monitor](#) API. The *cfunc* function provides a  
3       *status* to indicate whether or not the request was granted, and to provide some information as to the  
4       reason for any denial in the [pmix\\_info\\_cbfunc\\_t](#) array of [pmix\\_info\\_t](#) structures.

5           **12.3.3 PMIx\_Heartbeat**

6           **Summary**

7       Send a heartbeat to the PMIx server library

8           **Format**

PMIx v2.0

9       **PMIx\_Heartbeat () ;**

C

C

10           **Description**

11       A simplified macro wrapping [PMIx\\_Process\\_monitor\\_nb](#) that sends a heartbeat to the PMIx  
12       server library.

13           **12.3.4 Monitoring events**

14       The following monitoring events may be available for registration, depending upon implementation  
15       and host environment support:

16       **PMIX\_MONITOR\_HEARTBEAT\_ALERT**     Heartbeat failed to arrive within specified window.  
17           The process that triggered this alert will be identified in the event.

18       **PMIX\_MONITOR\_FILE\_ALERT**     File failed its monitoring detection criteria. The file that  
19           triggered this alert will be identified in the event.

20           **12.3.5 Monitoring attributes**

21       Attributes used to control monitoring of an executing application- these are values passed to the  
22       [PMIx\\_Process\\_monitor\\_nb](#) API and are not accessed using the [PMIx\\_Get](#) API.

23       **PMIX\_MONITOR\_ID "pmix.monitor.id" (char\*)**

24           Provide a string identifier for this request.

25       **PMIX\_MONITOR\_CANCEL "pmix.monitor.cancel" (char\*)**

26           Identifier to be canceled (**NULL** means cancel all monitoring for this process).

27       **PMIX\_MONITOR\_APP\_CONTROL "pmix.monitor.appctrl" (bool)**

28           The application desires to control the response to a monitoring event - i.e., the application is  
29           requesting that the host environment not take immediate action in response to the event (e.g.,  
30           terminating the job).

31       **PMIX\_MONITOR\_HEARTBEAT "pmix.monitor.mbeat" (void)**

32           Register to have the PMIx server monitor the requestor for heartbeats.

```

1   PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)
2       Send heartbeat to local PMIx server.
3   PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
4       Time in seconds before declaring heartbeat missed.
5   PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
6       Number of heartbeats that can be missed before generating the event.
7   PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
8       Register to monitor file for signs of life.
9   PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
10      Monitor size of given file is growing to determine if the application is running.
11   PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
12      Monitor time since last access of given file to determine if the application is running.
13   PMIX_MONITOR_FILE MODIFY "pmix.monitor.fmod" (char*)
14      Monitor time since last modified of given file to determine if the application is running.
15   PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
16      Time in seconds between checking the file.
17   PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
18      Number of file checks that can be missed before generating the event.

```

## 12.4 Logging

20 The logging interface supports posting information by applications and SMS elements to persistent  
21 storage. This function is *not* intended for output of computational results, but rather for reporting  
22 status and saving state information such as inserting computation progress reports into the  
23 application's SMS job log or error reports to the local syslog.

### 12.4.1 PMIx\_Log

#### Summary

Log data to a data service.

#### Format

*PMIx v3.0*

```

28     pmix_status_t
29     PMIx_Log(const pmix_info_t data[], size_t ndata,
30                 const pmix_info_t directives[], size_t ndirs);

```

31 **IN data**  
32 Array of info structures (array of handles)  
33 **IN ndata**  
34 Number of elements in the *data* array (**size\_t**)

```
1   IN  directives
2     Array of info structures (array of handles)
3   IN  ndirs
4     Number of elements in the directives array (size_t)
5   Return codes are one of the following:
6     PMIX_SUCCESS The logging request was successful.
7     PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry.
8     PMIX_ERR_NOT_SUPPORTED The PMIx implementation or host environment does not support
9       this function.
10    other appropriate PMIx error code
```

### Required Attributes

If the PMIx library does not itself perform this operation, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

```
14   PMIX_USERID "pmix.euid" (uint32_t)
15     Effective user ID of the connecting process.
16   PMIX_GRPID "pmix.egid" (uint32_t)
17     Effective group ID of the connecting process.
```

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

```
20   PMIX_LOG_STDERR "pmix.log.stderr" (char*)
21     Log string to stderr.
22   PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
23     Log string to stdout.
24   PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
25     Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
26     otherwise to local syslog.
27   PMIX_LOG_LOCAL_SYSLOG "pmix.log.1sys" (char*)
28     Log data to local syslog. Defaults to ERROR priority.
29   PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
30     Forward data to system “gateway” and log msg to that syslog. Defaults to ERROR priority.
31   PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)
32     Syslog priority level.
33   PMIX_LOG_ONCE "pmix.log.once" (bool)
34     Only log this once with whichever channel can first support it, taking the channels in priority
35     order.
```



## Optional Attributes

1 The following attributes are optional for host environments or PMIx libraries that support this  
2 operation:

```
3 PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
4 ID of source of the log request.

5 PMIX_LOG_TIMESTAMP "pmix.log.tstamp" (time_t)
6 Timestamp for log report.

7 PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstamp" (bool)
8 Generate timestamp for log.

9 PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
10 Label the output stream with the channel name (e.g., "stdout").

11 PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
12 Print timestamp in output string.

13 PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
14 Print the output stream in eXtensible Markup Language (XML) format.

15 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
16 Log via email based on pmix_info_t containing directives.

17 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
18 Comma-delimited list of email addresses that are to receive the message.

19 PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
20 Subject line for email.

21 PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
22 Message to be included in email.

23 PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
24 Log the provided information to the host environment's job record.

25 PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
26 Store the log data in a global data store (e.g., database).
```



1           **Description**  
2         Log data subject to the services offered by the host environment. The data to be logged is provided  
3         in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel.

4           **Advice to users**  
5  
6  
7  
8

It is strongly recommended that the **PMIx\_Log** API not be used by applications for streaming data as it is not a “performant” transport and can perturb the application since it involves the local PMIx server and host SMS daemon. Note that a return of **PMIX\_SUCCESS** only denotes that the data was successfully handed to the appropriate system call (for local channels) or the host environment and does not indicate receipt at the final destination.

9           **12.4.2 PMIx\_Log\_nb**  
10  
11

12           **Summary**  
13         Log data to a data service.  
14  
15

16           **Format**  
17         PMIx v2.0  
18  
19         pmix\_status\_t  
20         **PMIx\_Log\_nb**(const pmix\_info\_t data[], size\_t ndata,  
21                            const pmix\_info\_t directives[], size\_t ndirs,  
22                            pmix\_op\_cbfunc\_t cbfunc, void \*cbdata);  
23  
24

25           **IN data**  
26         Array of info structures (array of handles)  
27           **IN ndata**  
28         Number of elements in the *data* array (**size\_t**)  
29           **IN directives**  
30         Array of info structures (array of handles)  
31           **IN ndirs**  
32         Number of elements in the *directives* array (**size\_t**)  
33           **IN cbfunc**  
34         Callback function **pmix\_op\_cbfunc\_t** (function reference)  
35           **IN cbdata**  
36         Data to be passed to the callback function (memory reference)

37         Return codes are one of the following:

38         **PMIX\_SUCCESS** The logging request is valid and is being processed. The resulting status from  
39         the operation will be provided in the callback function. Note that the library must not invoke  
40         the callback function prior to returning from the API.

1       **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
2                  returned success - the *cbfunc* will *not* be called  
3       **PMIX\_ERR\_BAD\_PARAM** The logging request contains at least one incorrect entry that prevents  
4                  it from being processed. The callback function will not be called.  
5       **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function. The  
6                  callback function will not be called.  
7       other appropriate PMIx error code - the callback function will not be called.

## Required Attributes

8 If the PMIx library does not itself perform this operation, then it is required to pass any attributes  
9 provided by the client to the host environment for processing. In addition, it must include the  
10 following attributes in the passed *info* array:

11      **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
12                  Effective user ID of the connecting process.

13      **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
14                  Effective group ID of the connecting process.

15 Host environments or PMIx libraries that implement support for this operation are required to  
16 support the following attributes:

17      **PMIX\_LOG\_STDERR** "pmix.log.stderr" (**char\***)  
18                  Log string to **stderr**.

19      **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (**char\***)  
20                  Log string to **stdout**.

21      **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (**char\***)  
22                  Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
23                  otherwise to local syslog.

24      **PMIX\_LOG\_LOCAL\_SYSLOG** "pmix.log.lsys" (**char\***)  
25                  Log data to local syslog. Defaults to **ERROR** priority.

26      **PMIX\_LOG\_GLOBAL\_SYSLOG** "pmix.log.gsys" (**char\***)  
27                  Forward data to system “gateway” and log msg to that syslog. Defaults to **ERROR** priority.

28      **PMIX\_LOG\_SYSLOG\_PRI** "pmix.log.syspri" (**int**)  
29                  Syslog priority level.

30      **PMIX\_LOG\_ONCE** "pmix.log.once" (**bool**)  
31                  Only log this once with whichever channel can first support it, taking the channels in priority  
32                  order.

## Optional Attributes

1 The following attributes are optional for host environments or PMIx libraries that support this  
2 operation:

```
3 PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
4 ID of source of the log request.

5 PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
6 Timestamp for log report.

7 PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)
8 Generate timestamp for log.

9 PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
10 Label the output stream with the channel name (e.g., "stdout").

11 PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
12 Print timestamp in output string.

13 PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
14 Print the output stream in XML format.

15 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
16 Log via email based on pmix_info_t containing directives.

17 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
18 Comma-delimited list of email addresses that are to receive the message.

19 PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
20 Subject line for email.

21 PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
22 Message to be included in email.

23 PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
24 Log the provided information to the host environment's job record.

25 PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
26 Store the log data in a global data store (e.g., database).
```

1           **Description**  
2         Log data subject to the services offered by the host environment. The data to be logged is provided  
3         in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The  
4         callback function will be executed when the log operation has been completed. The *data* and  
5         *directives* arrays must be maintained until the callback is provided.

6           **Advice to users**  
7  
8  
9  
10

It is strongly recommended that the **PMIx\_Log\_nb** API not be used by applications for streaming  
data as it is not a “performant” transport and can perturb the application since it involves the local  
PMIx server and host SMS daemon. Note that a return of **PMIX\_SUCCESS** only denotes that the  
data was successfully handed to the appropriate system call (for local channels) or the host  
environment and does not indicate receipt at the final destination.

11          **12.4.3 Log attributes**  
12  
13

Attributes used to describe **PMIx\_Log** behavior - these are values passed to the **PMIx\_Log** API  
and therefore are not accessed using the **PMIx\_Get** API.

14          **PMIX\_LOG\_SOURCE** "pmix.log.source" (pmix\_proc\_t\*)  
15            ID of source of the log request.  
16          **PMIX\_LOG\_STDERR** "pmix.log.stderr" (char\*)  
17            Log string to **stderr**.  
18          **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (char\*)  
19            Log string to **stdout**.  
20          **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (char\*)  
21            Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
22            otherwise to local syslog.  
23          **PMIX\_LOG\_LOCAL\_SYSLOG** "pmix.log.lsys" (char\*)  
24            Log data to local syslog. Defaults to **ERROR** priority.  
25          **PMIX\_LOG\_GLOBAL\_SYSLOG** "pmix.log.gsys" (char\*)  
26            Forward data to system “gateway” and log msg to that syslog. Defaults to **ERROR** priority.  
27          **PMIX\_LOG\_SYSLOG\_PRI** "pmix.log.syspri" (int)  
28            Syslog priority level.  
29          **PMIX\_LOG\_TIMESTAMP** "pmix.log.tstmp" (time\_t)  
30            Timestamp for log report.  
31          **PMIX\_LOG\_GENERATE\_TIMESTAMP** "pmix.log.gtstamp" (bool)  
32            Generate timestamp for log.  
33          **PMIX\_LOG\_TAG\_OUTPUT** "pmix.log.tag" (bool)  
34            Label the output stream with the channel name (e.g., “stdout”).  
35          **PMIX\_LOG\_TIMESTAMP\_OUTPUT** "pmix.log.tsout" (bool)  
36            Print timestamp in output string.  
37          **PMIX\_LOG\_XML\_OUTPUT** "pmix.log.xml" (bool)

```
1      Print the output stream in XML format.  
2 PMIX_LOG_ONCE "pmix.log.once" (bool)  
3      Only log this once with whichever channel can first support it, taking the channels in priority  
4      order.  
5 PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)  
6      Message blob to be sent somewhere.  
7 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)  
8      Log via email based on pmix_info_t containing directives.  
9 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)  
10     Comma-delimited list of email addresses that are to receive the message.  
11 PMIX_LOG_EMAIL_SENDER_ADDR "pmix.log.emfaddr" (char*)  
12     Return email address of sender.  
13 PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)  
14     Subject line for email.  
15 PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)  
16     Message to be included in email.  
17 PMIX_LOG_EMAIL_SERVER "pmix.log.esrvr" (char*)  
18     Hostname (or IP address) of SMTP server.  
19 PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (int32_t)  
20     Port the email server is listening to.  
21 PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)  
22     Store the log data in a global data store (e.g., database).  
23 PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)  
24     Log the provided information to the host environment's job record.
```

## CHAPTER 13

# Process Sets and Groups

1 PMIx supports two slightly related, but functionally different concepts known as *process sets* and  
2 *process groups*. This chapter defines these two concepts and describes how they are utilized, along  
3 with their corresponding APIs.

## 13.1 Process Sets

5 A PMIx *Process Set* is a user-provided or host environment assigned label associated with a given  
6 set of application processes. Processes can belong to multiple process *sets* at a time. Users may  
7 define a PMIx process set at time of application execution. For example, if using the command line  
8 parallel launcher "prun", one could specify process sets as follows:

9 \$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp

10 In this example, the processes in the first application will be labeled with a **PMIX\_PSET\_NAMES**  
11 attribute with a value of *ocean* while those in the second application will be labeled with an *ice*  
12 value. During the execution, application processes could lookup the process set attribute for any  
13 process using **PMIx\_Get**. Alternatively, other executing applications could utilize the  
14 **PMIx\_Query\_info** APIs to obtain the number of declared process sets in the system, a list of  
15 their names, and other information about them. In other words, the *process set* identifier provides a  
16 label by which an application can derive information about a process and its application - it does  
17 *not*, however, confer any operational function.

18 Host environments can create or delete process sets at any time through the  
19 **PMIx\_server\_define\_process\_set** and **PMIx\_server\_delete\_process\_set**  
20 APIs. PMIx servers shall notify all local clients of process set operations via the  
21 **PMIX\_PROCESS\_SET\_DEFINE** or **PMIX\_PROCESS\_SET\_DELETE** events.

22 Process *sets* differ from process *groups* in several key ways:

- 23
- 24 • Process *sets* have no implied relationship between their members - i.e., a process in a process set  
has no concept of a “pset rank” as it would in a process *group*.

25

  - 26 • Process *set* identifiers are set by the host environment or by the user at time of application  
27 submission for execution - there are no PMIx APIs provided by which an application can define a  
process set or change a process *set* membership. In contrast, PMIx process *groups* can only be  
28 defined dynamically by the application.

- Process *sets* are immutable - members cannot be added or removed once the set has been defined. In contrast, PMIx process *groups* can dynamically change their membership using the appropriate APIs.
- Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are involved in an operation are translated by their PMIx server into their *native* identifier prior to the operation being passed to the host environment. For example, an application can define a process group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the group id of *foo*. If the application subsequently calls the **PMIX\_Fence** API with a process identifier of **{foo, PMIX\_RANK\_WILDCARD}**, the PMIx server will replace that identifier with an array consisting of **{210456, 0}** and **{210456, 1}** - the host-assigned identifiers of the participating processes - prior to processing the request.
- Process *groups* can request that the host environment assign a unique **size\_t** Process Group Context IDentifier (PGCID) to the group at time of group construction. An Message Passing Interface (MPI) library may, for example, use the PGCID as the MPI communicator identifier for the group.

The two concepts do, however, overlap in that they both involve collections of processes. Users desiring to create a process group based on a process set could, for example, obtain the membership array of the process set and use that as input to **PMIx\_Group\_construct**, perhaps including the process set name as the group identifier for clarity. Note that no linkage between the set and group of the same name is implied nor maintained - e.g., changes in process group membership can not be reflected in the process set using the same identifier.

### Advice to PMIx server hosts

The host environment is responsible for ensuring:

- consistent knowledge of process set membership across all involved PMIx servers; and
- that process set names do not conflict with system-assigned namespaces within the scope of the set.

## 13.1.1 Process Set Constants

**PMIx v4.0** The PMIx server is required to send a notification to all local clients upon creation or deletion of process sets. Client processes wishing to receive such notifications must register for the corresponding event:

**PMIX\_PROCESS\_SET\_DEFINE** The host environment has defined a new process set - the event will include the process set name (**PMIX\_PSET\_NAME**) and the membership (**PMIX\_PSET\_MEMBERS**).

**PMIX\_PROCESS\_SET\_DELETE** The host environment has deleted a process set - the event will include the process set name (**PMIX\_PSET\_NAME**).

## 13.1.2 Process Set Attributes

Several attributes are provided for querying the system regarding process sets using the `PMIx_Query_info` APIs.

`PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)`  
Return the number of process sets defined in the specified range (defaults to `PMIX_RANGE_SESSION`).  
`PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)`  
Return a `pmix_data_array_t` containing an array of strings of the process set names defined in the specified range (defaults to `PMIX_RANGE_SESSION`).  
`PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t*)`  
Return an array of `pmix_proc_t` containing the members of the specified process set.

The `PMIX_PROCESS_SET_DEFINE` event shall include the name of the newly defined process set and its members: `PMIX_PSET_NAME "pmix.pset.nm" (char*)`

The name of the newly defined process set.

`PMIX_PSET_MEMBERS "pmix.pset.mems" (pmix_data_array_t*)`  
An array of `pmix_proc_t` containing the members of the newly defined process set.

In addition, a process can request (via `PMIx_Get`) the process sets to which a given process (including itself) belongs:

`PMIX_PSET_NAMES "pmix.pset.nms" (pmix_data_array_t*)`  
Returns an array of `char*` string names of the process sets in which the given process is a member.

## 13.2 Process Groups

PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for operational purposes such as passing events or participating in PMIx fence operations. As with processes that assemble via `PMIx_Connect`, each member of the group is provided with both the job-level information of any other namespace represented in the group, and the contact information for all group members.

However, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected* when constructed via `PMIx_Connect`. Thus, *groups* differ from `PMIx_Connect` assemblages in several key areas, as detailed in the following sections.

### 13.2.1 Relation to the host environment

Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed, the server creates a tracker that associates the specified processes with the user-provided group identifier, and assigns a new *group rank* based on their relative position in the array of processes provided in the call to `PMIx_Group_construct`. Members of the group can subsequently

1 utilize the group identifier in PMIx function calls to address the group's members, using either  
2 **PMIX\_RANK\_WILDCARD** to refer to all of them or the group-level rank of specific members. The  
3 PMIx server will translate the specified processes into their RM-assigned identifiers prior to  
4 passing the request up to its host. Thus, the host environment has no visibility into the group's  
5 existence or membership.

6 In contrast, calls to **PMIx\_Connect** are relayed to the host environment. This means that the host  
7 RM should treat the failure of any process in the specified assemblage as a reportable event and  
8 take appropriate action. However, the environment is not required to define a new identifier for the  
9 connected assemblage or any of its member processes, nor does it define a new rank for each  
10 process within that assemblage. In addition, the PMIx server does not provide any tracking support  
11 for the assemblage. Thus, the caller is responsible for addressing members of the connected  
12 assemblage using their RM-provided identifiers.

### Advice to users

13 User-provided group identifiers must be distinct from both other group identifiers within the system  
14 and namespaces provided by the RM so as to avoid collisions between group identifiers and  
15 RM-assigned namespaces. This can usually be accomplished through the use of an  
16 application-specific prefix – e.g., “myapp-foo”

## 13.2.2 Construction procedure

18 **PMIx\_Connect** calls require that every process call the API before completing – i.e., it is  
19 modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a given  
20 application thread can only be involved in one connect/accept operation at a time, and is blocked in  
21 that operation until all specified processes participate. In addition, there is no provision for  
22 replacing processes in the assemblage due to failure to participate, nor a mechanism by which a  
23 process might decline participation.

24 In contrast, PMIx Groups are designed to be more flexible in their construction procedure by  
25 relaxing these constraints. While a standard blocking form of constructing groups is provided, the  
26 event notification system is utilized to provide a designated *group leader* with the ability to replace  
27 participants that fail to participate within a given timeout period. This provides a mechanism by  
28 which the application can, if desired, replace members on-the-fly or allow the group to proceed  
29 with partial membership. In such cases, the final group membership is returned to all participants  
30 upon completion of the operation.

31 Additionally, PMIx supports dynamic definition of group membership based on an invite/join  
32 model. A process can asynchronously initiate construction of a group of any processes via the  
33 **PMIx\_Group\_invite** function call. Invitations are delivered via a PMIx event (using the  
34 **PMIX\_GROUP\_INVITED** event) to the invited processes which can then either accept or decline  
35 the invitation using the **PMIx\_Group\_join** API. The initiating process tracks responses by  
36 registering for the events generated by the call to **PMIx\_Group\_join**, timeouts, or process

1 terminations, optionally replacing processes that decline the invitation, fail to respond in time, or  
2 terminate without responding. Upon completion of the operation, the final list of participants is  
3 communicated to each member of the new group.

### 4 13.2.3 Destruct procedure

5 Members of a PMIx Group may depart the group at any time via the **PMIx\_Group\_Leave** API.  
6 Other members are notified of the departure via the **PMIX\_GROUP\_LEFT** event to distinguish such  
7 events from those reporting process termination. This leaves the remaining members free to  
8 continue group operations. The **PMIx\_Group\_Destruct** operation offers a collective method  
9 akin to **PMIx\_Disconnect** for deconstructing the entire group.

10 In contrast, processes that assemble via **PMIx\_Connect** must all depart the assemblage together –  
11 i.e., no member can depart the assemblage while leaving the remaining members in it. Even the  
12 non-blocking form of **PMIx\_Disconnect** retains this requirement in that members remain a part  
13 of the assemblage until all members have called **PMIx\_Disconnect\_nb**.

14 Note that applications supporting dynamic group behaviors such as asynchronous departure take  
15 responsibility for ensuring global consistency in the group definition prior to executing group  
16 collective operations - i.e., it is the application's responsibility to either ensure that knowledge of  
17 the current group membership is globally consistent across the participants, or to register for  
18 appropriate events to deal with the lack of consistency during the operation.

#### Advice to users

19 The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs  
20 must register for the corresponding events. Failure to do so will likely lead to operational failures.  
21 Users are recommended to utilize the **PMIX\_TIMEOUT** directive (or retain an internal timer) on  
22 calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have  
23 not registered for required events will never respond.

### 24 13.2.4 Process Group Events

25 *PMIx v4.0* Asynchronous process group operations rely heavily on PMIx events. The following events have  
26 been defined for that purpose.

27 **PMIX\_GROUP\_INVITED** The process has been invited to join a PMIx Group - the identifier of  
28 the group and the ID's of other invited (or already joined) members will be included in the  
29 notification.

30 **PMIX\_GROUP\_LEFT** A process has asynchronously left a PMIx Group - the process identifier  
31 of the departing process will be included in the notification.

32 **PMIX\_GROUP\_MEMBER\_FAILED** A member of a PMIx Group has abnormally terminated  
33 (i.e., without formally leaving the group prior to termination) - the process identifier of the  
34 failed process will be included in the notification.

```
1   PMIX_GROUP_INVITE_ACCEPTED A process has accepted an invitation to join a PMIx  
2       Group - the identifier of the group being joined will be included in the notification.  
3   PMIX_GROUP_INVITE_DECLINED A process has declined an invitation to join a PMIx  
4       Group - the identifier of the declined group will be included in the notification.  
5   PMIX_GROUP_INVITE_FAILED An invited process failed or terminated prior to responding  
6       to the invitation - the identifier of the failed process will be included in the notification.  
7   PMIX_GROUP_MEMBERSHIP_UPDATE The membership of a PMIx group has changed - the  
8       identifiers of the revised membership will be included in the notification.  
9   PMIX_GROUP_CONSTRUCT_ABORT Any participant in a PMIx group construct operation  
10      that returns PMIX_GROUP_CONSTRUCT_ABORT from the leader failed event handler will  
11      cause all participants to receive an event notifying them of that status. Similarly, the leader  
12      may elect to abort the procedure by either returning this error code from the handler assigned  
13      to the PMIX_GROUP_INVITE_ACCEPTED or PMIX_GROUP_INVITE_DECLINED  
14      codes, or by generating an event for the abort code. Abort events will be sent to all invited or  
15      existing members of the group.  
16   PMIX_GROUP_CONSTRUCT_COMPLETE The group construct operation has completed - the  
17      final membership will be included in the notification.  
18   PMIX_GROUP_LEADER FAILED The current leader of a group including this process has  
19      abnormally terminated - the group identifier will be included in the notification.  
20   PMIX_GROUP_LEADER_SELECTED A new leader of a group including this process has been  
21      selected - the identifier of the new leader will be included in the notification.  
22   PMIX_GROUP_CONTEXT_ID_ASSIGNED A new PGCID has been assigned by the host  
23      environment to a group that includes this process - the group identifier will be included in the  
24      notification.
```

## 13.2.5 Process Group Attributes

26 *PMIx v4.0* Attributes for querying the system regarding process groups include:

```
27   PMIX_QUERY_NUM_GROUPS "pmixqry.pgrnum" (size_t)  
28       Return the number of process groups defined in the specified range (defaults to session).  
29       OPTIONAL QUALIFIERS: PMIX_RANGE.  
30   PMIX_QUERY_GROUP_NAMES "pmixqry.pgrp" (pmix_data_array_t*)  
31       Return a pmix_data_array_t containing an array of string names of the process groups  
32       defined in the specified range (defaults to session). OPTIONAL QUALIFIERS:  
33           PMIX_RANGE.  
34   PMIX_QUERY_GROUP_MEMBERSHIP  
35       "pmixqry.pgrpmems" (pmix_data_array_t*)  
36       Return a pmix_data_array_t of pmix_proc_t containing the members of the  
37       specified process group. REQUIRED QUALIFIERS: PMIX_GROUP_ID.
```

38 The following attributes are used as directives in PMIx Group operations:

```
39   PMIX_GROUP_ID "pmix.grp.id" (char*)
```

1 User-provided group identifier - as the group identifier may be used in PMIx operations, the  
 2 user is required to ensure that the provided ID is unique within the scope of the host  
 3 environment (e.g., by including some user-specific or application-specific prefix or suffix to  
 4 the string).

5 **PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (bool)  
 6 This process is the leader of the group.  
 7 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)  
 8 Participation is optional - do not return an error if any of the specified processes terminate  
 9 without having joined. The default is **false**.  
 10 **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)  
 11 Notify remaining members when another member terminates without first leaving the group.  
 12 **PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)  
 13 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective  
 14 operation.  
 15 **PMIX\_GROUP\_MEMBERSHIP** "pmix.grp.mbrs" (pmix\_data\_array\_t\*)  
 16 Array **pmix\_proc\_t** identifiers identifying the members of the specified group.  
 17 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)  
 18 Requests that the RM assign a new context identifier to the newly created group. The  
 19 identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
 20 specified in the request. Thus, the value serves as a means of identifying the group within  
 21 that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.  
 22 **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)  
 23 Group operation only involves local processes. PMIx implementations are *required* to  
 24 automatically scan an array of group members for local vs remote processes - if only local  
 25 processes are detected, the implementation need not execute a global collective for the  
 26 operation unless a context ID has been requested from the host environment. This can result  
 27 in significant time savings. This attribute can be used to optimize the operation by indicating  
 28 whether or not only local processes are represented, thus allowing the implementation to  
 29 bypass the scan.

30 The following attributes are used to return information at the conclusion of a PMIx Group  
 31 operation and/or in event notifications:

32 **PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (**size\_t**)  
 33 Context identifier assigned to the group by the host RM.  
 34 **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)  
 35 Data collected during group construction to ensure communication between group members  
 36 is supported upon completion of the operation.

37 In addition, a process can request (via **PMIx\_Get**) the process groups to which a given process  
 38 (including itself) belongs:

39 **PMIX\_GROUP\_NAMES** "pmix.pgrp.nm" (pmix\_data\_array\_t\*)

1                Returns an array of `char*` string names of the process groups in which the given process is  
2                a member.

### 3 13.2.6 `PMIx_Group_construct`

#### 4                **Summary**

5                Construct a PMIx process group.

#### 6                **Format**

7                *PMIx v4.0*

C

```
8 pmix_status_t
9 PMIx_Group_construct(const char grp[],
10                    const pmix_proc_t procs[], size_t nprocs,
11                    const pmix_info_t directives[],
12                    size_t ndirs,
13                    pmix_info_t **results,
                    size_t *nresults);
```

C

#### 14              **IN grp**

15                NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group  
16                identifier (string)

#### 17              **IN procs**

18                Array of `pmix_proc_t` structures containing the PMIx identifiers of the member processes  
19                (array of handles)

#### 20              **IN nprocs**

21                Number of elements in the *procs* array (`size_t`)

#### 22              **IN directives**

23                Array of `pmix_info_t` structures (array of handles)

#### 24              **IN ndirs**

25                Number of elements in the *directives* array (`size_t`)

#### 26              **INOUT results**

27                Pointer to a location where the array of `pmix_info_t` describing the results of the  
28                operation is to be returned (pointer to handle)

#### 29              **INOUT nresults**

30                Pointer to a `size_t` location where the number of elements in *results* is to be returned  
31                (memory reference)

32                Returns one of the following:

- 33                • `PMIX_SUCCESS`, indicating that the request has been successfully completed
- 34                • `PMIX_ERR_NOT_SUPPORTED` The PMIx library and/or the host RM does not support this  
35                operation
- 36                • a PMIx error constant indicating either an error in the input or that the request failed to be  
37                completed

## Required Attributes

1 The following attributes are *required* to be supported by all PMIx libraries that support this  
2 operation:

3 **PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (bool)

4 This process is the leader of the group.

5 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

6 Participation is optional - do not return an error if any of the specified processes terminate  
7 without having joined. The default is **false**.

8 **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

9 Group operation only involves local processes. PMIx implementations are *required* to  
10 automatically scan an array of group members for local vs remote processes - if only local  
11 processes are detected, the implementation need not execute a global collective for the  
12 operation unless a context ID has been requested from the host environment. This can result  
13 in significant time savings. This attribute can be used to optimize the operation by indicating  
14 whether or not only local processes are represented, thus allowing the implementation to  
15 bypass the scan.

16 **PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)

17 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective  
18 operation.

19 Host environments that support this operation are *required* to support the following attributes:

20 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

21 Requests that the RM assign a new context identifier to the newly created group. The  
22 identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
23 specified in the request. Thus, the value serves as a means of identifying the group within  
24 that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

25 **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

26 Notify remaining members when another member terminates without first leaving the group.  
27

## Optional Attributes

28 The following attributes are optional for host environments that support this operation:

29 **PMIX\_TIMEOUT** "pmix.timeout" (int)

30 Time in seconds before the specified operation should time out (zero indicating infinite) and  
31 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
32 caused by multiple layers (client, server, and host) simultaneously timing the operation.

## 1      Description

2      Construct a new group composed of the specified processes and identified with the provided group  
3      identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less  
4      than or equal to **PMIX\_MAX\_NSLEN**. Only characters accepted by standard string comparison  
5      functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group  
6      construct operations so long as each is provided with a unique group ID. The *directives* array can be  
7      used to pass user-level directives regarding timeout constraints and other options available from the  
8      PMIx server.

9      If the **PMIX\_GROUP\_NOTIFY\_TERMINATION** attribute is provided and has a value of **true**,  
10     then either the construct leader (if **PMIX\_GROUP\_LEADER** is provided) or all participants who  
11     register for the **PMIX\_GROUP\_MEMBER FAILED** event will receive events whenever a process  
12     fails or terminates prior to calling **PMIx\_Group\_construct** – i.e. if a *group leader* is declared,  
13     *only* that process will receive the event. In the absence of a declared leader, *all* specified group  
14     members will receive the event.

15     The event will contain the identifier of the process that failed to join plus any other information that  
16     the host RM provided. This provides an opportunity for the leader or the collective members to  
17     react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The  
18     decision is communicated to the PMIx library in the results array at the end of the event handler.  
19     This allows PMIx to properly adjust accounting for procedure completion. When construct is  
20     complete, the participating PMIx servers will be alerted to any change in participants and each  
21     group member will receive an updated group membership (marked with the  
22     **PMIX\_GROUP\_MEMBERSHIP** attribute) as part of the *results* array returned by this API.

23     Failure of the declared leader at any time will cause a **PMIX\_GROUP LEADER FAILED** event to  
24     be delivered to all participants so they can optionally declare a new leader. A new leader is  
25     identified by providing the **PMIX\_GROUP LEADER** attribute in the results array in the return of  
26     the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the  
27     new leader. Results of the leader selection will be communicated to all participants via a  
28     **PMIX\_GROUP LEADER SELECTED** event identifying the new leader. If no leader was selected,  
29     then the **pmix\_info\_t** provided to that event handler will include that information so the  
30     participants can take appropriate action.

31     Any participant that returns **PMIX\_GROUP CONSTRUCT ABORT** from either the  
32     **PMIX\_GROUP MEMBER FAILED** or the **PMIX\_GROUP LEADER FAILED** event handler will  
33     cause the construct process to abort, returning from the call with a  
34     **PMIX\_GROUP CONSTRUCT ABORT** status.

35     If the **PMIX\_GROUP NOTIFY TERMINATION** attribute is not provided or has a value of  
36     **false**, then the **PMIx\_Group\_construct** operation will simply return an error whenever a  
37     proposed group member fails or terminates prior to calling **PMIx\_Group\_construct**.

38     Providing the **PMIX\_GROUP OPTIONAL** attribute with a value of **true** directs the PMIx library  
39     to consider participation by any specified group member as non-required - thus, the operation will  
40     return **PMIX\_SUCCESS** if all members participate, or **PMIX\_ERR\_PARTIAL\_SUCCESS** if some

1 members fail to participate. The *results* array will contain the final group membership in the latter  
2 case. Note that this use-case can cause the operation to hang if the **PMIX\_TIMEOUT** attribute is  
3 not specified and one or more group members fail to call **PMIx\_Group\_construct** while  
4 continuing to execute. Also, note that no leader or member failed events will be generated during  
5 the operation.

6 Processes in a group under construction are not allowed to leave the group until group construction  
7 is complete. Upon completion of the construct procedure, each group member will have access to  
8 the job-level information of all namespaces represented in the group plus any information posted  
9 via **PMIx\_Put** (subject to the usual scoping directives) for every group member.

#### Advice to PMIx library implementers

10 At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related  
11 information from each participating namespace plus any information posted by group members via  
12 **PMIx\_Put** (subject to scoping directives) is available to each member via calls to **PMIx\_Get**.

#### Advice to PMIx server hosts

13 The collective nature of this API generally results in use of a fence-like operation by the backend  
14 host environment. Host environments that utilize the array of process participants as a *signature* for  
15 such operations may experience potential conflicts should both a **PMIx\_Group\_construct** and  
16 a **PMIx\_Fence** operation involving the same participants be simultaneously executed. As PMIx  
17 allows for such use-cases, it is therefore the responsibility of the host environment to resolve any  
18 potential conflicts.

### 13.2.7 **PMIx\_Group\_construct\_nb**

#### Summary

Non-blocking form of **PMIx\_Group\_construct**.

1           **Format**

C

```
2       pmix_status_t  
3       PMIx_Group_construct_nb(const char grp[],  
4                            const pmix_proc_t procs[], size_t nprocs,  
5                            const pmix_info_t directives[],  
6                            size_t ndirs,  
7                            pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

8       **IN** **grp**

9       NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group  
10      identifier (string)

11      **IN** **procs**

12      Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the member processes  
13      (array of handles)

14      **IN** **nprocs**

15      Number of elements in the *procs* array (**size\_t**)

16      **IN** **directives**

17      Array of **pmix\_info\_t** structures (array of handles)

18      **IN** **ndirs**

19      Number of elements in the *directives* array (**size\_t**)

20      **IN** **cbfunc**

21      Callback function **pmix\_info\_cbfunc\_t** (function reference)

22      **IN** **cbdata**

23      Data to be passed to the callback function (memory reference)

24      Returns one of the following:

- **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided  
call back function will be executed upon completion of the operation. Note that the library *must*  
*not* invoke the call back function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
returned *success* - the *cbfunc* will *not* be called.
- **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*  
will *not* be called.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected - the  
*cbfunc* will *not* be called.

34      If executed, the status returned in the provided call back function will be one of the following  
35      constants:

- **PMIX\_SUCCESS** The operation succeeded and all specified members participated.

- **PMIX\_ERR\_PARTIAL\_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function.
- **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request's failure.

## Required Attributes

PMIx libraries that choose not to support this operation *must* return **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called.

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

**PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (bool)

This process is the leader of the group.

**PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is **false**.

**PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

Group operation only involves local processes. PMIx implementations are *required* to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

**PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)

Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are *required* to provide the following attributes:

**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

**PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

Notify remaining members when another member terminates without first leaving the group.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT** "pmix.timeout" (**int**)

3 Time in seconds before the specified operation should time out (zero indicating infinite) and  
4 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
5 caused by multiple layers (client, server, and host) simultaneously timing the operation.

### 6 **Description**

7 Non-blocking version of the **PMIx\_Group\_construct** operation. The callback function will be  
8 called once all group members have called either **PMIx\_Group\_construct** or  
9 **PMIx\_Group\_construct\_nb**.

## 10 **13.2.8 PMIx\_Group\_destruct**

### 11 **Summary**

12 Destuct a PMIx process group.

### 13 **Format**

PMIx v4.0

C

```
14     pmix_status_t
15     PMIx_Group_destruct(const char grp[],
16                           const pmix_info_t directives[],
17                           size_t ndirs);
```

C

#### 18 **IN grp**

19 NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
20 identifier of the group to be destructed (string)

#### 21 **IN directives**

22 Array of **pmix\_info\_t** structures (array of handles)

#### 23 **IN ndirs**

24 Number of elements in the *directives* array (**size\_t**)

25 Returns one of the following:

- 26 • **PMIX\_SUCCESS**, indicating that the request has been successfully completed
- 27 • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this  
28 operation
- 29 • a PMIx error constant indicating either an error in the input or that the request failed to be  
30 completed

## Required Attributes

For implementations and host environments that support the operation, there are no identified required attributes for this API.

## Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX\_TIMEOUT "pmix.timeout" (int)**

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling **PMIx\_Group\_destruct** or its non-blocking version unless the **PMIX\_GROUP\_NOTIFY\_TERMINATION** attribute was provided (with a value of **false**) at time of group construction. If notification was requested, then the **PMIX\_GROUP\_MEMBER\_FAILED** event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The **PMIx\_Group\_destruct** operation will subsequently return **PMIX\_SUCCESS** when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

## Advice to PMIx server hosts

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a **PMIx\_Group\_destruct** and a **PMIx\_Fence** operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

### 13.2.9 **PMIx\_Group\_destruct\_nb**

#### Summary

Non-blocking form of **PMIx\_Group\_destruct**.

1      **Format**

2      *PMIx v4.0*      C

```
3      pmix_status_t
4      PMIx_Group_destruct_nb(const char grp[],  

5                        const pmix_info_t directives[],  

6                       size_t ndirs,  

7                       pmix_op_cbfunc_t cbfunc, void *cbdata);
```

- 8      **IN** **grp**  
9      NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
10     identifier of the group to be destructed (string)  
11     **IN** **directives**  
12     Array of **pmix\_info\_t** structures (array of handles)  
13     **IN** **ndirs**  
14     Number of elements in the *directives* array (**size\_t**)  
15     **IN** **cbfunc**  
16     Callback function **pmix\_op\_cbfunc\_t** (function reference)  
17     **IN** **cbdata**  
18     Data to be passed to the callback function (memory reference)

19     Returns one of the following:

- 20     • **PMIX\_SUCCESS**, indicating that the request is being processed - result will be returned in the  
21       provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning  
22       from the API.  
23     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
24       returned *success* - the *cbfunc* will *not* be called  
25     • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*  
26       will *not* be called.  
27     • a PMIx error constant indicating either an error in the input or that the request was immediately  
28       processed and failed - the *cbfunc* will *not* be called.

29     If executed, the status returned in the provided callback function will be one of the following  
constants:

- 30     • **PMIX\_SUCCESS** The operation was successfully completed.  
31     • **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM  
32       does not.  
33     • a non-zero PMIx error constant indicating a reason for the request's failure.

## Required Attributes

1 PMIx libraries that choose not to support this operation *must* return  
2 **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called. For implementations and host  
3 environments that support the operation, there are no identified required attributes for this API.

## Optional Attributes

4 The following attributes are optional for host environments that support this operation:

5 **PMIX\_TIMEOUT "pmix.timeout" (int)**

6 Time in seconds before the specified operation should time out (zero indicating infinite) and  
7 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
8 caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

10 Non-blocking version of the **PMIx\_Group\_destruct** operation. The callback function will be  
11 called once all members of the group have executed either **PMIx\_Group\_destruct** or  
12 **PMIx\_Group\_destruct\_nb**.

### 13 13.2.10 **PMIx\_Group\_invite**

#### 14 Summary

15 Asynchronously construct a PMIx process group.

1      **Format**

2      *pmix\_status\_t*  
3      **PMIx\_Group\_invite**(const char grp[],  
4                            const pmix\_proc\_t procs[], size\_t nprocs,  
5                            const pmix\_info\_t directives[], size\_t ndirs,  
6                            pmix\_info\_t \*\*results, size\_t \*nresult);

C

C

7      **IN    grp**

8       NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group  
9       identifier (string)

10     **IN    procs**

11     Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the processes to be  
12     invited (array of handles)

13     **IN    nprocs**

14     Number of elements in the *procs* array (**size\_t**)

15     **IN    directives**

16     Array of **pmix\_info\_t** structures (array of handles)

17     **IN    ndirs**

18     Number of elements in the *directives* array (**size\_t**)

19     **INOUT results**

20     Pointer to a location where the array of **pmix\_info\_t** describing the results of the  
21     operation is to be returned (pointer to handle)

22     **INOUT nresults**

23     Pointer to a **size\_t** location where the number of elements in *results* is to be returned  
24     (memory reference)

25     Returns one of the following:

- 26     • **PMIX\_SUCCESS**, indicating that the request has been successfully completed.
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this  
28       operation.
- 29     • a PMIx error constant indicating either an error in the input or that the request failed to be  
30       completed.

31     ----- Required Attributes -----

32     The following attributes are *required* to be supported by all PMIx libraries that support this  
operation:

33     **PMIX\_GROUP\_OPTIONAL "pmix.grp.opt" (bool)**

34     Participation is optional - do not return an error if any of the specified processes terminate  
35     without having joined. The default is **false**.

```
1 PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)
2     Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective
3     operation.
4 Host environments that support this operation are required to provide the following attributes:
5 PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)
6     Requests that the RM assign a new context identifier to the newly created group. The
7     identifier is an unsigned, size_t value that the RM guarantees to be unique across the range
8     specified in the request. Thus, the value serves as a means of identifying the group within
9     that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
10 PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)
11     Notify remaining members when another member terminates without first leaving the group.
12
```

### Optional Attributes

The following attributes are optional for host environments that support this operation:

```
14 PMIX_TIMEOUT "pmix.timeout" (int)
15     Time in seconds before the specified operation should time out (zero indicating infinite) and
16     return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions
17     caused by multiple layers (client, server, and host) simultaneously timing the operation.
```

## Description

Explicitly invite the specified processes to join a group. The process making the **PMIx\_Group\_invite** call is automatically declared to be the *group leader*. Each invited process will be notified of the invitation via the **PMIX\_GROUP\_INVITED** event - the processes being invited must therefore register for the **PMIX\_GROUP\_INVITED** event in order to be notified of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of invite versus event registration is required.

The invitation event will include the identity of the inviting process plus the name of the group. When ready to respond, each invited process provides a response using either the blocking or non-blocking form of **PMIx\_Group\_join**. This will notify the inviting process that the invitation was either accepted (via the **PMIX\_GROUP\_INVITE\_ACCEPTED** event) or declined (via the **PMIX\_GROUP\_INVITE\_DECLINED** event). The **PMIX\_GROUP\_INVITE\_ACCEPTED** event is captured by the PMIx client library of the inviting process – i.e., the application itself does not need to register for this event. The library will track the number of accepting processes and alert the inviting process (by returning from the blocking form of **PMIx\_Group\_invite** or calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the **PMIX\_GROUP\_INVITE\_DECLINED** if the application allows invited processes to decline the invitation. This provides an opportunity for the

1 application to either invite a replacement, declare “abort”, or choose to remove the declining  
2 process from the final group. The inviting process should also register to receive  
3 **PMIX\_GROUP\_INVITE\_FAILED** events whenever a process fails or terminates prior to  
4 responding to the invitation. Actions taken by the inviting process in response to these events must  
5 be communicated at the end of the event handler by returning the corresponding result so that the  
6 PMIx library can adjust accordingly.

7 Upon completion of the operation, all members of the new group will receive access to the job-level  
8 information of each other’s namespaces plus any information posted via **PMIx\_Put** by the other  
9 members.

10 The inviting process is automatically considered the leader of the asynchronous group construction  
11 procedure and will receive all failure or termination events for invited members prior to completion.  
12 The inviting process is required to provide a **PMIX\_GROUP\_CONSTRUCT\_COMPLETE** event once  
13 the group has been fully assembled – this event is used by the PMIx library as a trigger to release  
14 participants from their call to **PMIx\_Group\_join** and provides information (e.g., the final group  
15 membership) to be returned in the *results* array.

16 Failure of the inviting process at any time will cause a **PMIX\_GROUP\_LEADER\_FAILED** event to  
17 be delivered to all participants so they can optionally declare a new leader. A new leader is  
18 identified by providing the **PMIX\_GROUP\_LEADER** attribute in the results array in the return of  
19 the event handler. Only one process is allowed to return that attribute, declaring itself as the new  
20 leader. Results of the leader selection will be communicated to all participants via a  
21 **PMIX\_GROUP\_LEADER\_SELECTED** event identifying the new leader. If no leader was selected,  
22 then the status code provided in the event handler will provide an error value so the participants can  
23 take appropriate action.

---

#### Advice to users

---

24 Applications are not allowed to use the group in any operations until group construction is  
25 complete. This is required in order to ensure consistent knowledge of group membership across all  
26 participants.

### 27 **13.2.11 PMIx\_Group\_invite\_nb**

#### 28 **Summary**

29 Non-blocking form of **PMIx\_Group\_invite**.

1           **Format**

2        *PMIx v4.0*

C

```
3        pmix_status_t
4        PMIx_Group_invite_nb(const char grp[],  
5                            const pmix_proc_t procs[], size_t nprocs,  
6                            const pmix_info_t directives[], size_t ndirs,  
7                            pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

7        **IN**    **grp**

8           NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group  
9           identifier (string)

10      **IN**    **procs**

11        Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the processes to be  
12        invited (array of handles)

13      **IN**    **nprocs**

14        Number of elements in the *procs* array (**size\_t**)

15      **IN**    **directives**

16        Array of **pmix\_info\_t** structures (array of handles)

17      **IN**    **ndirs**

18        Number of elements in the *directives* array (**size\_t**)

19      **IN**    **cbfunc**

20        Callback function **pmix\_info\_cbfunc\_t** (function reference)

21      **IN**    **cbdata**

22        Data to be passed to the callback function (memory reference)

23        Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called.
- **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library does not support this operation - the *cbfunc* will *not* be called.
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called.

33        If executed, the status returned in the provided callback function will be one of the following  
34        constants:

- **PMIX\_SUCCESS** The operation succeeded and all specified members participated.

- **PMIX\_ERR\_PARTIAL\_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function.
  - **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
  - a non-zero PMIx error constant indicating a reason for the request's failure.

## Required Attributes

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

**PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is **false**.

**PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)

Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are *required* to provide the following attributes:

**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, `size_t` value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to `PMIX_RANGE_SESSION`.

**PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

Notify remaining members when another member terminates without first leaving the group.

## Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX\_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

Non-blocking version of the [PMIx\\_Group\\_invite](#) operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either [PMIx\\_Group\\_join](#) or [PMIx\\_Group\\_join\\_nb](#).

## 13.2.12 PMIx\_Group\_join

### Summary

Accept an invitation to join a PMIx process group.

### Format

PMIx v4.0

C

```
5     pmix_status_t
6     PMIx_Group_join(const char grp[],           C
7             const pmix_proc_t *leader,
8                     pmix_group_opt_t opt,
9                     const pmix_info_t directives[], size_t ndirs,
10                    pmix_info_t **results, size_t *nresult);
```

#### IN grp

NULL-terminated character array of maximum size [PMIX\\_MAX\\_NSLEN](#) containing the group identifier (string)

#### IN leader

Process that generated the invitation (handle)

#### IN opt

Accept or decline flag ([pmix\\_group\\_opt\\_t](#))

#### IN directives

Array of [pmix\\_info\\_t](#) structures (array of handles)

#### IN ndirs

Number of elements in the *directives* array ([size\\_t](#))

#### INOUT results

Pointer to a location where the array of [pmix\\_info\\_t](#) describing the results of the operation is to be returned (pointer to handle)

#### INOUT nresults

Pointer to a [size\\_t](#) location where the number of elements in *results* is to be returned (memory reference)

Returns one of the following:

- [PMIX\\_SUCCESS](#), indicating that the request has been successfully completed.
- [PMIX\\_ERR\\_NOT\\_SUPPORTED](#) The PMIx library and/or the host RM does not support this operation.
- a PMIx error constant indicating either an error in the input or that the request failed to be completed.

### Required Attributes

There are no identified required attributes for implementers.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT "pmix.timeout" (int)**

3 Time in seconds before the specified operation should time out (zero indicating infinite) and  
4 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
5 caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

6 Respond to an invitation to join a group that is being asynchronously constructed. The process must  
7 have registered for the **PMIX\_GROUP\_INVITED** event in order to be notified of the invitation.  
8 When called, the event information will include the **pmix\_proc\_t** identifier of the process that  
9 generated the invitation along with the identifier of the group being constructed. When ready to  
10 respond, the process provides a response using either form of **PMIx\_Group\_join**.  
11

### Advice to users

12 Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the  
13 blocking form of this call unless it first “thread shifts” out of the handler and into its own thread  
14 context. Likewise, while it is safe to call the non-blocking form of the API from the event handler,  
15 the process *must not* block in the handler while waiting for the callback function to be called.

16 Calling this function causes the inviting process (aka the *group leader*) to be notified that the  
17 process has either accepted or declined the request. The blocking form of the API will return once  
18 the group has been completely constructed or the group’s construction has failed (as described  
19 below) – likewise, the callback function of the non-blocking form will be executed upon the same  
20 conditions.

21 Failure of the leader during the call to **PMIx\_Group\_join** will cause a  
22 **PMIX\_GROUP\_LEADER\_FAILED** event to be delivered to all invited participants so they can  
23 optionally declare a new leader. A new leader is identified by providing the  
24 **PMIX\_GROUP\_LEADER** attribute in the results array in the return of the event handler. Only one  
25 process is allowed to return that attribute, declaring itself as the new leader. Results of the leader  
26 selection will be communicated to all participants via a **PMIX\_GROUP\_LEADER\_SELECTED**  
27 event identifying the new leader. If no leader was selected, then the status code provided in the  
28 event handler will provide an error value so the participants can take appropriate action.

29 Any participant that returns **PMIX\_GROUP\_CONSTRUCT\_ABORT** from the leader failed event  
30 handler will cause all participants to receive an event notifying them of that status. Similarly, the  
31 leader may elect to abort the procedure by either returning **PMIX\_GROUP\_CONSTRUCT\_ABORT**  
32 from the handler assigned to the **PMIX\_GROUP\_INVITE\_ACCEPTED** or  
33 **PMIX\_GROUP\_INVITE\_DECLINED** codes, or by generating an event for the abort code. Abort  
34 events will be sent to all invited participants.

## 13.2.13 PMIx\_Group\_join\_nb

### 2 Summary

3 Non-blocking form of [PMIx\\_Group\\_join](#)

### 4 Format

5 *PMIx v4.0*

C

```
6     pmix_status_t
7     PMIx_Group_join_nb(const char grp[],           C
8                     const pmix_proc_t *leader,
9                     pmix_group_opt_t opt,
10                    const pmix_info_t directives[], size_t ndirs,
11                    pmix_info_cfunc_t cbfunc, void *cbdata);
```

#### 11 IN grp

12 NULL-terminated character array of maximum size [PMIX\\_MAX\\_NSLEN](#) containing the group  
13 identifier (string)

#### 14 IN leader

15 Process that generated the invitation (handle)

#### 16 IN opt

17 Accept or decline flag ([pmix\\_group\\_opt\\_t](#))

#### 18 IN directives

19 Array of [pmix\\_info\\_t](#) structures (array of handles)

#### 20 IN ndirs

21 Number of elements in the *directives* array ([size\\_t](#))

#### 22 IN cbfunc

23 Callback function [pmix\\_info\\_cfunc\\_t](#) (function reference)

#### 24 IN cbdata

25 Data to be passed to the callback function (memory reference)

26 Returns one of the following:

- 27 • [PMIX\\_SUCCESS](#), indicating that the request is being processed - result will be returned in the  
28 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning  
29 from the API.
- 30 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
31 returned *success* - the *cbfunc* will *not* be called.
- 32 • [PMIX\\_ERR\\_NOT\\_SUPPORTED](#) The PMIx library does not support this operation - the *cbfunc*  
33 will *not* be called.
- 34 • a PMIx error constant indicating either an error in the input or that the request was immediately  
35 processed and failed - the *cbfunc* will *not* be called.

1 If executed, the status returned in the provided callback function will be one of the following  
2 constants:

- 3 • **PMIX\_SUCCESS** The operation succeeded and group membership is in the callback function  
4 parameters.
- 5 • **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM  
6 does not.
- 7 • a non-zero PMIx error constant indicating a reason for the request's failure.

▼----- Required Attributes -----▼

8 There are no identified required attributes for implementers.  
▲-----

▼----- Optional Attributes -----▼

9 The following attributes are optional for host environments that support this operation:

10 **PMIX\_TIMEOUT "pmix.timeout" (int)**

11 Time in seconds before the specified operation should time out (zero indicating infinite) and  
12 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
13 caused by multiple layers (client, server, and host) simultaneously timing the operation.  
▲-----

14 **Description**

15 Non-blocking version of the **PMIx\_Group\_join** operation. The callback function will be called  
16 once all invited members of the group (or their substitutes) have executed either  
17 **PMIx\_Group\_join** or **PMIx\_Group\_join\_nb**.

18 **13.2.13.1 Group accept/decline directives**

19 *PMIx v4.0* The **pmix\_group\_opt\_t** type is a **uint8\_t** value used with the **PMIx\_Group\_join** API to  
20 indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

21 **PMIX\_GROUP\_DECLINE** Decline the invitation.

22 **PMIX\_GROUP\_ACCEPT** Accept the invitation.

23 **13.2.14 PMIx\_Group\_leave**

24 **Summary**

25 Leave a PMIx process group.

1           **Format**

2            PMIx v4.0

3            pmix\_status\_t

4            PMIx\_Group\_leave(const char grp[],

5                        const pmix\_info\_t directives[],

6                       size\_t ndirs);

C

C

6           **IN    grp**

7            NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group  
8            identifier (string)

9           **IN    directives**

10           Array of **pmix\_info\_t** structures (array of handles)

11           **IN    ndirs**

12           Number of elements in the *directives* array (**size\_t**)

13           Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request has been communicated to the local PMIx server.
- **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this operation.
- a PMIx error constant indicating either an error in the input or that the request is unsupported.

18           **Required Attributes**

19           There are no identified required attributes for implementers.

19           **Description**

20           Calls to **PMIx\_Group\_leave** (or its non-blocking form) will cause a **PMIX\_GROUP\_LEFT**  
21           event to be generated notifying all members of the group of the caller's departure. The function will  
22           return (or the non-blocking function will execute the specified callback function) once the event has  
23           been locally generated and is not indicative of remote receipt.

24           **Advice to users**

25           The **PMIx\_Group\_leave** API is intended solely for asynchronous departures of individual  
26           processes from a group as it is not a scalable operation – i.e., when a process determines it should  
27           no longer be a part of a defined group, but the remainder of the group retains a valid reason to  
28           continue in existence. Developers are advised to use **PMIx\_Group\_destruct** (or its  
non-blocking form) for all other scenarios as it represents a more scalable operation.

## 13.2.15 PMIx\_Group\_leave\_nb

### 2 Summary

3 Non-blocking form of [PMIx\\_Group\\_leave](#).

### 4 Format

5 *PMIx v4.0*

C

```
6     pmix_status_t
7     PMIx_Group_leave_nb(const char grp[],  

8                         const pmix_info_t directives[],  

9                         size_t ndirs,  

10                        pmix_op_cbfunc_t cbfunc,  

11                        void *cbdata);
```

C

#### 11 IN grp

12 NULL-terminated character array of maximum size [PMIX\\_MAX\\_NSLEN](#) containing the group  
13 identifier (string)

#### 14 IN directives

15 Array of [pmix\\_info\\_t](#) structures (array of handles)

#### 16 IN ndirs

17 Number of elements in the *directives* array ([size\\_t](#))

#### 18 IN cbfunc

19 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

#### 20 IN cbdata

21 Data to be passed to the callback function (memory reference)

22 Returns one of the following:

- 23 • [PMIX\\_SUCCESS](#), indicating that the request is being processed - result will be returned in the  
24 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning  
25 from the API.
- 26 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
27 returned *success* - the *cbfunc* will *not* be called.
- 28 • [PMIX\\_ERR\\_NOT\\_SUPPORTED](#) The PMIx library does not support this operation - the *cbfunc*  
29 will *not* be called.
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the *cbfunc* will *not* be called.

32 If executed, the status returned in the provided callback function will be one of the following  
33 constants:

- 34 • [PMIX\\_SUCCESS](#) The operation succeeded - i.e., the [PMIX\\_GROUP\\_LEFT](#) event was generated.

- **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx library supports this operation, the host RM does not.
  - a non-zero PMIx error constant indicating a reason for the request's failure.

## Required Attributes

There are no identified required attributes for implementers.

## Description

Non-blocking version of the `PMIx_Group_leave` operation. The callback function will be called once the event has been locally generated and is not indicative of remote receipt.

## CHAPTER 14

# Fabric Support Definitions

---

As the drive for performance continues, interest has grown in scheduling algorithms that take into account network locality of the allocated resources and in optimizing collective communication patterns by structuring them to follow fabric topology. In addition, concerns over the time required to initiate execution of parallel applications and enable communication across them have grown as the size of those applications extends into the hundreds of thousands of individual processes spanning tens of thousands of nodes.

PMIx supports the communication part of these efforts by defining data types and attributes by which fabric endpoints and coordinates for processes and devices can be obtained from the host environment. When used in conjunction with other PMIx methods described in Chapter 16, this results in the ability of a process to obtain the fabric endpoint and coordinate of all other processes without incurring additional overhead associated with a global exchange of that information. This includes:

- Defining several interfaces specifically intended to support WLMs by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric.
- Supporting hierarchical collective operations by providing the fabric coordinates for all devices on participating nodes as well as a list of the peers sharing each fabric switch. This enables one, for example, to aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches based on detailed knowledge of the fabric location of each participant.
- Enabling the "*instant on*" paradigm to mitigate the scalable launch problem by providing each process with a rich set of information about the environment and the application, including everything required for communication between peers within the application, at time of process start of execution.

Meeting these needs in the case where only a single fabric device exists on each node is relatively straightforward - PMIx and the host environment provide a single endpoint for each process plus a coordinate for the device on each node, and there is no uncertainty regarding the endpoint each process will use. Extending this to the multiple device per node case is more difficult as the choice of endpoint by any given process cannot be known in advance, and questions arise regarding reachability between devices on different nodes. Resolving these ambiguities without requiring a global operation requires that PMIx provide both (a) an endpoint for each application process on each of its local devices; and (b) the fabric coordinates of all remote and local devices on participating nodes. It also requires that each process open all of its assigned endpoints as the endpoint selected for contact by a remote peer cannot be known in advance.

1 While these steps ensure the ability of a process to connect to a remote peer, it leaves unanswered  
2 the question of selecting the *preferred* device for that communication. If multiple devices are  
3 present on a node, then the application can benefit from having each process utilize its "closest"  
4 fabric device (i.e., the device that minimizes the communication distance between the process'  
5 location and that device) for messaging operations. In some cases, messaging libraries prefer to  
6 also retain the ability to use non-nearest devices, prioritizing the devices based on distance to  
7 support multi-device operations (e.g., for large message transmission in parallel).

8 PMIx supports this requirement by providing the array of process-to-device distance information  
9 for each process and local fabric device at start of execution. Both minimum and maximum  
10 distances are provided since a single process can occupy multiple processor locations. In addition,  
11 since processes can relocate themselves by changing their processor bindings, PMIx provides an  
12 API that allows the process to dynamically request an update to its distance array.

13 However, while these measures assist a process in selecting its own best endpoint, they do not  
14 resolve the uncertainty over the choice of preferred device by a remote peer. There are two methods  
15 by which this ambiguity can be resolved:

- 16 a) A process can select a remote endpoint to use based on its own preferred device and reachability  
17 of the peer's remote devices. Once the initial connection has been made, the two processes can  
18 exchange information and mutually determine their desired communication path going forward.
- 19 b) The application can use knowledge of both the local and remote distance arrays to compute the  
20 best communication path and establish that connection. In some instances (e.g., a homogeneous  
21 system), a PMIx server may provide distance information for both local and remote devices.  
22 Alternatively, when this isn't available, an application can opt to collect the information using  
23 the **PMIX\_COLLECT\_GENERATED\_JOB\_INFO** with the **PMIx\_Fence** API, or can obtain it  
24 on a one peer-at-a-time basis using the **PMIx\_Get** API on systems where the host environment  
25 supports the *Direct Mode* operation.

26 Information on fabric coordinates, endpoints, and device distances are provided as *reserved keys* as  
27 detailed in Chapter 6 - i.e., they are to be available at client start of execution and are subject to the  
28 retrieval rules of Section 6.2. Examples for retrieving fabric-related information include retrieval of:

- 29 • An array of information on fabric devices for a node by passing **PMIX\_FABRIC\_DEVICES** as  
30 the key to **PMIx\_Get** along with the **PMIX\_HOSTNAME** of the node as a directive
- 31 • An array of information on a specific fabric device by passing **PMIX\_FABRIC\_DEVICE** as the  
32 key to **PMIx\_Get** along with the **PMIX\_FABRIC\_DEVICE\_ID** of the device as a directive
- 33 • An array of information on a specific fabric device by passing **PMIX\_FABRIC\_DEVICE** as the  
34 key to **PMIx\_Get** along with both **PMIX\_FABRIC\_DEVICE\_NAME** of the device and the  
35 **PMIX\_HOSTNAME** of the node as directives

36 When requesting data on a device, returned data must include at least the following attributes:

- 37 • **PMIX\_HOSTNAME** "pmix.hname" (**char\***)

- 1                   Name of the host, as returned by the `gethostname` utility or its equivalent. The  
 2                   `PMIX_NODEID` may be returned in its place, or in addition to the hostname.
- 3     • **`PMIX_FABRIC_DEVICE_ID`** "pmix.fabdev.id" (**`string`**)  
 4                   System-wide Universally Unique IDentifier (UUID) of a particular fabric device.
- 5     • **`PMIX_FABRIC_DEVICE_NAME`** "pmix.fabdev.nm" (**`string`**)  
 6                   The operating system name associated with the device. This may be a logical fabric  
 7                   interface name (e.g. "eth0" or "eno1") or an absolute filename.
- 8     • **`PMIX_FABRIC_DEVICE_VENDOR`** "pmix.fabdev.vndr" (**`string`**)  
 9                   Indicates the name of the vendor that distributes the device.
- 10    • **`PMIX_FABRIC_DEVICE_BUS_TYPE`** "pmix.fabdev.btyp" (**`string`**)  
 11                  The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
- 12    • **`PMIX_FABRIC_DEVICE_PCI_DEVID`** "pmix.fabdev.pcidevid" (**`string`**)  
 13                  A node-level unique identifier for a Peripheral Component Interconnect (PCI) device.  
 14                  Provided only if the device is located on a PCI bus. The identifier is constructed as a  
 15                  four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit  
 16                  device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus,  
 17                  an example identifier might be "abc1:0f:23:01". The combination of node identifier  
 18                  (`PMIX_HOSTNAME` or `PMIX_NODEID`) and `PMIX_FABRIC_DEVICE_PCI_DEVID`  
 19                  shall be unique within the overall system. This item should be included if the device bus  
 20                  type is PCI - the equivalent should be provided for any other bus type.
- 21       The returned array may optionally contain one or more of the following in addition to the above list:
- 22     • **`PMIX_FABRIC_DEVICE_INDEX`** "pmix.fabdev.idx" (**`uint32_t`**)  
 23                  Index of the device within an associated communication cost matrix.
- 24     • **`PMIX_FABRIC_DEVICE_VENDORID`** "pmix.fabdev.vendid" (**`string`**)  
 25                  This is a vendor-provided identifier for the device or product.
- 26     • **`PMIX_FABRIC_DEVICE_DRIVER`** "pmix.fabdev.driver" (**`string`**)  
 27                  The name of the driver associated with the device.
- 28     • **`PMIX_FABRIC_DEVICE_FIRMWARE`** "pmix.fabdev.fmwr" (**`string`**)  
 29                  The device's firmware version.
- 30     • **`PMIX_FABRIC_DEVICE_ADDRESS`** "pmix.fabdev.addr" (**`string`**)  
 31                  The primary link-level address associated with the device, such as a Media Access  
 32                  Control (MAC) address. If multiple addresses are available, only one will be reported.
- 33     • **`PMIX_FABRIC_DEVICE_COORDINATES`** "pmix.fab.coord" (**`pmix_geometry_t`**)  
 34                  The `pmix_geometry_t` fabric coordinates for the device, including values for all  
 35                  supported coordinate views.

- **PMIX\_FABRIC\_DEVICE\_MTU** "pmix.fabdev.mtu" (**size\_t**)  
The maximum transfer unit of link level frames or packets, in bytes.
- **PMIX\_FABRIC\_DEVICE\_SPEED** "pmix.fabdev.speed" (**size\_t**)  
The active link data rate, given in bits per second.
- **PMIX\_FABRIC\_DEVICE\_STATE** "pmix.fabdev.state" (**pmix\_link\_state\_t**)  
The last available physical port state for the specified device. Possible values are **PMIX\_LINK\_STATE\_UNKNOWN**, **PMIX\_LINK\_DOWN**, and **PMIX\_LINK\_UP**, to indicate if the port state is unknown or not applicable (unknown), inactive (down), or active (up).
- **PMIX\_FABRIC\_DEVICE\_TYPE** "pmix.fabdev.type" (**string**)  
Specifies the type of fabric interface currently active on the device, such as Ethernet or InfiniBand.

The remainder of this chapter details the events, data types, attributes, and APIs associated with fabric-related operations.

## 14.1 Fabric Support Events

The following events are defined for use in fabric-related operations.

- PMIX\_FABRIC\_UPDATE\_PENDING** The PMIx server library has been alerted to a change in the fabric that requires updating of one or more registered **pmix\_fabric\_t** objects.
- PMIX\_FABRIC\_UPDATED** The PMIx server library has completed updating the entries of all affected **pmix\_fabric\_t** objects registered with the library. Access to the entries of those objects may now resume.
- PMIX\_FABRIC\_UPDATE\_ENDPOINTS** Endpoint assignments have been updated, usually in response to migration or restart of a process. Clients should use **PMIx\_Get** to update any internally cached connections.

## 14.2 Fabric Support Datatypes

Several datatype definitions have been created to support fabric-related operations and information.

### 14.2.1 Fabric Endpoint Structure

The **pmix\_endpoint\_t** structure contains an assigned endpoint for a given fabric device.

PMIx v4.0

C

```
29     typedef struct pmix_endpoint {
30         char *uuid;
31         pmix_byte_object_t endpt;
32     } pmix_endpoint_t;
```



1      The *uuid* field contains the UUID of the fabric device and the *endpt* field contains a fabric  
2      vendor-specific object identifying the communication endpoint assigned to the process.

### 3    14.2.2 Fabric endpoint support macros

4      The following macros are provided to support the `pmix_endpoint_t` structure.

#### 5      Initialize the endpoint structure

6      Initialize the `pmix_endpoint_t` fields.

PMIx v4.0

7      `PMIX_ENDPOINT_CONSTRUCT(m)`

8      IN    m

9      Pointer to the structure to be initialized (pointer to `pmix_endpoint_t`)

#### 10     Destruct the endpoint structure

11     Destruct the `pmix_endpoint_t` fields.

PMIx v4.0

12     `PMIX_ENDPOINT_DESTRUCT(m)`

13     IN    m

14     Pointer to the structure to be destructed (pointer to `pmix_endpoint_t`)

#### 15     Create an endpoint array

16     Allocate and initialize a `pmix_endpoint_t` array.

PMIx v4.0

17     `PMIX_ENDPOINT_CREATE(m, n)`

18     INOUT m

19     Address where the pointer to the array of `pmix_endpoint_t` structures shall be stored  
20     (handle)

21     IN    n

22     Number of structures to be allocated (`size_t`)

1      **Release an endpoint array**  
2      Release an array of `pmix_endpoint_t` structures.

PMIx v4.0

C

3      `PMIX_ENDPOINT_FREE(m, n)`

C

4      **IN m**  
5      Pointer to the array of `pmix_endpoint_t` structures (handle)  
6      **IN n**  
7      Number of structures in the array (`size_t`)

### 14.2.3 Fabric Device Distance Structure

The `pmix_device_distance_t` structure contains the minimum and maximum relative distance from the caller to a given fabric device.

PMIx v4.0

C

```
11     typedef struct pmix_device_distance {  
12        char *uuid;  
13        uint16_t mindist;  
14        uint16_t maxdist;  
15     } pmix_device_distance_t;
```

C

The two distance fields provide the minimum and maximum relative distance to the device from the binding location (as sampled at the time of the request) of the process, expressed as a 16-bit integer value where a smaller number indicates that this device is closer to the process than a device with a larger distance value.

Relative distances only apply to similar devices (i.e., devices from the same fabric) and cannot be used to compare devices from different fabrics. Both minimum and maximum distances are provided to support cases where the process may be bound to more than one location, and the locations are at different distances from the device.

A relative distance value of `UINT16_MAX` indicates that the distance from the process to the device could not be provided. This may be due to lack of available information (e.g., the PMIx library not having access to device locations) or other factors.

### 14.2.4 Fabric device distance support macros

The following macros are provided to support the `pmix_device_distance_t` structure.

1           **Initialize the device distance structure**  
2        Initialize the `pmix_device_distance_t` fields.  
3         `PMIX_DEVICE_DIST_CONSTRUCT(m)`   
4        **IN m**  
5        Pointer to the structure to be initialized (pointer to `pmix_device_distance_t`)  
6           **Destruct the device distance structure**  
7        Destruct the `pmix_device_distance_t` fields.  
8         `PMIX_DEVICE_DIST_DESTRUCT(m)`   
9        **IN m**  
10      Pointer to the structure to be destructed (pointer to `pmix_device_distance_t`)  
11           **Create an device distance array**  
12      Allocate and initialize a `pmix_device_distance_t` array.  
13         `PMIX_DEVICE_DIST_CREATE(m, n)`   
14        **INOUT m**  
15      Address where the pointer to the array of `pmix_device_distance_t` structures shall be  
16      stored (handle)  
17        **IN n**  
18      Number of structures to be allocated (`size_t`)  
19           **Release an device distance array**  
20      Release an array of `pmix_device_distance_t` structures.  
21         `PMIX_DEVICE_DIST_FREE(m, n)`   
22        **IN m**  
23      Pointer to the array of `pmix_device_distance_t` structures (handle)  
24        **IN n**  
25      Number of structures in the array (`size_t`)

## 14.2.5 Fabric Coordinate Structure

2 The `pmix_coord_t` structure describes the fabric coordinates of a specified device in a given  
3 view.

```
4     typedef struct pmix_coord {
5         pmix_coord_view_t view;
6         uint32_t *coord;
7         size_t dims;
8     } pmix_coord_t;
```

9 All coordinate values shall be expressed as unsigned integers due to their units being defined in  
10 fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity  
11 and not relative communication distance.

### Advice to PMIx library implementers

12 Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the  
13 coordinate data is to be stored within the PMIx library. Implementers are free to store the  
14 coordinate in whatever format they choose.

15 A fabric coordinate is associated with a given fabric device and must be unique within a given view.  
16 Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates  
17 are logically grouped within the *node* realm (as described in Section 6.1) and can be retrieved per  
18 the rules detailed in Section 6.1.5.

## 14.2.6 Fabric coordinate support macros

20 The following macros are provided to support the `pmix_coord_t` structure.

### 21 Initialize the coord structure

22 Initialize the `pmix_coord_t` fields.

PMIx v4.0

```
23     PMIX_COORD_CONSTRUCT(m)
```

24 IN m

25 Pointer to the structure to be initialized (pointer to `pmix_coord_t`)

1           **Destruct the coord structure**

2           Destruct the `pmix_coord_t` fields.

PMIx v4.0

C

3           `PMIX_COORD_DESTROY(m)`

C

4           **IN    m**

5           Pointer to the structure to be destructed (pointer to `pmix_coord_t`)

6           **Create a coord array**

7           Allocate and initialize a `pmix_coord_t` array.

PMIx v4.0

C

8           `PMIX_COORD_CREATE(m, n)`

C

9           **INOUT m**

10          Address where the pointer to the array of `pmix_coord_t` structures shall be stored (handle)

11          **IN    n**

12          Number of structures to be allocated (`size_t`)

13          **Release a coord array**

14          Release an array of `pmix_coord_t` structures.

PMIx v4.0

C

15          `PMIX_COORD_FREE(m, n)`

C

16          **IN    m**

17          Pointer to the array of `pmix_coord_t` structures (handle)

18          **IN    n**

19          Number of structures in the array (`size_t`)

20      **14.2.7 Fabric Geometry Structure**

21          The `pmix_geometry_t` structure describes the fabric coordinates of a specified device.

PMIx v4.0

C

```
22         typedef struct pmix_geometry {
23             size_t fabric;
24             char *uuid;
25             pmix_coord_t *coordinates;
26             size_t ncoords;
27         } pmix_geometry_t;
```

C

1 All coordinate values shall be expressed as unsigned integers due to their units being defined in  
2 fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity  
3 and not relative communication distance.

#### Advice to PMIx library implementers

4 Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the  
5 coordinate data is to be stored within the PMIx library. Implementers are free to store the  
6 coordinate in whatever format they choose.

7 A fabric coordinate is associated with a given fabric device and must be unique within a given view.  
8 Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates  
9 are logically grouped within the *node* realm (as described in Section 6.1) and can be retrieved per  
10 the rules detailed in Section 6.1.5.

### 14.2.8 Fabric geometry support macros

12 The following macros are provided to support the `pmix_geometry_t` structure.

#### 13 Initialize the geometry structure

14 Initialize the `pmix_geometry_t` fields.

PMIx v4.0

C

15 `PMIX_GEOMETRY_CONSTRUCT(m)`

C

16 IN m

17 Pointer to the structure to be initialized (pointer to `pmix_geometry_t`)

#### 18 Destruct the geometry structure

19 Destruct the `pmix_geometry_t` fields.

PMIx v4.0

C

20 `PMIX_GEOMETRY_DESTRUCT(m)`

C

21 IN m

22 Pointer to the structure to be destructed (pointer to `pmix_geometry_t`)

```

1 Create a geometry array
2 Allocate and initialize a pmix_geometry_t array.
3 PMIx v4.0 ━━━━━━ C ━━━━━━
4 PMIX_GEOMETRY_CREATE (m, n) ━━━━━━ C ━━━━━━
5 INOUT m
6 Address where the pointer to the array of pmix_geometry_t structures shall be stored
7 (handle)
8 IN n
9 Number of structures to be allocated (size_t)
10 Release a geometry array
11 Release an array of pmix_geometry_t structures.
12 PMIx v4.0 ━━━━━━ C ━━━━━━
13 PMIX_GEOMETRY_FREE (m, n) ━━━━━━ C ━━━━━━
14 IN m
15 Pointer to the array of pmix_geometry_t structures (handle)
16 IN n
17 Number of structures in the array (size_t)

```

## 14.2.9 Fabric Coordinate Views

```

1 PMIx v4.0 ━━━━━━ C ━━━━━━
2 typedef uint8_t pmix_coord_view_t;
3 #define PMIX_COORD_VIEW_UNDEF 0x00
4 #define PMIX_COORD_LOGICAL_VIEW 0x01
5 #define PMIX_COORD_PHYSICAL_VIEW 0x02
6 ━━━━━━ C ━━━━━━

```

21 Fabric coordinates can be reported based on different *views* according to user preference at the time  
22 of request. The following views have been defined:

```

23 PMIX_COORD_VIEW_UNDEF The coordinate view has not been defined.
24 PMIX_COORD_LOGICAL_VIEW The coordinates are provided in a logical view, typically
25 given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by
26 the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains,
27 and other similar factors employed by that fabric.
28 PMIX_COORD_PHYSICAL_VIEW The coordinates are provided in a physical view based on
29 the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative
30 position of that interface on the specific fabric cabling.

```

31 If the requester does not specify a view, coordinates shall default to the *logical* view.

## 14.2.10 Fabric Link State

2 The `pmix_link_state_t` is a `uint32_t` type for fabric link states.

PMIx v4.0

C

3 `typedef uint8_t pmix_link_state_t;`

C

4 The following constants can be used to set a variable of the type `pmix_link_state_t`. All  
5 definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state  
6 values start at zero.

7 `PMIX_LINK_STATE_UNKNOWN` The port state is unknown or not applicable.

8 `PMIX_LINK_DOWN` The port is inactive.

9 `PMIX_LINK_UP` The port is active.

## 14.2.11 Fabric Operation Constants

11 PMIx v4.0 The `pmix_fabric_operation_t` data type is an enumerated type for specifying fabric  
12 operations used in the PMIx server module's `pmix_server_fabric_fn_t` API.

13 `PMIX_FABRIC_REQUEST_INFO` Request information on a specific fabric - if the fabric isn't  
14 specified as per `PMIx_Fabric_register`, then return information on the default fabric of  
15 the overall system. Information to be returned is described in `pmix_fabric_t`.

16 `PMIX_FABRIC_UPDATE_INFO` Update information on a specific fabric - the index of the  
17 fabric (`PMIX_FABRIC_INDEX`) to be updated must be provided.

## 14.2.12 Fabric registration structure

19 The `pmix_fabric_t` structure is used by a WLM to interact with fabric-related PMIx interfaces,  
20 and to provide information about the fabric for use in scheduling algorithms or other purposes.

PMIx v4.0

C

21 `typedef struct pmix_fabric_s {`  
22     `char *name;`  
23     `size_t index;`  
24     `pmix_info_t *info;`  
25     `size_t ninfo;`  
26     `void *module;`  
27 } pmix\_fabric\_t;;

1 Note that in this structure:

- 2 • *name* is an optional user-supplied string name identifying the fabric being referenced by this
- 3 struct. If provided, the field must be a **NULL**-terminated string composed of standard
- 4 alphanumeric values supported by common utilities such as *strcmp*;
- 5 • *index* is a PMIx-provided number identifying this object;
- 6 • *info* is an array of **pmix\_info\_t** containing information (provided by the PMIx library) about
- 7 the fabric;
- 8 • *ninfo* is the number of elements in the *info* array;
- 9 • *module* points to an opaque object reserved for use by the PMIx server library.

10 Note that only the *name* field is provided by the user - all other fields are provided by the PMIx  
 11 library and must not be modified by the user. The *info* array contains a varying amount of  
 12 information depending upon both the PMIx implementation and information available from the  
 13 fabric vendor. At a minimum, it must contain (ordering is arbitrary):

### Required Attributes

- 14 **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (**string**)  
 15 Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.
- 16 **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (**string**)  
 17 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).
- 18 **PMIX\_FABRIC\_NUM\_DEVICES** "pmix.fab.nverts" (**size\_t**)  
 19 Total number of fabric devices in the overall system - corresponds to the number of rows or  
 20 columns in the cost matrix.

21 and may optionally contain one or more of the following:

### Optional Attributes

- 22 **PMIX\_FABRIC\_COST\_MATRIX** "pmix.fab.cm" (**pointer**)  
 23 Pointer to a two-dimensional square array of point-to-point relative communication costs  
 24 expressed as **uint16\_t** values.
- 25 **PMIX\_FABRIC\_GROUPS** "pmix.fab.grp" (**string**)  
 26 A string delineating the group membership of nodes in the overall system, where each fabric  
 27 group consists of the group number followed by a colon and a comma-delimited list of nodes  
 28 in that group, with the groups delimited by semi-colons (e.g.,  
 29 **0:node000, node002, node004, node006;1:node001, node003,**  
 30 **node005, node007**)
- 31 **PMIX\_FABRIC\_DIMS** "pmix.fab.dims" (**uint32\_t**)

```

1 Number of dimensions in the specified fabric plane/view. If no plane is specified in a
2 request, then the dimensions of all planes in the overall system will be returned as a
3 pmix_data_array_t containing an array of uint32_t values. Default is to provide
4 dimensions in logical view.

5 PMIX_FABRIC_PLANE "pmix.fab.plane" (string)
6 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
7 for information, specifies the plane whose information is to be returned. When used directly
8 as a key in a request, returns a pmix_data_array_t of string identifiers for all fabric
9 planes in the overall system.

10 PMIX_FABRIC_SHAPE "pmix.fab.shape" (pmix_data_array_t*)
11 The size of each dimension in the specified fabric plane/view, returned in a
12 pmix_data_array_t containing an array of uint32_t values. The size is defined as
13 the number of elements present in that dimension - e.g., the number of devices in one
14 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of
15 each plane in the overall system will be returned in a pmix_data_array_t array where
16 each element is itself a two-element array containing the PMIX_FABRIC_PLANE followed
17 by that plane's fabric shape. Default is to provide the shape in logical view.

18 PMIX_FABRIC_SHAPE_STRING "pmix.fab.shapestr" (string)
19 Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the
20 shape of each plane in the overall system will be returned in a pmix_data_array_t array
21 where each element is itself a two-element array containing the PMIX_FABRIC_PLANE
22 followed by that plane's fabric shape string. Default is to provide the shape in logical view.

23 While unusual due to scaling issues, implementations may include an array of
24 PMIX_FABRIC_DEVICE elements describing the device information for each device in the
25 overall system. Each element shall contain a pmix_data_array_t of pmix_info_t values
26 describing the device. Each array may contain one or more of the following (ordering is arbitrary):

27 PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string)
28 The operating system name associated with the device. This may be a logical fabric interface
29 name (e.g. "eth0" or "eno1") or an absolute filename.

30 PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string)
31 Indicates the name of the vendor that distributes the device.

32 PMIX_FABRIC_DEVICE_ID "pmix.fabdev.id" (string)
33 System-wide UUID of a particular fabric device.

34 PMIX_HOSTNAME "pmix.hname" (char*)
35 Name of the host, as returned by the gethostname utility or its equivalent.

36 PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string)
37 The name of the driver associated with the device.

38 PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string)

```

```

1      The device's firmware version.
2      PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)
3      The primary link-level address associated with the device, such as a MAC address. If
4      multiple addresses are available, only one will be reported.
5      PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
6      The maximum transfer unit of link level frames or packets, in bytes.
7      PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
8      The active link data rate, given in bits per second.
9      PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" (pmix_link_state_t)
10     The last available physical port state for the specified device. Possible values are
11     PMIX_LINK_STATE_UNKNOWN, PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate
12     if the port state is unknown or not applicable (unknown), inactive (down), or active (up).
13     PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
14     Specifies the type of fabric interface currently active on the device, such as Ethernet or
15     InfiniBand.
16     PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
17     The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
18     PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
19     A node-level unique identifier for a PCI device. Provided only if the device is located on a
20     PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
21     the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
22     zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
23     combination of node identifier (PMIX_HOSTNAME or PMIX_NODEID) and
24     PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the overall system.

```



### 25 14.2.12.1 Initialize the fabric structure

26 Initialize the **pmix\_fabric\_t** fields.

27 *PMIx v4.0*

28 **PMIX\_FABRIC\_CONSTRUCT** (*m*)

29 **IN** *m*

Pointer to the structure to be initialized (pointer to **pmix\_fabric\_t**)

## 14.3 Fabric Support Attributes

The following attribute is used by the PMIx server library supporting the system's WLM to indicate that it wants access to the fabric support functions:

```
4 PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
5     Server requests access to WLM-supporting features - passed solely to the
6     PMIx_server_init API to indicate that the library is to be initialized for scheduler
7     support.
```

The following attributes may be returned in response to fabric-specific APIs or queries (e.g., **PMIx\_Get** or **PMIx\_Query\_info**). These attributes are not related to a specific *data realm* (as described in Section 6.1) - the **PMIx\_Get** function shall therefore ignore the value in its *proc* process identifier argument when retrieving these values.

```
12 PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer)
13     Pointer to a two-dimensional square array of point-to-point relative communication costs
14     expressed as uint16_t values.
15 PMIX_FABRIC_GROUPS "pmix.fab.grp" (string)
16     A string delineating the group membership of nodes in the overall system, where each fabric
17     group consists of the group number followed by a colon and a comma-delimited list of nodes
18     in that group, with the groups delimited by semi-colons (e.g.,
19     0:node000,node002,node004,node006;1:node001,node003,
20     node005,node007)
21 PMIX_FABRIC_PLANE "pmix.fab.plane" (string)
22     ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
23     for information, specifies the plane whose information is to be returned. When used directly
24     as a key in a request, returns a pmix_data_array_t of string identifiers for all fabric
25     planes in the overall system.
26 PMIX_FABRIC_SWITCH "pmix.fab.switch" (string)
27     ID string of a fabric switch. When used as a modifier in a request for information, specifies
28     the switch whose information is to be returned. When used directly as a key in a request,
29     returns a pmix_data_array_t of string identifiers for all fabric switches in the overall
30     system.
```

The following attributes may be returned in response to queries (e.g., **PMIx\_Get** or **PMIx\_Query\_info**). A qualifier (e.g., **PMIX\_FABRIC\_INDEX**) identifying the fabric whose value is being referenced must be provided for queries on systems supporting more than one fabric when values for the non-default fabric are requested. These attributes are not related to a specific *data realm* (as described in Section 6.1) - the **PMIx\_Get** function shall therefore ignore the value in its *proc* process identifier argument when retrieving these values.

```
37 PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string)
38     Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.
39 PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)
```

1 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).  
 2 **PMIX\_FABRIC\_INDEX** "pmix.fab.idx" (**size\_t**)  
 3 The index of the fabric as returned in **pmix\_fabric\_t**.  
 4 **PMIX\_FABRIC\_NUM\_DEVICES** "pmix.fab.nverts" (**size\_t**)  
 5 Total number of fabric devices in the overall system - corresponds to the number of rows or  
 6 columns in the cost matrix.  
 7 **PMIX\_FABRIC\_VIEW** "pmix.fab.view" (**pmix\_coord\_view\_t**)  
 8 Used purely as a qualifier to requests, specifies the view type (e.g., local vs. physical) for the  
 9 requested information.  
 10 **PMIX\_FABRIC\_DIMS** "pmix.fab.dims" (**uint32\_t**)  
 11 Number of dimensions in the specified fabric plane/view. If no plane is specified in a  
 12 request, then the dimensions of all planes in the overall system will be returned as a  
 13 **pmix\_data\_array\_t** containing an array of **uint32\_t** values. Default is to provide  
 14 dimensions in *logical* view.  
 15 **PMIX\_FABRIC\_SHAPE** "pmix.fab.shape" (**pmix\_data\_array\_t\***)  
 16 The size of each dimension in the specified fabric plane/view, returned in a  
 17 **pmix\_data\_array\_t** containing an array of **uint32\_t** values. The size is defined as  
 18 the number of elements present in that dimension - e.g., the number of devices in one  
 19 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of  
 20 each plane in the overall system will be returned in a **pmix\_data\_array\_t** array where  
 21 each element is itself a two-element array containing the **PMIX\_FABRIC\_PLANE** followed  
 22 by that plane's fabric shape. Default is to provide the shape in *logical* view.  
 23 **PMIX\_FABRIC\_SHAPE\_STRING** "pmix.fab.shapestr" (**string**)  
 24 Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the  
 25 shape of each plane in the overall system will be returned in a **pmix\_data\_array\_t** array  
 26 where each element is itself a two-element array containing the **PMIX\_FABRIC\_PLANE**  
 27 followed by that plane's fabric shape string. Default is to provide the shape in *logical* view.

28 The following attributes are related to the *node realm* (as described in Section 6.1.5) and are  
 29 retrieved according to those rules.

30 **PMIX\_FABRIC\_DEVICES** "pmix.fab.devs" (**pmix\_data\_array\_t**)  
 31 Array of **pmix\_info\_t** containing information for all devices on the specified node. Each  
 32 element of the array will contain a **PMIX\_FABRIC\_DEVICE** entry, which in turn will  
 33 contain an array of information on a given device.  
 34 **PMIX\_FABRIC\_COORDINATES** "pmix.fab.coords" (**pmix\_data\_array\_t**)  
 35 Array of **pmix\_geometry\_t** fabric coordinates for devices on the specified node. The  
 36 array will contain the coordinates of all devices on the node, including values for all  
 37 supported coordinate views. The information for devices on the local node shall be provided  
 38 if the node is not specified in the request.  
 39 **PMIX\_FABRIC\_DEVICE** "pmix.fabdev" (**pmix\_data\_array\_t**)  
 40 An array of **pmix\_info\_t** describing a particular fabric device using one or more of the  
 41 attributes defined below. The first element in the array shall be the  
 42 **PMIX\_FABRIC\_DEVICE\_ID** of the device.

```

1   PMIX_FABRIC_DEVICE_ID "pmix.fabdev.id" (string)
2     System-wide UUID of a particular fabric device.
3   PMIX_FABRIC_DEVICE_INDEX "pmix.fabdev.idx" (uint32_t)
4     Index of the device within an associated communication cost matrix.
5   PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string)
6     The operating system name associated with the device. This may be a logical fabric interface
7     name (e.g. "eth0" or "eno1") or an absolute filename.
8   PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string)
9     Indicates the name of the vendor that distributes the device.
10  PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
11    The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
12  PMIX_FABRIC_DEVICE_VENDORID "pmix.fabdev.vendid" (string)
13    This is a vendor-provided identifier for the device or product.
14  PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string)
15    The name of the driver associated with the device.
16  PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string)
17    The device's firmware version.
18  PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)
19    The primary link-level address associated with the device, such as a MAC address. If
20    multiple addresses are available, only one will be reported.
21  PMIX_FABRIC_DEVICE_COORDINATES "pmix.fab.coord" (pmix_geometry_t)
22    The pmix_geometry_t fabric coordinates for the device, including values for all
23    supported coordinate views.
24  PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
25    The maximum transfer unit of link level frames or packets, in bytes.
26  PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
27    The active link data rate, given in bits per second.
28  PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" (pmix_link_state_t)
29    The last available physical port state for the specified device. Possible values are
30    PMIX_LINK_STATE_UNKNOWN, PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate
31    if the port state is unknown or not applicable (unknown), inactive (down), or active (up).
32  PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
33    Specifies the type of fabric interface currently active on the device, such as Ethernet or
34    InfiniBand.
35  PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
36    A node-level unique identifier for a PCI device. Provided only if the device is located on a
37    PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
38    the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
39    zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
40    combination of node identifier (PMIX_HOSTNAME or PMIX_NODEID) and
41    PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the overall system.

```

1 The following attributes are related to the *process realm* (as described in Section 6.1.4) and are  
2 retrieved according to those rules.

3 **PMIX\_FABRIC\_ENDPT** "pmix.fab.endpt" (pmix\_data\_array\_t)

4 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given  
5 process (e.g., in the case where multiple devices are associated with a package to which the  
6 process is bound), the returned values will be provided in a **pmix\_data\_array\_t** of  
7 **pmix\_endpoint\_t** elements.

8 **PMIX\_FABRIC\_DEVICE\_DIST** "pmix.fab.endptdist" (pmix\_data\_array\_t)

9 Relative distance from the location of the calling process (either as sampled at the time of a  
10 **PMIx\_Fabric\_update\_distances** request, or based on the initial binding location  
11 set at time of start of execution) to each local fabric device, returned as an array of  
12 **pmix\_device\_distance\_t** elements in no particular order.

13 The following attributes are related to the *job realm* (as described in Section 6.1.2) and are retrieved  
14 according to those rules.

15 **PMIX\_SWITCH\_PEERS** "pmix.speers" (pmix\_data\_array\_t)

16 Peer ranks that share the same switch as the process specified in the call to **PMIx\_Get**.  
17 Returns a **pmix\_data\_array\_t** array of **pmix\_info\_t** results, each element  
18 containing the **PMIX\_SWITCH\_PEERS** key with a three-element **pmix\_data\_array\_t**  
19 array of **pmix\_info\_t** containing the **PMIX\_FABRIC\_DEVICE\_ID** of the local fabric  
20 device, the **PMIX\_FABRIC\_SWITCH** identifying the switch to which it is connected, and a  
21 comma-delimited string of peer ranks sharing the switch to which that device is connected.

## 22 14.4 Fabric Support Functions

23 The following APIs allow the WLM to request specific services from the fabric subsystem via the  
24 PMIx library.

### Advice to PMIx server hosts

25 Due to their high cost in terms of execution, memory consumption, and interactions with other  
26 SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation  
27 of these APIs be restricted to a single PMIx server in a system that is supporting the SMS  
28 component responsible for the scheduling of allocations (i.e., the system *scheduler*). The  
29 **PMIX\_SERVER\_SCHEDULER** attribute can be used for this purpose to control the execution path.  
30 Clients, tools, and other servers utilizing these functions are advised to have their requests  
31 forwarded to the server supporting the scheduler using the **pmix\_server\_fabric\_fn\_t**  
32 server module function, as needed.

## 14.4.1 PMIx\_Fabric\_register

### Summary

Register for access to fabric-related information.

### Format

PMIx v4.0

```
5     pmix_status_t
6     PMIx_Fabric_register(pmix_fabric_t *fabric,
7                           const pmix_info_t directives[],
8                           size_t ndirs);
```

#### INOUT fabric

address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

#### IN directives

an optional array of values indicating desired behaviors and/or fabric to be accessed. If `NULL`, then the highest priority available fabric will be used (array of handles)

#### IN ndirs

Number of elements in the *directives* array (integer)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

### Required Attributes

The following directives are required to be supported by all PMIx libraries to aid users in identifying the fabric whose data is being sought:

`PMIX_FABRIC_PLANE` "pmix.fab.plane" (`string`)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in the overall system.

`PMIX_FABRIC_IDENTIFIER` "pmix.fab.id" (`string`)

An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

`PMIX_FABRIC_VENDOR` "pmix.fab.vndr" (`string`)

Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

1           **Description**

2       Register for access to fabric-related information, including the communication cost matrix. This  
3       call must be made prior to requesting information from a fabric. The caller may request access to a  
4       particular fabric using the vendor, type, or identifier, or to a specific *fabric plane* via the  
5       **PMIX\_FABRIC\_PLANE** attribute - otherwise, information for the default fabric will be returned.  
6       Upon successful completion of the call, information will have been filled into the fields of the  
7       provided *fabric* structure.

8       For performance reasons, the PMIx library does not provide thread protection for accessing the  
9       information in the **pmix\_fabric\_t** structure. Instead, the PMIx implementation shall provide  
10      two methods for coordinating updates to the provided fabric information:

- 11     • Users may periodically poll for updates using the **PMIx\_Fabric\_update** API
- 12     • Users may register for **PMIX\_FABRIC\_UPDATE\_PENDING** events indicating that an update to  
13       the cost matrix is pending. When received, users are required to terminate or pause any actions  
14       involving access to the cost matrix before returning from the event. Completion of the  
15       **PMIX\_FABRIC\_UPDATE\_PENDING** event handler indicates to the PMIx library that the fabric  
16       object's entries are available for updating. This may include releasing and re-allocating memory  
17       as the number of vertices may have changed (e.g., due to addition or removal of one or more  
18       devices). When the update has been completed, the PMIx library will generate a  
19       **PMIX\_FABRIC\_UPDATED** event indicating that it is safe to begin using the updated fabric  
20       object(s).

21       There is no requirement that the caller exclusively use either one of these options. For example, the  
22       user may choose to both register for fabric update events, but poll for an update prior to some  
23       critical operation.

24    **14.4.2 PMIx\_Fabric\_register\_nb**

25           **Summary**

26       Register for access to fabric-related information.

27           **Format**

28       PMIx v4.0

```
29       pmix_status_t
30       PMIx_Fabric_register_nb(pmix_fabric_t *fabric,
31                            const pmix_info_t directives[],
32                            size_t ndirs,
33                            pmix_op_cfunc_t cbfunc, void *cbdata);
```

33           **INPUT fabric**

34       address of a **pmix\_fabric\_t** (backed by storage). User may populate the "name" field at  
35       will - PMIx does not utilize this field (handle)

1           **IN** **directives**  
2        an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**,  
3        then the highest priority available fabric will be used (array of handles)  
4           **IN** **ndirs**  
5        Number of elements in the *directives* array (integer)  
6           **IN** **cbfunc**  
7        Callback function **pmix\_op\_cbfunc\_t** (function reference)  
8           **IN** **cbdata**  
9        Data to be passed to the callback function (memory reference)

10     >Returns one of the following:

11       • **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided  
12        callback function will be executed upon completion of the operation. Note that the library must  
13        not invoke the callback function prior to returning from the API.

14       • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this  
15        case, the provided callback function will not be executed

## 16     **Description**

17     Non-blocking form of **PMIx\_Fabric\_register**. The caller is not allowed to access the  
18     provided **pmix\_fabric\_t** until the callback function has been executed, at which time the fabric  
19     information will have been loaded into the provided structure.

### 20    **14.4.3 PMIx\_Fabric\_update**

#### 21    **Summary**

22    Update fabric-related information.

#### 23    **Format**

24    *PMIx v4.0*

25    

```
pmix_status_t
PMIx_Fabric_update(pmix_fabric_t *fabric);
```

#### 26    **INOUT fabric**

27       address of a **pmix\_fabric\_t** (backed by storage) (handle)

28    Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

#### 29    **Description**

30    Update fabric-related information. This call can be made at any time to request an update of the  
31    fabric information contained in the provided **pmix\_fabric\_t** object. The caller is not allowed to  
32    access the provided **pmix\_fabric\_t** until the call has returned. Upon successful return, the  
33    information fields in the *fabric* structure will have been updated.

## 14.4.4 PMIx\_Fabric\_update\_nb

### Summary

Update fabric-related information.

### Format

PMIx v4.0

```
5 pmix_status_t  
6 PMIx_Fabric_update_nb(pmix_fabric_t *fabric,  
7 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

#### INOUT fabric

address of a `pmix_fabric_t` (handle)

#### IN cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

#### IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

- `PMIX_SUCCESS` indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

### Description

Non-blocking form of `PMIx_Fabric_update`. The caller is not allowed to access the provided `pmix_fabric_t` until the callback function has been executed, at which time the fields in the provided *fabric* structure will have been updated.

## 14.4.5 PMIx\_Fabric\_deregister

### Summary

Deregister a fabric object.

### Format

PMIx v4.0

```
28 pmix_status_t  
29 PMIx_Fabric_deregister(pmix_fabric_t *fabric);
```

C

#### IN fabric

address of a `pmix_fabric_t` (handle)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

## Description

Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information (e.g., cost matrix) associated with it. Contents of the provided `pmix_fabric_t` will be invalidated upon function return.

#### 14.4.6 PMIx\_Fabric\_deregister\_nb

## Summary

Deregister a fabric object.

## Format

PMIx v4.0

IN fabric

address of a **pmix\_fabric\_t** (handle)

IN chfunc

Callback function **pmix op cbfunc t** (function reference)

IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
  - a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

## Description

Non-blocking form of [PMIx\\_Fabric\\_deregister](#). Provided *fabric* must not be accessed until after callback function has been executed.

#### 14.4.7 PMIx Fabric update distances

## Summary

Update distances from current process location to local fabric devices.

1           **Format**

2           *PMIx v4.0*      C  
3           `pmix_status_t`  
4           `PMIx_Fabric_update_distances(pmix_device_distance_t *distances[],`  
5                            `size_t *ndist);`

5           **INOUT distances**

6           Pointer to an address where the array of `pmix_device_distance_t` structures  
7           containing the distances from the caller to local fabric devices is to be returned (handle)

8           **INOUT ndist**

9           Pointer to an address where the number of elements in the *distances* array is to be returned  
10          (handle)

11         Returns one of the following:

- 12
  - `PMIX_SUCCESS` indicating that the distances were returned.
  - a non-zero PMIx error constant indicating the reason the request failed.

14         **Description**

15         Both the minimum and maximum distance fields in the elements of the array shall be filled with the  
16         respective distances between the current process location and the respective fabric device. Any  
17         distance information stored in the local PMIx server's cache should also be updated so that  
18         subsequent queries return the updated values.

19         

## 14.4.8 `PMIx_Fabric_update_distances_nb`

20         **Summary**

21         Update distances from current process location to local fabric devices.

22         **Format**

23           *PMIx v4.0*      C  
24           `pmix_status_t`  
25           `PMIx_Fabric_update_distances_nb(pmix_info_cfunc_t cbfunc,`  
26                            `void *cbdata);`

26         **IN cbfunc**

27         Callback function `pmix_info_cfunc_t` (function reference)

28         **IN cbdata**

29         Data to be passed to the callback function (memory reference)

30         Returns one of the following:

- 1     • **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided  
2       callback function will be executed upon completion of the operation. Note that the library must  
3       not invoke the callback function prior to returning from the API.  
4     • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this  
5       case, the provided callback function will not be executed

6     **Description**

7     Non-blocking form of the **PMIx\_Fabric\_update\_distances** API. If successful, the  
8       requested data will be returned under the **PMIX\_FABRIC\_DEVICE\_DIST** attribute in the *info*  
9       array of the callback function.

## CHAPTER 15

# Security

---

1 PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes.  
2 By definition, *client* processes must be preregistered with the PMIx server library via the  
3 **PMIx\_server\_register\_client** API before they are spawned. This API requires that the  
4 host pass the expected effective UID/GID of the client process.

5 When the client attempts to connect to the PMIx server, the server shall use available standard  
6 Operating System (OS) methods to determine the effective UID/GID of the process requesting the  
7 connection. PMIx implementations shall not rely on any values reported by the client process itself.  
8 The effective UID/GID reported by the OS is compared to the values provided by the host during  
9 registration - if the values fail to match, the PMIx server is required to drop the connection request.  
10 This ensures that the PMIx server does not allow connection from a client that doesn't at least meet  
11 some minimal security requirement.

12 Once the requesting client passes the initial test, the PMIx server can, at the choice of the  
13 implementor, perform additional security checks. This may involve a variety of methods such as  
14 exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server  
15 reports the client connection request to the host via the  
16 **pmix\_server\_client\_connected\_fn\_t** interface, if provided. The host may perform any  
17 additional checks and operations before responding with either **PMIX\_SUCCESS** to indicate that  
18 the connection is approved, or a PMIx error constant indicating that the connection request is  
19 refused. In this latter case, the PMIx server is required to drop the connection.

20 Tools started by the host environment are classed as a subgroup of client processes and follow the  
21 client process procedure. However, tools that are not started by the host environment must be  
22 handled differently as registration information is not available prior to the connection request. In  
23 these cases, the PMIx server library is required to use available standard OS methods to get the  
24 effective UID/GID of the tool and report them upwards as part of invoking the  
25 **pmix\_server\_tool\_connection\_fn\_t** interface, deferring initial security screening to the  
26 host. Host environments willing to accept tool connections must therefore both explicitly enable  
27 them via the **PMIX\_SERVER\_TOOL\_SUPPORT** attribute, thereby confirming acceptance of the  
28 authentication and authorization burden, and provide the  
29 **pmix\_server\_tool\_connection\_fn\_t** server module function pointer.

## 30 15.1 Obtaining Credentials

31 Applications and tools often interact with the host environment in ways that require security beyond  
32 just verifying the user's identity - e.g., access to that user's relevant authorizations. This is

1 particularly important when tools connect directly to a system-level PMIx server that may be  
2 operating at a privileged level. A variety of system management software packages provide  
3 authorization services, but the lack of standardized interfaces makes portability problematic.

4 This section defines two PMIx client-side APIs for this purpose. These are most likely to be used  
5 by user-space applications/tools, but are not restricted to that realm.

## 6 15.1.1 PMIx\_Get\_credential

### 7 Summary

8 Request a credential from the PMIx server library or the host environment.

### 9 Format

PMIx v3.0

C

```
10 pmix_status_t
11 PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,
12                         pmix_byte_object_t *credential);
```

C

13 IN info

14 Array of `pmix_info_t` structures (array of handles)

15 IN ninfo

16 Number of elements in the *info* array (`size_t`)

17 IN credential

18 Address of a `pmix_byte_object_t` within which to return credential (handle)

19 Returns one of the following:

- 20 • `PMIX_SUCCESS`, indicating that the credential has been returned in the provided  
21 `pmix_byte_object_t`
- 22 • a PMIx error constant indicating either an error in the input or that the request is unsupported

### Required Attributes

23 There are no required attributes for this API. Note that implementations may choose to internally  
24 execute integration for some security environments (e.g., directly contacting a *munge* server).

25 Implementations that support the operation but cannot directly process the client's request must  
26 pass any attributes that are provided by the client to the host environment for processing. In  
27 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
28 library to the host environment:

```
29 PMIX_USERID "pmix.euid" (uint32_t)
30             Effective user ID of the connecting process.
31 PMIX_GRP_ID "pmix.egid" (uint32_t)
32             Effective group ID of the connecting process.
```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX\_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

Request a credential from the PMIx server library or the host environment. The credential is returned as a **pmix\_byte\_object\_t** to support potential binary formats - it is therefore opaque to the caller. No information as to the source of the credential is provided.

### 15.1.2 PMIx\_Get\_credential\_nb

#### Summary

Request a credential from the PMIx server library or the host environment.

#### Format

PMIx v3.0

C

```
14     pmix_status_t  
15     PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,  
16                               pmix_credential_cbfunc_t cbfunc,  
17                               void *cbdata);
```

C

IN **info**

Array of **pmix\_info\_t** structures (array of handles)

IN **ninfo**

Number of elements in the *info* array (**size\_t**)

IN **cbfunc**

Callback function to return credential (**pmix\_credential\_cbfunc\_t** function reference)

IN **cbdata**

Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided *cbfunc*
- a PMIx error constant indicating either an error in the input or that the request is unsupported - the *cbfunc* will *not* be called

## Required Attributes

1 There are no required attributes for this API. Note that implementations may choose to internally  
2 execute integration for some security environments (e.g., directly contacting a *munge* server).

3 Implementations that support the operation but cannot directly process the client's request must  
4 pass any attributes that are provided by the client to the host environment for processing. In  
5 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
6 library to the host environment:

7 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
8       Effective user ID of the connecting process.  
9 **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
10      Effective group ID of the connecting process.

## Optional Attributes

11 The following attributes are optional for host environments that support this operation:

12 **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
13     Time in seconds before the specified operation should time out (zero indicating infinite) and  
14     return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
15     caused by multiple layers (client, server, and host) simultaneously timing the operation.

## 16 **Description**

17 Request a credential from the PMIx server library or the host environment. This version of the API  
18 is generally preferred in scenarios where the host environment may have to contact a remote  
19 credential service. Thus, provision is made for the system to return additional information (e.g., the  
20 identity of the issuing agent) outside of the credential itself and visible to the application.

### 21 **15.1.3 Credential Attributes**

22 The following attributes are defined to support credential operations:

23 **PMIX\_CRED\_TYPE** "pmix.sec.ctype" (**char\***)  
24     When passed in **PMIx\_Get\_credential**, a prioritized, comma-delimited list of desired  
25     credential types for use in environments where multiple authentication mechanisms may be  
26     available. When returned in a callback function, a string identifier of the credential type.  
27 **PMIX\_CRYPTO\_KEY** "pmix.sec.key" (**pmix\_byte\_object\_t**)  
28     Blob containing crypto key.

## 1 15.2 Validating Credentials

2 Given a credential, PMIx provides two methods by which a caller can request that the system  
3 validate it, returning any additional information (e.g., authorizations) conveyed within the  
4 credential.

### 5 15.2.1 PMIx\_Validate\_credential

#### 6 Summary

7 Request validation of a credential by the PMIx server library or the host environment.

#### 8 Format

PMIx v3.0

```
9 pmix_status_t
10 PMIx_Validate_credential(const pmix_byte_object_t *cred,
11                           const pmix_info_t info[], size_t ninfo,
12                           pmix_info_t **results, size_t *nresults);
```

#### 13 IN cred

14 Pointer to `pmix_byte_object_t` containing the credential (handle)

#### 15 IN info

16 Array of `pmix_info_t` structures (array of handles)

#### 17 IN ninfo

18 Number of elements in the *info* array (`size_t`)

#### 19 INOUT results

20 Address where a pointer to an array of `pmix_info_t` containing the results of the request  
21 can be returned (memory reference)

#### 22 INOUT nresults

23 Address where the number of elements in *results* can be returned (handle)

24 Returns one of the following:

- 25 • **PMIX\_SUCCESS**, indicating that the request was processed and returned *success* (i.e., the  
26 credential was both valid and any information it contained was successfully processed). Details  
27 of the result will be returned in the *results* array
- 28 • a PMIx error constant indicating either an error in the parsing of the credential or that the request  
29 was refused

## Required Attributes

1 There are no required attributes for this API. Note that implementations may choose to internally  
2 execute integration for some security environments (e.g., directly contacting a *munge* server).

3 Implementations that support the operation but cannot directly process the client's request must  
4 pass any attributes that are provided by the client to the host environment for processing. In  
5 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
6 library to the host environment:

7 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
8       Effective user ID of the connecting process.  
9 **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
10      Effective group ID of the connecting process.

## Optional Attributes

11 The following attributes are optional for host environments that support this operation:

12 **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
13     Time in seconds before the specified operation should time out (zero indicating infinite) and  
14     return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
15     caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Description

16 Request validation of a credential by the PMIx server library or the host environment.  
17

### 18 **15.2.2 PMIx\_Validate\_credential\_nb**

#### 19 **Summary**

20 Request validation of a credential by the PMIx server library or the host environment. Provision is  
21 made for the system to return additional information regarding possible authorization limitations  
22 beyond simple authentication.

1      **Format**

C

```
2      pmix_status_t  
3      PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,  
4                                const pmix_info_t info[], size_t ninfo,  
5                                pmix_validation_cbfunc_t cbfunc,  
6                                void *cbdata);
```

7      **IN cred**

8      Pointer to `pmix_byte_object_t` containing the credential (handle)

9      **IN info**

10     Array of `pmix_info_t` structures (array of handles)

11     **IN ninfo**

12     Number of elements in the *info* array (`size_t`)

13     **IN cbfunc**

14     Callback function to return result (`pmix_validation_cbfunc_t` function reference)

15     **IN cbdata**

16     Data to be passed to the callback function (memory reference)

17     Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided *cbfunc*
- a PMIx error constant indicating either an error in the input or that the request is unsupported - the *cbfunc* will *not* be called

22     Upon completion of processing the callback function will be executed. Note that the callback  
23     function must not be executed prior to return from the API.

▼----- Required Attributes -----▼

24     There are no required attributes for this API. Note that implementations may choose to internally  
25     execute integration for some security environments (e.g., directly contacting a *munge* server).

26     Implementations that support the operation but cannot directly process the client's request must  
27     pass any attributes that are provided by the client to the host environment for processing. In  
28     addition, the following attributes are required to be included in the *info* array passed from the PMIx  
29     library to the host environment:

30     **PMIX\_USERID "pmix.euid" (uint32\_t)**  
31         Effective user ID of the connecting process.

32     **PMIX\_GRP\_ID "pmix.egid" (uint32\_t)**  
33         Effective group ID of the connecting process.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (zero indicating infinite) and  
4 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
5 caused by multiple layers (client, server, and host) simultaneously timing the operation.

## 6 Description

7 Request validation of a credential by the PMIx server library or the host environment. This version  
8 of the API is generally preferred in scenarios where the host environment may have to contact a  
9 remote credential service. Provision is made for the system to return additional information (e.g.,  
10 possible authorization limitations) beyond simple authentication.

## CHAPTER 16

# Server-Specific Interfaces

---

The process that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes:

- collecting inventory to support scheduling algorithms,
- providing subsystems with an opportunity to precondition their resources for optimized application support,
- generating regular expressions,
- registering information to be passed to client processes, and
- requesting information on behalf of a remote process.

Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

## 16.1 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers.

### 16.1.1 PMIx\_server\_init

#### Summary

Initialize the PMIx server.

#### Format

PMIx v1.0

C

```
pmix_status_t  
PMIx_server_init(pmix_server_module_t *module,  
                  pmix_info_t info[], size_t ninfo);
```

```

1 INOUT module
2     pmix_server_module_t structure (handle)
3 IN info
4     Array of pmix_info_t structures (array of handles)
5 IN ninfo
6     Number of elements in the info array (size_t)
7 Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

**Required Attributes**

The following attributes are required to be supported by all PMIx libraries:

```

9 PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
10    Name of the namespace to use for this PMIx server.

11 PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
12    Rank of this PMIx server.

13 PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)
14    Top-level temporary directory for all client processes connected to this server, and where the
15    PMIx server will place its tool rendezvous point and contact information.

16 PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
17    Temporary directory for this system, and where a PMIx server that declares itself to be a
18    system-level server will place a tool rendezvous point and contact information.

19 PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)
20    The host RM wants to declare itself as willing to accept tool connection requests.

21 PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)
22    The host RM wants to declare itself as being the local system server for PMIx connection
23    requests.

24 PMIX_SERVER_SESSION_SUPPORT "pmix.srvr.sess" (bool)
25    The host RM wants to declare itself as being the local session server for PMIx connection
26    requests.

27 PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)
28    Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
29    (e.g., logging to email).

30 PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
31    Server is supporting system scheduler and desires access to appropriate services.

```

## Optional Attributes

1 The following attributes are optional for implementers of PMIx libraries:

2   **PMIX\_USOCK\_DISABLE** "pmix.usock.disable" (bool)

3     Disable legacy UNIX socket (usock) support. If the library supports Unix socket  
4     connections, this attribute may be supported for disabling it.

5   **PMIX\_SOCKET\_MODE** "pmix.sockmode" (uint32\_t)

6     POSIX *mode\_t* (9 bits valid). If the library supports socket connections, this attribute may  
7     be supported for setting the socket mode.

8   **PMIX\_SINGLE\_LISTENER** "pmix.sing.listnr" (bool)

9     Use only one rendezvous socket, letting priorities and/or environment parameters select the  
10    active transport.

11   **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (char\*)

12     If provided, directs that the TCP URI be reported and indicates the desired method of  
13     reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket  
14     connections, this attribute may be supported for reporting the URI.

15   **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)

16     Comma-delimited list of devices and/or CIDR notation to include when establishing the  
17     TCP connection. If the library supports TCP socket connections, this attribute may be  
18     supported for specifying the interfaces to be used.

19   **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)

20     Comma-delimited list of devices and/or CIDR notation to exclude when establishing the  
21     TCP connection. If the library supports TCP socket connections, this attribute may be  
22     supported for specifying the interfaces that are *not* to be used.

23   **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)

24     The IPv4 port to be used.. If the library supports IPV4 connections, this attribute may be  
25     supported for specifying the port to be used.

26   **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)

27     The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be  
28     supported for specifying the port to be used.

29   **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)

30     Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,  
31     this attribute may be supported for disabling it.

32   **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)

33     Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,  
34     this attribute may be supported for disabling it.

35   **PMIX\_SERVER\_REMOTE\_CONNECTIONS** "pmix.srvr.remote" (bool)

1 Allow connections from remote tools. Forces the PMIx server to not exclusively use  
2 loopback device. If the library supports connections from remote tools, this attribute may  
3 be supported for enabling or disabling it.

4 **PMIX\_EVENT\_BASE** "pmix.evbase" (**struct event\_base** \*)  
5 Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.

6 **PMIX\_TOPOLOGY2** "pmix.topo2" (**pmix\_topology\_t**)  
7 Provide a pointer to an implementation-specific description of the local node topology.

8 **PMIX\_SERVER\_SHARE\_TOPOLOGY** "pmix.srvr.share" (**bool**)  
9 The PMIx server is to share its copy of the local node topology (whether given to it or  
10 self-discovered) with any clients. The PMIx server will perform the necessary actions to  
11 scalably expose the description to the local clients. This includes creating any required  
12 shared memory backing stores and/ or XML representations, plus ensuring that all necessary  
13 key-value pairs for clients to access the description are included in the job-level information  
14 provided to each client. All required files are to be installed under the effective  
15 **PMIX\_SERVER\_TMPDIR** directory. The PMIx server library is responsible for cleaning up  
16 any artifacts (e.g., shared memory backing files or cached key-value pairs) at library finalize.

17 **PMIX\_SERVER\_ENABLE\_MONITORING** "pmix.srv.monitor" (**bool**)  
18 Enable PMIx internal monitoring by the PMIx server.

## 19 Description

20 Initialize the PMIx server support library, and provide a pointer to a **pmix\_server\_module\_t**  
21 structure containing the caller's callback functions. The array of **pmix\_info\_t** structs is used to  
22 pass additional info that may be required by the server when initializing. For example, it may  
23 include the **PMIX\_SERVER\_TOOL\_SUPPORT** attribute, thereby indicating that the daemon is  
24 willing to accept connection requests from tools.

## Advice to PMIx server hosts

25 Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module*  
26 structure. Doing so indicates that the host environment will not provide support for multi-node  
27 operations such as **PMIx\_Fence**, but does intend to support local clients access to information.

### 28 16.1.2 **PMIx\_server\_finalize**

#### 29 Summary

30 Finalize the PMIx server library.

---

<sup>1</sup><http://libevent.org/>

1      **Format**

2      `pmix_status_t`  
3      `PMIx_server_finalize(void);`

C

C

4      Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

5      **Description**

6      Finalize the PMIx server support library, terminating all connections to attached tools and any local  
7      clients. All memory usage is released.

8    **16.1.3 Server Initialization Attributes**

9      These attributes are used to direct the configuration and operation of the PMIx server library by  
10     passing them into `PMIx_server_init`.

11     `PMIX_TOPOLOGY2 "pmix.topo2" (pmix_topology_t)`

12       Provide a pointer to an implementation-specific description of the local node topology.

13     `PMIX_SERVER_SHARE_TOPOLOGY "pmix.srvr.share" (bool)`

14       The PMIx server is to share its copy of the local node topology (whether given to it or  
15       self-discovered) with any clients.

16     `PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)`

17       Disable legacy UNIX socket (usock) support.

18     `PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)`

19       POSIX mode\_t (9 bits valid).

20     `PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)`

21       Use only one rendezvous socket, letting priorities and/or environment parameters select the  
22       active transport.

23     `PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)`

24       The host RM wants to declare itself as willing to accept tool connection requests.

25     `PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)`

26       Allow connections from remote tools. Forces the PMIx server to not exclusively use  
27       loopback device.

28     `PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)`

29       The host RM wants to declare itself as being the local system server for PMIx connection  
30       requests.

31     `PMIX_SERVER_SESSION_SUPPORT "pmix.srvr.sess" (bool)`

32       The host RM wants to declare itself as being the local session server for PMIx connection  
33       requests.

34     `PMIX_SERVER_START_TIME "pmix.srvr.strtime" (char*)`

35       Time when the server started - i.e., when the server created it's rendezvous file (given in  
36       ctime string format).

37     `PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)`

```

1      Top-level temporary directory for all client processes connected to this server, and where the
2      PMIx server will place its tool rendezvous point and contact information.
3  PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
4      Temporary directory for this system, and where a PMIx server that declares itself to be a
5      system-level server will place a tool rendezvous point and contact information.
6  PMIX_SERVER_ENABLE_MONITORING "pmix.srv.monitor" (bool)
7      Enable PMIx internal monitoring by the PMIx server.
8  PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
9      Name of the namespace to use for this PMIx server.
10 PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
11     Rank of this PMIx server.
12 PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)
13     Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
14     (e.g., logging to email).
15 PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
16     Server is supporting system scheduler and desires access to appropriate services.
17 PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)
18     Pointer to libevent2 event_base to use in place of the internal progress thread.

```

## 16.2 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

### 16.2.1 PMIx\_generate\_regex

#### Summary

Generate a compressed representation of the input string.

#### Format

*PMIx v1.0*

```

23
24
25
26  pmix_status_t
27  PMIx_generate_regex(const char *input, char **output);

```

**IN** **input**  
String to process (string)

**OUT** **output**  
Compressed representation of *input* (array of bytes)

Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

---

<sup>2</sup><http://libevent.org/>

1           **Description**

2       Given a comma-separated list of *input* values, generate a reduced size representation of the input  
3       that can be passed down to the PMIx server library's **PMIx\_server\_register\_nspace** API  
4       for parsing. The order of the individual values in the *input* string is preserved across the operation.  
5       The caller is responsible for releasing the returned data.

6       The precise compressed representations will be implementation specific. However, all PMIx  
7       implementations are required to include a **NULL**-terminated string in the output representation that  
8       can be printed for diagnostic purposes.

---

**Advice to PMIx server hosts**

---

9       The returned representation may be an arbitrary array of bytes as opposed to a valid  
10      **NULL**-terminated string. However, the method used to generate the representation shall be  
11      identified with a colon-delimited string at the beginning of the output. For example, an output  
12      starting with "**pmix:\0**" might indicate that the representation is a PMIx-defined regular  
13      expression represented as a **NULL**-terminated string following the "**pmix:\0**" prefix. In contrast,  
14      an output starting with "**blob:\0**" might indicate a compressed binary array follows the prefix.

15     Communicating the resulting output should be done by first packing the returned expression using  
16     the **PMIx\_Data\_pack**, declaring the input to be of type **PMIX\_REGEX**, and then obtaining the  
17     resulting blob to be communicated using the **PMIX\_DATA\_BUFFER\_UNLOAD** macro. The  
18     reciprocal method can be used on the remote end prior to passing the regex into  
19     **PMIx\_server\_register\_nspace**. The pack/unpack routines will ensure proper handling of  
20     the data based on the regex prefix.

21    **16.2.2 PMIx\_generate\_ppn**

22    **Summary**

23    Generate a compressed representation of the input identifying the processes on each node.

24    **Format**

25    *PMIx v1.0*

26    

```
pmix_status_t
PMIx_generate_ppn(const char *input, char **ppn);
```

27    **IN    input**

28      String to process (string)

29    **OUT   ppn**

30      Compressed representation of *input* (array of bytes)

31      Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

1           **Description**

2         The input shall consist of a semicolon-separated list of ranges representing the ranks of processes  
3         on each node of the job - e.g., "1-4;2-5;8,10,11,12;6,7,9". Each field of the input must  
4         correspond to the node name provided at that position in the input to **PMIx\_generate\_regex**.  
5         Thus, in the example, ranks 1-4 would be located on the first node of the comma-separated list of  
6         names provided to **PMIx\_generate\_regex**, and ranks 2-5 would be on the second name in the  
7         list.

8           **Advice to PMIx server hosts**

9         The returned representation may be an arbitrary array of bytes as opposed to a valid  
10        **NULL**-terminated string. However, the method used to generate the representation shall be  
11        identified with a colon-delimited string at the beginning of the output. For example, an output  
12        starting with "**pmix**:" indicates that the representation is a PMIx-defined regular expression  
13        represented as a **NULL**-terminated string. In contrast, an output starting with  
"blob:\0size=1234:" is a compressed binary array.

14        Communicating the resulting output should be done by first packing the returned expression using  
15        the **PMIx\_Data\_pack**, declaring the input to be of type **PMIX\_REGEX**, and then obtaining the  
16        blob to be communicated using the **PMIX\_DATA\_BUFFER\_UNLOAD** macro. The pack/unpack  
17        routines will ensure proper handling of the data based on the regex prefix.

18        **16.2.3 PMIx\_server\_register\_nspace**

19           **Summary**

20         Setup the data about a particular namespace.

21           **Format**

PMIx v1.0

C

```
22        pmix_status_t
23        PMIx_server_register_nspace(const pmix_nspace_t nspace,
24                                    int nlocalprocs,
25                                    pmix_info_t info[], size_t ninfo,
26                                    pmix_op_cbfunc_t cbfunc,
27                                    void *cbdata);
```

```

1   IN  nspace
2     Character array of maximum size PMIX_MAX_NSLEN containing the namespace identifier
3     (string)
4   IN  nlocalprocs
5     number of local processes (integer)
6   IN  info
7     Array of info structures (array of handles)
8   IN  ninfo
9     Number of elements in the info array (integer)
10  IN  cbfunc
11    Callback function pmix_op_cbfunc_t to be executed upon completion of the operation.
12    A NULL function reference indicates that the function is to be executed as a blocking
13    operation (function reference)
14  IN  cbdata
15    Data to be passed to the callback function (memory reference)

16 Returns one of the following:
17
18  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
19    will be returned in the provided cbfunc. Note that the library must not invoke the callback
    function prior to returning from the API.
20
21  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
    returned success - the cbfunc will not be called
22
23  • a PMIx error constant indicating either an error in the input or that the request was immediately
    processed and failed - the cbfunc will not be called

```

### Required Attributes

The following attributes are required to be supported by all PMIx libraries:

```

25  PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)
26    Registration is for this namespace only, do not copy job data.

27  PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)
28    Provide an array of pmix_info_t containing session-realm information. The
29    PMIX_SESSION_ID attribute is required to be included in the array.

30  PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
31    Provide an array of pmix_info_t containing job-realm information. The
32    PMIX_SESSION_ID attribute of the session containing the job is required to be included in
33    the array whenever the PMIx server library may host multiple sessions (e.g., when executing
34    with a host RM daemon). As information is registered one job (aka namespace) at a time via
35    the PMIx_server_register_nspace API, there is no requirement that the array
36    contain either the PMIX_NAMESPACE or PMIX_JOBID attributes when used in that context

```

(though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is ambiguous.

**PMIX\_APP\_INFO\_ARRAY** "pmix.app.arr" (**pmix\_data\_array\_t**)

Provide an array of `pmix_info_t` containing application-realm information. The `PMIX_NSPACE` or `PMIX_JOBID` attributes of the *job* containing the application, plus its `PMIX_APPNUM` attribute, must to be included in the array when the array is *not* included as part of a call to `PMIX_server_register_nspace` - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

**PMIX\_PROC\_INFO\_ARRAY** "pmix\_pdata" (pmix\_data\_array\_t)

Provide an array of `pmix_info_t` containing process-realm information. The `PMIX_RANK` and `PMIX_NSPACE` attributes, or the `PMIX_PROCID` attribute, are required to be included in the array when the array is not included as part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the process is ambiguous. All three may be included if desired. When the array is included in some broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID` attribute must be included (the others are optional).

**PMIX NODE INFO ARRAY** "pmix.node.arr" (pmix\_data\_array\_t)

Provide an array of `pmix_info_t` containing node-realm information. At a minimum, either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the array, though both may be included.

Host environments are required to provide a wide range of session-, job-, application-, node-, and process-realm information, and may choose to provide a similarly wide range of optional information. The information is broadly separated into categories based on the *data realm* definitions explained in Section 6.1, and retrieved according to the rules detailed in Section 6.2.

Session-realm information may be passed as individual `pmix_info_t` entries, or as part of a `pmix_data_array_t` using the `PMIX_SESSION_INFO_ARRAY` attribute. The list of data referenced in this way shall include:

- **PMIX UNIV SIZE** "pmix.univ.size" (uint32\_t)

Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is equivalent to the **PMIX\_MAX\_PROCS** attributed combined with **PMIX\_SESSION\_INFO** array - it is included in the PMIx Standard for historical reasons.

- **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)

Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Must be provided if `PMIX_UNIV_SIZE` is not given. Requires use of the `PMIX_SESSION_INFO` attribute to avoid ambiguity when retrieving it.

- PMIX SESSION ID "pmix.session.id" (uint32\_t)

- 1                   Session identifier assigned by the scheduler.
- 2 plus the following optional information:
- 3     • **PMIX\_CLUSTER\_ID** "pmix.clid" (**char\***)  
4        A string name for the cluster this allocation is on. As this information is not related to the  
5        namespace, it is best passed using the **PMIx\_server\_register\_resources** API.
  - 6     • **PMIX\_ALLOCATED\_NODELIST** "pmix.alist" (**char\***)  
7        Comma-delimited list or regular expression of all nodes in the specified realm regardless  
8        of whether or not they currently host processes.
- 9 Job-realm information may be passed as individual **pmix\_info\_t** entries, or as part of a  
10 **pmix\_data\_array\_t** using the **PMIX\_JOB\_INFO\_ARRAY** attribute. The list of data  
11 referenced in this way shall include:
- 12    • **PMIX\_SERVER\_NSPACE** "pmix.srv.nspace" (**char\***)  
13      Name of the namespace to use for this PMIx server. Identifies the namespace of the PMIx  
14      server itself
  - 15    • **PMIX\_SERVER\_RANK** "pmix.srv.rank" (**pmix\_rank\_t**)  
16      Rank of this PMIx server. Identifies the rank of the PMIx server itself.
  - 17    • **PMIX\_NSPACE** "pmix.nspace" (**char\***)  
18      Namespace of the job - may be a numerical value expressed as a string, but is often an  
19      alphanumeric string carrying information solely of use to the system. Identifies the  
20      namespace of the job being registered.
  - 21    • **PMIX\_JOBID** "pmix.jobid" (**char\***)  
22      Job identifier assigned by the scheduler to the specified job - may be identical to the  
23      namespace, but is often a numerical value expressed as a string (e.g., "12345.3").
  - 24    • **PMIX\_JOB\_SIZE** "pmix.job.size" (**uint32\_t**)  
25      Total number of processes in the specified job across all contained applications. Note that  
26      this value can be different from **PMIX\_MAX\_PROCS**. For example, users may choose to  
27      subdivide an allocation (running several jobs in parallel within it), and dynamic  
28      programming models may support adding and removing processes from a running *job*  
29      on-the-fly. In the latter case, PMIx events may be used to notify processes within the job  
30      that the job size has changed.
  - 31    • **PMIX\_MAX\_PROCS** "pmix.max.size" (**uint32\_t**)  
32      Maximum number of processes that can be executed in the specified realm. Typically, this  
33      is a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
34      description. Retrieval of this attribute defaults to the job level unless an appropriate  
35      specification is given (e.g., **PMIX\_SESSION\_INFO**).
  - 36    • **PMIX\_NODE\_MAP** "pmix.nmap" (**char\***)  
37      Regular expression of nodes currently hosting processes in the specified realm - see  
38      16.2.3.2 for an explanation of its generation.

- 1     • **PMIX\_PROC\_MAP** "pmix.pmap" (**char\***)  
 2         Regular expression describing processes on each node in the specified realm - see [16.2.3.2](#)  
 3         for an explanation of its generation.
- 4     plus the following optional information:
- 5     • **PMIX\_NPROC\_OFFSET** "pmix.offset" (**pmix\_rank\_t**)  
 6         Starting global rank of the specified job.
- 7     • **PMIX\_JOB\_NUM\_APPS** "pmix.job.napps" (**uint32\_t**)  
 8         Number of applications in the specified job. This is a required attribute if more than one  
 9         application is included in the job.
- 10    • **PMIX\_MAPBY** "pmix.mapby" (**char\***)  
 11         Process mapping policy - when accessed using **PMIx\_Get**, use the  
 12         **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
 13         provided namespace. Supported values are launcher specific.
- 14    • **PMIX\_RANKBY** "pmix.rankby" (**char\***)  
 15         Process ranking policy - when accessed using **PMIx\_Get**, use the  
 16         **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for  
 17         the provided namespace. Supported values are launcher specific.
- 18    • **PMIX\_BINDTO** "pmix.bindto" (**char\***)  
 19         Process binding policy - when accessed using **PMIx\_Get**, use the  
 20         **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
 21         provided namespace. Supported values are launcher specific.
- 22    • **PMIX\_HOSTNAME\_KEEP\_FQDN** "pmix.fqdn" (**bool**)  
 23         FQDNs are being retained by the PMIx library.
- 24    • **PMIX\_ANL\_MAP** "pmix.anlmap" (**char\***)  
 25         Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
- 26     If more than one application is included in the namespace, then the host environment is also  
 27     required to supply data consisting of the following items for each application in the job, passed as a  
 28     **pmix\_data\_array\_t** using the **PMIX\_APP\_INFO\_ARRAY** attribute:
- 29     • **PMIX\_APPNUM** "pmix.appnum" (**uint32\_t**)  
 30         The application number within the job in which the specified process is a member. This  
 31         attribute must appear at the beginning of the array.
- 32     • **PMIX\_APP\_SIZE** "pmix.app.size" (**uint32\_t**)  
 33         Number of processes in the specified application, regardless of their execution state - i.e.,  
 34         this number may include processes that either failed to start or have already terminated.
- 35     • **PMIX\_MAX\_PROCS** "pmix.max.size" (**uint32\_t**)  
 36         Maximum number of processes that can be executed in the specified realm. Typically, this  
 37         is a constraint imposed by a scheduler or by user settings in a hostfile or other resource

- 1                   description. Requires use of the **PMIX\_APP\_INFO** attribute to avoid ambiguity when  
 2                   retrieving it.
- 3     • **PMIX\_APPLDR** "pmix.aldr" (**pmix\_rank\_t**)  
 4                   Lowest rank in the specified application.
- 5     • **PMIX\_WDIR** "pmix.wdir" (**char\***)  
 6                   Working directory for spawned processes. This attribute is required for all registrations,  
 7                   but may be provided as an individual **pmix\_info\_t** entry if only one application is  
 8                   included in the namespace.
- 9     • **PMIX\_APP\_ARGV** "pmix.app.argv" (**char\***)  
 10                  Consolidated argv passed to the spawn command for the given application (e.g., "./myapp  
 11                  arg1 arg2 arg3"). This attribute is required for all registrations, but may be provided as an  
 12                  individual **pmix\_info\_t** entry if only one application is included in the namespace.
- 13                  plus the following optional information:
- 14     • **PMIX\_PSET\_NAMES** "pmix.pset.nms" (**pmix\_data\_array\_t\***)  
 15                  Returns an array of **char\*** string names of the process sets in which the given process is  
 16                  a member.
- 17                  The data may also include attributes provided by the host environment that identify the  
 18                  programming model (as specified by the user) being executed within the application. The PMIx  
 19                  server library may utilize this information to customize the environment to fit that model (e.g.,  
 20                  adding environmental variables specified by the corresponding standard for that model):
- 21     • **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (**char\***)  
 22                  Programming model being initialized (e.g., "MPI" or "OpenMP").
- 23     • **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (**char\***)  
 24                  Programming model implementation ID (e.g., "OpenMPI" or "MPICH").
- 25     • **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (**char\***)  
 26                  Programming model version string (e.g., "2.1.1").
- 27                  Node-realm information may be passed as individual **pmix\_info\_t** entries if only one node will  
 28                  host processes from the job being registered, or as part of a **pmix\_data\_array\_t** using the  
 29                  **PMIX\_NODE\_INFO\_ARRAY** attribute when multiple nodes are involved in the job. The list of data  
 30                  referenced in this way shall include:
- 31     • **PMIX\_NODEID** "pmix.nodeid" (**uint32\_t**)  
 32                  Node identifier expressed as the node's index (beginning at zero) in an array of nodes  
 33                  within the active session. The value must be unique and directly correlate to the  
 34                  **PMIX\_HOSTNAME** of the node - i.e., users can interchangeably reference the same  
 35                  location using either the **PMIX\_HOSTNAME** or corresponding **PMIX\_NODEID**.
- 36     • **PMIX\_HOSTNAME** "pmix.hname" (**char\***)

1 Name of the host, as returned by the `gethostname` utility or its equivalent. As this  
2 information is not related to the namespace, it can be passed using the  
3 `PMIx_server_register_resources` API. However, either it or the  
4 `PMIX_NODEID` must be included in the array to properly identify the node.

- 5 • `PMIX_HOSTNAME_ALIASES` "pmix.alias" (`char*`)  
6 Comma-delimited list of names by which the target node is known. As this information is  
7 not related to the namespace, it is best passed using the  
8 `PMIx_server_register_resources` API.
- 9 • `PMIX_LOCAL_SIZE` "pmix.local.size" (`uint32_t`)  
10 Number of processes in the specified job or application realm on the caller's node.  
11 Defaults to job realm unless the `PMIX_APP_INFO` and the `PMIX_APPNUM` qualifiers are  
12 given.
- 13 • `PMIX_NODE_SIZE` "pmix.node.size" (`uint32_t`)  
14 Number of processes across all jobs that are executing upon the node.
- 15 • `PMIX_LOCALLDR` "pmix.lldr" (`pmix_rank_t`)  
16 Lowest rank within the specified job on the node (defaults to current node in absence of  
17 `PMIX_HOSTNAME` or `PMIX_NODEID` qualifier).
- 18 • `PMIX_LOCAL_PEERS` "pmix.lpeers" (`char*`)  
19 Comma-delimited list of ranks that are executing on the local node within the specified  
20 namespace – shortcut for `PMIxResolve_peers` for the local node.

21 plus the following information for the server's own node:

- 22 • `PMIX_TMPDIR` "pmix.tmpdir" (`char*`)  
23 Full path to the top-level temporary directory assigned to the session.
- 24 • `PMIX_NSDIR` "pmix.nsdir" (`char*`)  
25 Full path to the temporary directory assigned to the specified job, under `PMIX_TMPDIR`.
- 26 • `PMIX_LOCAL_PROCS` "pmix.lprocs" (`pmix_proc_t array`)  
27 Array of `pmix_proc_t` of all processes executing on the local node – shortcut for  
28 `PMIxResolve_peers` for the local node and a `NULL` namespace argument. The  
29 process identifier is ignored for this attribute.

30 The data may also include the following optional information for the server's own node:

- 31 • `PMIX_LOCAL_CPUSETS` "pmix.lcpus" (`pmix_data_array_t`)  
32 A `pmix_data_array_t` array of string representations of the PU binding bitmaps  
33 applied to each local `peer` on the caller's node upon launch. Each string shall begin with  
34 the name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap  
35 string itself. The array shall be in the same order as the processes returned by  
36 `PMIX_LOCAL_PEERS` for that namespace.
- 37 • `PMIX_AVAIL_PHYS_MEMORY` "pmix.pmem" (`uint64_t`)

1 Total available physical memory on a node. As this information is not related to the  
2 namespace, it can be passed using the **PMIx\_server\_register\_resources** API.

3 and the following optional information for other nodes:

- 4 • **PMIX\_MAX\_PROCS** "pmix.max.size" (**uint32\_t**)

5 Maximum number of processes that can be executed in the specified realm. Typically, this  
6 is a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
7 description. Requires use of the **PMIX\_NODE\_INFO** attribute to avoid ambiguity when  
8 retrieving it.

9 Process-realm information shall include the following data for each process in the job, passed as a  
10 **pmix\_data\_array\_t** using the **PMIX\_PROC\_INFO\_ARRAY** attribute:

- 11 • **PMIX\_RANK** "pmix.rank" (**pmix\_rank\_t**)

12 Process rank within the job, starting from zero.

- 13 • **PMIX\_APPNUM** "pmix.appnum" (**uint32\_t**)

14 The application number within the job in which the specified process is a member. This  
15 attribute may be omitted if only one application is present in the namespace.

- 16 • **PMIX\_APP\_RANK** "pmix.apprank" (**pmix\_rank\_t**)

17 Rank of the specified process within its application. This attribute may be omitted if only  
18 one application is present in the namespace.

- 19 • **PMIX\_GLOBAL\_RANK** "pmix.grank" (**pmix\_rank\_t**)

20 Rank of the specified process spanning across all jobs in this session, starting with zero.  
21 Note that no ordering of the jobs is implied when computing this value. As jobs can start  
22 and end at random times, this is defined as a continually growing number - i.e., it is not  
23 dynamically adjusted as individual jobs and processes are started or terminated.

- 24 • **PMIX\_LOCAL\_RANK** "pmix.lrank" (**uint16\_t**)

25 Rank of the specified process on its node - refers to the numerical location (starting from  
26 zero) of the process on its node when counting only those processes from the same job  
27 that share the node, ordered by their overall rank within that job.

- 28 • **PMIX\_NODE\_RANK** "pmix.nrank" (**uint16\_t**)

29 Rank of the specified process on its node spanning all jobs- refers to the numerical location  
30 (starting from zero) of the process on its node when counting all processes (regardless of  
31 job) that share the node, ordered by their overall rank within the job. The value represents  
32 a snapshot in time when the specified process was started on its node and is not  
33 dynamically adjusted as processes from other jobs are started or terminated on the node.

- 34 • **PMIX\_NODEID** "pmix.nodeid" (**uint32\_t**)

35 Node identifier expressed as the node's index (beginning at zero) in an array of nodes  
36 within the active session. The value must be unique and directly correlate to the  
37 **PMIX\_HOSTNAME** of the node - i.e., users can interchangeably reference the same  
38 location using either the **PMIX\_HOSTNAME** or corresponding **PMIX\_NODEID**.

- ```

1   • PMIX_REINCARNATION "pmix.reinc" (uint32_t)
2     Number of times this process has been re-instantiated - i.e, a value of zero indicates that
3     the process has never been restarted.
4   • PMIX_SPAWNED "pmix.spawned" (bool)
5     true if this process resulted from a call to PMIx_Spawn. Lack of inclusion (i.e., a return
6     status of PMIX_ERR_NOT_FOUND) corresponds to a value of false for this attribute.
7 plus the following information for processes that are local to the server:
8   • PMIX_LOCALITY_STRING "pmix.locstr" (char*)
9     String describing a process's bound location - referenced using the process's rank. The
10    string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon.
11    The remainder of the string represents the corresponding locality as expressed by the
12    underlying implementation. The entire string must be passed to
13    PMIx_Get_relative_locality for processing. Note that hosts are only required to
14    provide locality strings for local client processes - thus, a call to PMIx_Get for the
15    locality string of a process that returns PMIX_ERR_NOT_FOUND indicates that the
16    process is not executing on the same node.
17   • PMIX_PROCDIR "pmix.pdir" (char*)
18     Full path to the subdirectory under PMIX_NSDIR assigned to the specified process.
19 and the following optional information - note that this information can be derived from information
20 already provided by other attributes, but it may be included here for ease of retrieval by users:
21   • PMIX_HOSTNAME "pmix.hname" (char*)
22     Name of the host, as returned by the gethostname utility or its equivalent.
23


---


24 Attributes not directly provided by the host environment may be derived by the PMIx server library
25 from other required information and included in the data made available to the server library's
26 clients.


```

## 27      **Description**

28 Pass job-related information to the PMIx server library for distribution to local client processes.

### Advice to PMIx server hosts

29 Host environments are required to execute this operation prior to starting any local application
30 process within the given namespace.

31 The PMIx server must register all namespaces that will participate in collective operations with
32 local processes. This means that the server must register a namespace even if it will not host any
33 local processes from within that namespace if any local process of another namespace might at
34 some point perform an operation involving one or more processes from the new namespace. This is

1 necessary so that the collective operation can identify the participants and know when it is locally  
2 complete.

3 The caller must also provide the number of local processes that will be launched within this  
4 namespace. This is required for the PMIx server library to correctly handle collectives as a  
5 collective operation call can occur before all the local processes have been started.

6 A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

### Advice to users

7 The number of local processes for any given namespace is generally fixed at the time of application  
8 launch. Calls to **PMIx\_Spawn** result in processes launched in their own namespace, not that of  
9 their parent. However, it is possible for processes to *migrate* to another node via a call to  
10 **PMIx\_Job\_control\_nb**, thus resulting in a change to the number of local processes on both  
11 the initial node and the node to which the process moved. It is therefore critical that applications  
12 not migrate processes without first ensuring that PMIx-based collective operations are not in  
13 progress, and that no such operations be initiated until process migration has completed.

#### 16.2.3.1 Namespace registration attributes

15 The following attributes are defined specifically for use with the  
16 **PMIx\_server\_register\_nspace** API: **PMIX\_REGISTER\_NODATA**  
17 "**pmix.reg.nodata**" (**bool**)

18 Registration is for this namespace only, do not copy job data.

19 The following attributes are used to assemble information according to its data realm (*session*, *job*,  
20 *application*, *node*, or *process* as defined in Section 6.1) for registration where ambiguity may exist -  
21 see 16.2.3.2 for examples of their use.

22 **PMIX\_SESSION\_INFO\_ARRAY** "pmix.ssn.arr" (**pmix\_data\_array\_t**)  
23 Provide an array of **pmix\_info\_t** containing session-realm information. The  
24 **PMIX\_SESSION\_ID** attribute is required to be included in the array.  
25 **PMIX\_JOB\_INFO\_ARRAY** "pmix.job.arr" (**pmix\_data\_array\_t**)  
26 Provide an array of **pmix\_info\_t** containing job-realm information. The  
27 **PMIX\_SESSION\_ID** attribute of the *session* containing the *job* is required to be included in  
28 the array whenever the PMIx server library may host multiple sessions (e.g., when executing  
29 with a host RM daemon). As information is registered one job (aka namespace) at a time via  
30 the **PMIx\_server\_register\_nspace** API, there is no requirement that the array  
31 contain either the **PMIX\_NSPACE** or **PMIX\_JOBID** attributes when used in that context  
32 (though either or both of them may be included). At least one of the job identifiers must be  
33 provided in all other contexts where the job being referenced is ambiguous.  
34 **PMIX\_APP\_INFO\_ARRAY** "pmix.app.arr" (**pmix\_data\_array\_t**)

1      Provide an array of `pmix_info_t` containing application-realm information. The  
2      `PMIX_NSPACE` or `PMIX_JOBID` attributes of the *job* containing the application, plus its  
3      `PMIX_APPNUM` attribute, must be included in the array when the array is *not* included as  
4      part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the  
5      application is ambiguous. The job identification is otherwise optional.

6      **PMIX\_PROC\_INFO\_ARRAY** "pmix\_pdata" (`pmix_data_array_t`)

7      Provide an array of `pmix_info_t` containing process-realm information. The  
8      `PMIX_RANK` and `PMIX_NSPACE` attributes, or the `PMIX_PROCID` attribute, are required  
9      to be included in the array when the array is not included as part of a call to  
10     `PMIx_server_register_nspace` - i.e., when the job containing the process is  
11     ambiguous. All three may be included if desired. When the array is included in some  
12     broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID`  
13     attribute must be included (the others are optional).

14     **PMIX\_NODE\_INFO\_ARRAY** "pmix.node.arr" (`pmix_data_array_t`)

15     Provide an array of `pmix_info_t` containing node-realm information. At a minimum,  
16     either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the  
17     array, though both may be included.

18     Note that these assemblages can be used hierarchically:

- a `PMIX_JOB_INFO_ARRAY` might contain multiple `PMIX_APP_INFO_ARRAY` elements, each describing values for a specific application within the job.
- a `PMIX_JOB_INFO_ARRAY` could contain a `PMIX_NODE_INFO_ARRAY` for each node hosting processes from that job, each array describing job-level values for that node.
- a `PMIX_SESSION_INFO_ARRAY` might contain multiple `PMIX_JOB_INFO_ARRAY` elements, each describing a job executing within the session. Each job array could, in turn, contain both application and node arrays, thus providing a complete picture of the active operations within the allocation.

---

Advice to PMIx library implementers

---

27     PMIx implementations must be capable of properly parsing and storing any hierarchical depth of  
28     information arrays. The resulting stored values are must to be accessible via both `PMIx_Get` and  
29     `PMIx_Query_info_nb` APIs, assuming appropriate directives are provided by the caller.

---

### 16.2.3.2 Assembling the registration information

2 The following description is not intended to represent the actual layout of information in a given  
3 PMIx library. Instead, it is describes how information provided in the *info* parameter of the  
4 **PMIx\_server\_register\_nspace** shall be organized for proper processing by a PMIx server  
5 library. The ordering of the various information elements is arbitrary - they are presented in a  
6 top-down hierarchical form solely for clarity in reading.

#### Advice to PMIx server hosts

7 Creating the *info* array of data requires knowing in advance the number of elements required for the  
8 array. This can be difficult to compute and somewhat fragile in practice. One method for resolving  
9 the problem is to create a linked list of objects, each containing a single **pmix\_info\_t** structure.  
10 Allocation and manipulation of the list can then be accomplished using existing standard methods.  
11 Upon completion, the final *info* array can be allocated based on the number of elements on the list,  
12 and then the values in the list object **pmix\_info\_t** structures transferred to the corresponding  
13 array element utilizing the **PMIX\_INFO\_XFER** macro.

14 A common building block used in several areas is the construction of a regular expression  
15 identifying the nodes involved in that area - e.g., the nodes in a *session* or *job*. PMIx provides  
16 several tools to facilitate this operation, beginning by constructing an argv-like array of node  
17 names. This array is then passed to the **PMIx\_generate\_regex** function to create a regular  
18 expression parseable by the PMIx server library, as shown below:

C

```
19 char **nodes = NULL;
20 char *nodelist;
21 char *regex;
22 size_t n;
23 pmix_status_t rc;
24 pmix_info_t info;
25
26 /* loop over an array of nodes, adding each
27  * name to the array */
28 for (n=0; n < num_nodes; n++) {
29     /* filter the nodes to ignore those not included
30      * in the target range (session, job, etc.). In
31      * this example, all nodes are accepted */
32     PMIX_ARGV_APPEND(&nodes, node[n]->name);
33 }
34
35 /* join into a comma-delimited string */
36 nodelist = PMIX_ARGV_JOIN(nodes, ',');
```

```

1  /* release the array */
2  PMIX_ARGV_FREE(nodes);
3
4  /* generate regex */
5  rc = PMIx_generate_regex(nodelist, &regex);
6
7  /* release list */
8  free(nodelist);
9
10 /* pass the regex as the value to the PMIX_NODE_MAP key */
11 PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
12 /* release the regex */
13 free(regex);

```



14     Changing the filter criteria allows the construction of node maps for any level of information.

15     A similar method is used to construct the map of processes on each node from the namespace being  
16     registered. This may be done for each information level of interest (e.g., to identify the process map  
17     for the entire *job* or for each *application* in the job) by changing the search criteria. An example is  
18     shown below for the case of creating the process map for a *job*:



```

19 char **ndppn;
20 char rank[30];
21 char **ppnarray = NULL;
22 char *ppn;
23 char *localranks;
24 char *regex;
25 size_t n, m;
26 pmix_status_t rc;
27 pmix_info_t info;
28
29 /* loop over an array of nodes */
30 for (n=0; n < num_nodes; n++) {
31     /* for each node, construct an array of ranks on that node */
32     ndppn = NULL;
33     for (m=0; m < node[n]->num_procs; m++) {
34         /* ignore processes that are not part of the target job */
35         if (!PMIX_CHECK_NSPACE(targetjob, node[n]->proc[m].nspace)) {
36             continue;
37         }
38         sprintf(rank, 30, "%d", node[n]->proc[m].rank);
39         PMIX_ARGV_APPEND(&ndppn, rank);

```

```

1      }
2      /* convert the array into a comma-delimited string of ranks */
3      localranks = PMIX_ARGV_JOIN(ndppn, ',');
4      /* release the local array */
5      PMIX_ARGV_FREE(ndppn);
6      /* add this node's contribution to the overall array */
7      PMIX_ARGV_APPEND(&ppnarray, localranks);
8      /* release the local list */
9      free(localranks);
10     }
11
12     /* join into a semicolon-delimited string */
13     ppn = PMIX_ARGV_JOIN(ppnarray, ';');
14
15     /* release the array */
16     PMIX_ARGV_FREE(ppnarray);
17
18     /* generate ppn regex */
19     rc = PMIx_generate_ppn(ppn, &regex);
20
21     /* release list */
22     free(ppn);
23
24     /* pass the regex as the value to the PMIX_PROC_MAP key */
25     PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
26     /* release the regex */
27     free(regex);

```

C

Note that the **PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes are linked in that the order of entries in the process map must match the ordering of nodes in the node map - i.e., there is no provision in the PMIx process map regular expression generator/parser pair supporting an out-of-order node or a node that has no corresponding process map entry (e.g., a node with no processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

- Session-level information includes all session-specific values. In many cases, only two values (**PMIX\_SESSION\_ID** and **PMIX\_UNIV\_SIZE**) are included in the registration array. Since both of these values are session-specific, they can be specified independently - i.e., in their own **pmix\_info\_t** elements of the *info* array. Alternatively, they can be provided as a **pmix\_data\_array\_t** array of **pmix\_info\_t** using the **PMIX\_SESSION\_INFO\_ARRAY** attribute and identified by including the **PMIX\_SESSION\_ID** attribute in the array - this is required in cases where non-specific attributes (e.g., **PMIX\_NUM\_NODES** or **PMIX\_NODE\_MAP**) are passed to describe aspects of the session. Note that the node map can include nodes not used by the job being registered as no corresponding process map is specified.

1      The *info* array at this point might look like (where the labels identify the corresponding attribute  
2      - e.g., “Session ID” corresponds to the **PMIX\_SESSION\_ID** attribute):

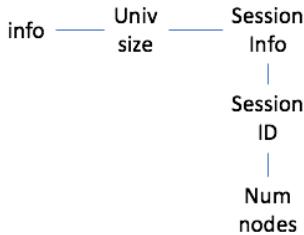


Figure 16.1.: Session-level information elements

- 3      • Job-level information includes all job-specific values such as **PMIX\_JOB\_SIZE**,  
4      **PMIX\_JOB\_NUM\_APPS**, and **PMIX\_JOBID**. Since each invocation of  
5      **PMIx\_server\_register\_nspace** describes a single *job*, job-specific values can be  
6      specified independently - i.e., in their own **pmix\_info\_t** elements of the *info* array.  
7      Alternatively, they can be provided as a **pmix\_data\_array\_t** array of **pmix\_info\_t**  
8      identified by the **PMIX\_JOB\_INFO\_ARRAY** attribute - this is required in cases where  
9      non-specific attributes (e.g., **PMIX\_NODE\_MAP**) are passed to describe aspects of the job. Note  
10     that since the invocation only involves a single namespace, there is no need to include the  
11     **PMIX\_NSPACE** attribute in the array.

12     Upon conclusion of this step, the *info* array might look like:

13     Note that in this example, **PMIX\_NUM\_NODES** is not required as that information is contained in  
14     the **PMIX\_NODE\_MAP** attribute. Similarly, **PMIX\_JOB\_SIZE** is not technically required as that  
15     information is contained in the **PMIX\_PROC\_MAP** when combined with the corresponding node  
16     map - however, there is no issue with including the job size as a separate entry.

17     The example also illustrates the hierarchical use of the **PMIX\_NODE\_INFO\_ARRAY** attribute.  
18     In this case, we have chosen to pass several job-related values for each node - since those values  
19     are non-unique across the job, they must be passed in a node-info container. Note that the choice  
20     of what information to pass into the PMIx server library versus what information to derive from  
21     other values at time of request is left to the host environment. PMIx implementors in turn may, if  
22     they choose, pre-parse registration data to create expanded views (thus enabling faster response  
23     to requests at the expense of memory footprint) or to compress views into tighter representations  
24     (thus trading minimized footprint for longer response times).

- 25     • Application-level information includes all application-specific values such as **PMIX\_APP\_SIZE**  
26     and **PMIX\_APPLDR**. If the *job* contains only a single *application*, then the application-specific  
27     values can be specified independently - i.e., in their own **pmix\_info\_t** elements of the *info*  
28     array - or as a **pmix\_data\_array\_t** array of **pmix\_info\_t** using the  
29     **PMIX\_APP\_INFO\_ARRAY** attribute and identified by including the **PMIX\_APPNUM** attribute in

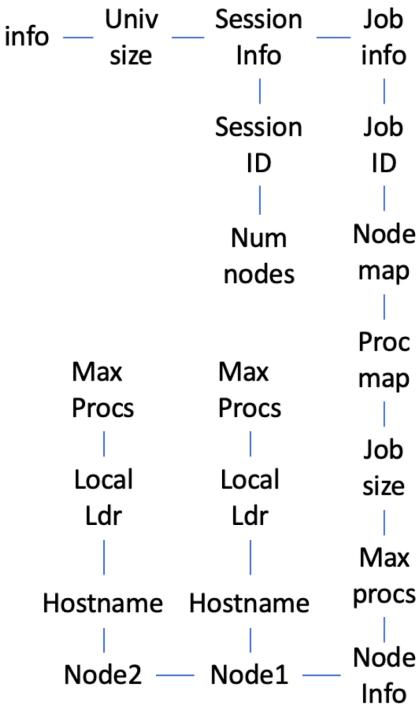


Figure 16.2.: Job-level information elements

the array. Use of the array format is must in cases where non-specific attributes (e.g., [PMIX\\_NODE\\_MAP](#)) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the [PMIX\\_APP\\_INFO\\_ARRAY](#) format, each identified by its [PMIX\\_APPNUM](#) value.

Upon conclusion of this step, the *info* array might look like that shown in [16.3](#), assuming there are two applications in the job being registered:

- Process-level information includes an entry for each process in the job being registered, each entry marked with the [PMIX\\_PROC\\_INFO\\_ARRAY](#) attribute. The *rank* of the process must be the first entry in the array - this provides efficiency when storing the data. Upon conclusion of this step, the *info* array might look like the diagram in [16.4](#):
- For purposes of this example, node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases, the values included in this level are unique to it and can be specified independently - i.e., in their own [pmix\\_info\\_t](#) elements of the *info* array. Alternatively, they can be provided as a

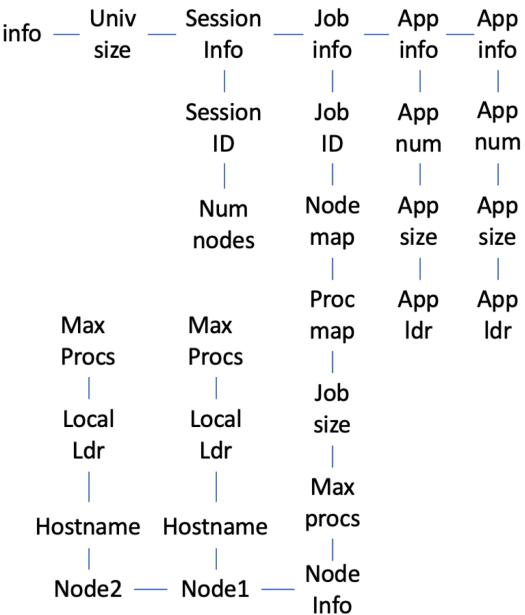


Figure 16.3.: Application-level information elements

1      **pmix\_data\_array\_t** array of **pmix\_info\_t** using the **PMIX\_NODE\_INFO\_ARRAY**  
 2      attribute - this is required in cases where non-specific attributes are passed to describe aspects of  
 3      the node, or where values for multiple nodes are being provided.

4      The node-level information requires two elements that must be constructed in a manner similar to  
 5      that used for the node map. The **PMIX\_LOCAL\_PEERS** value is computed based on the  
 6      processes on the local node, filtered to select those from the job being registered, as shown below  
 7      using the tools provided by PMIx:

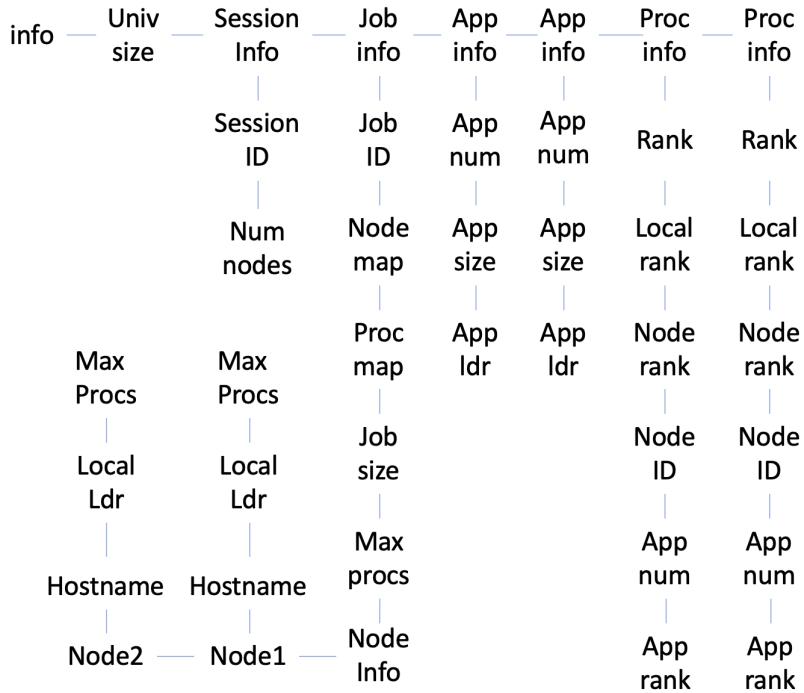


Figure 16.4.: Process-level information elements

C

```

1  char **ndppn = NULL;
2  char rank[30];
3  char *localranks;
4  size_t m;
5  pmix_info_t info;
6
7  for (m=0; m < mynode->num_procs; m++) {
8      /* ignore processes that are not part of the target job */
9      if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace)) {
10          continue;
11      }
12      sprintf(rank, 30, "%d", mynode->proc[m].rank);
13      PMIX_ARGV_APPEND(&ndppn, rank);
14  }

```

```

1  /* convert the array into a comma-delimited string of ranks */
2  localranks = PMIX_ARGV_JOIN(ndppn, ',');
3  /* release the local array */
4  PMIX_ARGV_FREE(ndppn);
5
6  /* pass the string as the value to the PMIX_LOCAL_PEERS key */
7  PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
8
9  /* release the list */
10 free(localranks);

```

C

11 The **PMIX\_LOCAL\_CPUSETS** value is constructed in a similar manner. In the provided  
12 example, it is assumed that an Hardware Locality (HWLOC) cpuset representation (a  
13 comma-delimited string of processor IDs) of the processors assigned to each process has  
14 previously been generated and stored on the process description. Thus, the value can be  
15 constructed as shown below:

C

```

16 char **ndcpus = NULL;
17 char *localcpus;
18 size_t m;
19 pmix_info_t info;
20
21 for (m=0; m < mynode->num_procs; m++) {
22     /* ignore processes that are not part of the target job */
23     if (!PMIX_CHECK_NSPACE(targetjob, mynode->proc[m].nspace)) {
24         continue;
25     }
26     PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
27 }
28 /* convert the array into a colon-delimited string */
29 localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
30 /* release the local array */
31 PMIX_ARGV_FREE(ndcpus);
32
33 /* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
34 PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);
35
36 /* release the list */
37 free(localcpus);

```

C

38 Note that for efficiency, these two values can be computed at the same time.

1

The final *info* array might therefore look like the diagram in 16.5:

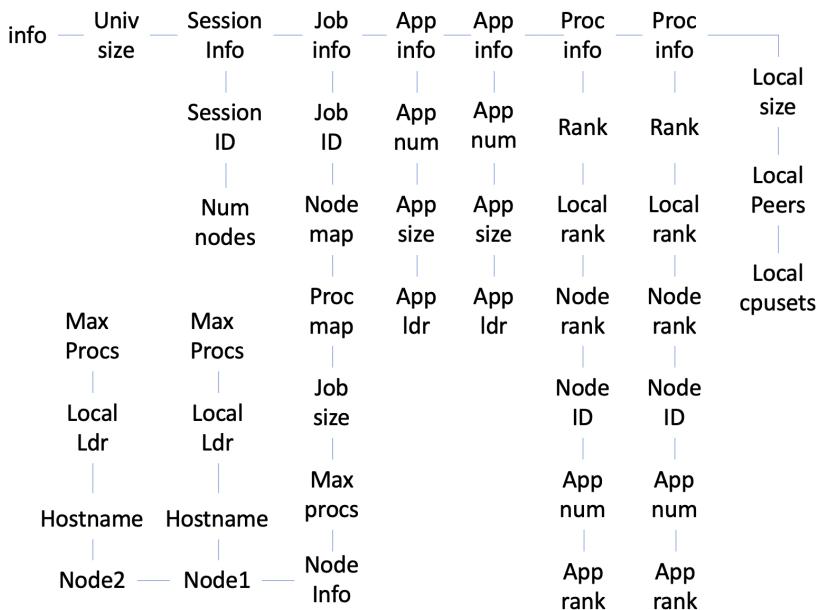


Figure 16.5.: Final information array

## 2 16.2.4 PMIx\_server\_deregister\_nspace

### 3 Summary

4 Deregister a namespace.

### 5 Format

6 *PMIx v1.0*

C

```
7 void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
                                     pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

8 IN **nspac**

9 Namespace (string)

10 IN **cbfunc**

11 Callback function **pmix\_op\_cbfunc\_t**. A **NULL** function reference indicates that the  
12 function is to be executed as a blocking operation. (function reference)

13 IN **cbdata**

14 Data to be passed to the callback function (memory reference)

## Description

Deregister the specified *nspace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library must not invoke the callback function prior to returning from the API, and that a **NULL** *cfunc* reference indicates that the function is to be executed as a blocking operation.

### 16.2.5 PMIx server register resources

## Summary

Register non-namespace related information with the local PMIx server library.

## Format

*PMIx v4.0*

```
pmix_status_t  
PMIx_server_register_resources(pmix_info_t info[], size_t ninfo,  
                               pmix_op_cbfunc_t cbfunc,  
                               void *cbdata);
```

|           |               |                                                                                                                                                                         |
|-----------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IN</b> | <b>info</b>   | Array of info structures (array of handles)                                                                                                                             |
| <b>IN</b> | <b>ninfo</b>  | Number of elements in the <i>info</i> array (integer)                                                                                                                   |
| <b>IN</b> | <b>cbfunc</b> | Callback function <b>pmix_op_cbfunc_t</b> . A <b>NULL</b> function reference indicates that the function is to be executed as a blocking operation (function reference) |
| <b>IN</b> | <b>cbdata</b> | Data to be passed to the callback function (memory reference)                                                                                                           |

## Description

Pass information about resources not associated with a given namespace to the PMIx server library for distribution to local client processes. This includes information on fabric devices, Graphics Processing Units (GPUs), and other resources. All information provided through this API shall be made available to each job as part of its job-level information. Duplicate information provided with the `PMIx_server_register_nspace` API shall override any information provided by this function for that namespace, but only for that specific namespace.

## Advice to PMIx server hosts

Note that information passed in this manner could also have been included in a call to `PMIx_server_register_nspace` - e.g., as part of a `PMIX_NODE_INFO_ARRAY` array. This API is provided as a logical alternative for code clarity, especially where multiple jobs may be supported by a single PMIx server library instance, to avoid multiple registration of static resource information.

A `NULL` `cbfunc` reference indicates that the function is to be executed as a blocking operation.

### 16.2.6 `PMIx_server_deregister_resources`

#### Summary

Remove specified non-namespace related information from the local PMIx server library.

#### Format

*PMIx v4.0*

```
11     pmix_status_t
12     PMIx_server_deregister_resources(pmix_info_t info[], size_t ninfo,
13   pmix_op_cbfunc_t cbfunc,
14   void *cbdata);
```

C

C

#### IN `info`

Array of info structures (array of handles)

#### IN `ninfo`

Number of elements in the `info` array (integer)

#### IN `cbfunc`

Callback function `pmix_op_cbfunc_t`. A `NULL` function reference indicates that the function is to be executed as a blocking operation (function reference)

#### IN `cbdata`

Data to be passed to the callback function (memory reference)

1           **Description**

2         Remove information about resources not associated with a given namespace from the PMIx server  
3         library. Only the *key* fields of the provided *info* array shall be used for the operation - the associated  
4         values shall be ignored except where they serve as qualifiers to the request. For example, to remove  
5         a specific fabric device from a given node, the *info* array might include a  
6         **PMIX\_NODE\_INFO\_ARRAY** containing the **PMIX\_NODEID** or **PMIX\_HOSTNAME** identifying  
7         the node hosting the device, and the **PMIX\_FABRIC\_DEVICE\_NAME** specifying the device to be  
8         removed. Alternatively, the device could be removed using only the **PMIX\_FABRIC\_DEVICE\_ID**  
9         as this is unique across the overall system.

---

Advice to PMIx server hosts

---

10      As information not related to namespaces is considered *static*, there is no requirement that the host  
11     environment deregister resources prior to finalizing the PMIx server library. The server library  
12     shall properly cleanup as part of its normal finalize operations. Deregistration of resources is only  
13     required, therefore, when the host environment determines that client processes should no longer  
14     have access to that information.

15      A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

16   **16.2.7 PMIx\_server\_register\_client**

17   **Summary**

18   Register a client process with the PMIx server library.

19   **Format**

PMIx v1.0

C

```
20       pmix_status_t
21       PMIx_server_register_client(const pmix_proc_t *proc,
22                            uid_t uid, gid_t gid,
23                            void *server_object,
24                            pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

25   **IN proc**
26       **pmix\_proc\_t** structure (handle)
27   **IN uid**
28       user id (integer)
29   **IN gid**
30       group id (integer)
31   **IN server\_object**
32       (memory reference)

1     **IN    cbfunc**  
2        Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)  
3     **IN    cldata**  
4        Data to be passed to the callback function (memory reference)  
5        Returns one of the following:  
6        

- [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
7           will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
8           function prior to returning from the API.
- [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
9           returned *success* - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately  
10          processed and failed - the *cbfunc* will not be called

### 13     **Description**

14     Register a client process with the PMIx server library.

15     The host server can also, if it desires, provide an object it wishes to be returned when a server  
16        function is called that relates to a specific process. For example, the host server may have an object  
17        that tracks the specific client. Passing the object to the library allows the library to provide that  
18        object to the host server during subsequent calls related to that client, such as a  
19        [pmix\\_server\\_client\\_connected\\_fn\\_t](#) function. This allows the host server to access  
20        the object without performing a lookup based on the client's namespace and rank.

### Advice to PMIx server hosts

21     Host environments are required to execute this operation prior to starting the client process. The  
22        expected user ID and group ID of the child process allows the server library to properly authenticate  
23        clients as they connect by requiring the two values to match. Accordingly, the detected user and  
24        group ID's of the connecting process are not included in the  
25        [pmix\\_server\\_client\\_connected\\_fn\\_t](#) server module function.

### Advice to PMIx library implementers

26     For security purposes, the PMIx server library should check the user and group ID's of a  
27        connecting process against those provided for the declared client process identifier via the  
28        [PMIx\\_server\\_register\\_client](#) prior to completing the connection.

## 29     16.2.8    **PMIx\_server\_deregister\_client**

### 30     **Summary**

31     Deregister a client and purge all data relating to it.

```
1 Format
2 void
3 PMIx_server_deregister_client(const pmix_proc_t *proc,
4                                 pmix_op_cbfunc_t cbfunc, void *cbdata);
5 IN proc
6     pmix_proc_t structure (handle)
7 IN cbfunc
8     Callback function pmix_op_cbfunc_t (function reference)
9 IN cbdata
10    Data to be passed to the callback function (memory reference)
```

## Description

The `PMIx_server_deregister_nspace` API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library must not invoke the callback function prior to returning from the API.

## 16.2.9 PMIx server setup fork

## Summary

Setup the environment of a child process to be forked by the host.

## Format

```
PMIx v1.0 C
21     pmix_status_t
22     PMIx_server_setup_fork(const pmix_proc_t *proc,
23                               char ***env);
24     IN  proc
25         pmix_proc_t structure (handle)
26     IN  env
27         Environment array (array of strings)
28     Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

## Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via [PMIx\\_Spawn](#)) plus other values (e.g., variables required to properly initialize the client's fabric library).

## Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process.

### 16.2.10 PMIx server dmodex request

## Summary

Define a function by which the host server can request modex data from the local PMIx server.

## Format

PMIx v1.0

0

```
pmix_status_t  
PMIx_server_dmodex_request(const pmix_proc_t *proc,  
                           pmix_dmodex_response_fn_t cbfunc,  
                           void *cbdata);
```

- IN **proc** **pmix\_proc\_t** structure (handle)
- IN **cbfunc** Callback function **pmix\_dmodex**
- IN **cbdata** Data to be passed to the callback function

Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
  - a PMIx error constant indicating an error in the input - the *cbfunc* will not be called

1           **Description**

2       Define a function by which the host server can request modex data from the local PMIx server.  
3       Traditional wireup procedures revolve around the per-process posting of data (e.g., location and  
4       endpoint information) via the **PMIx\_Put** and **PMIx\_Commit** functions followed by a  
5       **PMIx\_Fence** barrier that globally exchanges the posted information. However, the barrier  
6       operation represents a significant time impact at large scale.

7       PMIx supports an alternative wireup method known as *Direct Modex* that replaces the  
8       barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In  
9       place of the barrier operation, data posted by each process is cached on the local PMIx server.  
10      When a process requests the information posted by a particular peer, it first checks the local cache  
11      to see if the data is already available. If not, then the request is passed to the local PMIx server,  
12      which subsequently requests that its RM host request the data from the RM daemon on the node  
13      where the specified peer process is located. Upon receiving the request, the RM daemon passes the  
14      request into its PMIx server library using the **PMIx\_server\_dmodex\_request** function,  
15      receiving the response in the provided *cbfunc* once the indicated process has posted its information.  
16      The RM daemon then returns the data to the requesting daemon, who subsequently passes the data  
17      to its PMIx server library for transfer to the requesting client.

---

Advice to users

---

18      While direct modex allows for faster launch times by eliminating the barrier operation, per-peer  
19      retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by  
20      returning posted information from all processes on a node upon first request - but in general direct  
21      modex remains best suited for sparsely connected applications.

22     **16.2.10.1 Server Direct Modex Response Callback Function**

23      The **PMIx\_server\_dmodex\_request** callback function.

24     **Summary**

25      Provide a function by which the local PMIx server library can return connection and other data  
26      posted by local application processes to the host resource manager.

## Format

Q

IN status

Returned status of the request (**pmix status t**)

IN data

Pointer to a data "blob" containing the requested information (handle)

IN SZ

Number of bytes in the *data* blob (integer)

IN cbdata

Data passed into the initial call to `PMIx_server_dmodex_request` (memory reference)

## Description

Define a function to be called by the PMIx server library for return of information posted by a local application process (via [PMIx\\_Put](#) with subsequent [PMIx\\_Commit](#)) in response to a request from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd upon return from the function.

## 19 16.2.11 PMIx\_server\_setup\_application

## Summary

Provide a function by which a launcher can request application-specific setup data prior to launch of a *job*.

## Format

Q

24

## **pmix\_status\_t**

```
PMIx_server_setup_application(const pmix_nspace_t nspace,
                           pmix_info_t info[], size_t ninfo,
                           pmix_setup_application_cbfunc_t cbfunc,
                           void *cbdata);
```

IN nspace

namespace (string)

IN info

Array of info structures (array of handles)

```

1   IN  ninfo
2     Number of elements in the info array (integer)
3   IN  cbfunc
4     Callback function pmix_setup_application_cbfunc_t (function reference)
5   IN  cbdatal
6     Data to be passed to the cbfunc callback function (memory reference)
7
8   Returns one of the following:
9
10  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
11    will be returned in the provided cbfunc. Note that the library must not invoke the callback
12    function prior to returning from the API.
13
14  • a PMIx error constant indicating either an error in the input - the cbfunc will not be called

```

### Required Attributes

PMIx libraries that support this operation are required to support the following:

```

13  PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)
14    Harvest and include relevant environmental variables.

15  PMIX_SETUP_APP_NONENVARS ""pmix.setup.nenv" (bool)
16    Include all relevant data other than environmental variables.

17  PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)
18    Include all relevant data.

19  PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
20    Array of pmix_info_t describing requested fabric resources. This must include at least:
21    PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
22    PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.

23  PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
24    The key to be used when accessing this requested fabric allocation. The fabric allocation
25    will be returned/stored as a pmix_data_array_t of pmix_info_t whose first
26    element is composed of this key and the allocated resource description. The type of the
27    included value depends upon the fabric support. For example, a TCP allocation might
28    consist of a comma-delimited string of socket ranges such as "32000-32100,
29    33005,38123-38146". Additional array entries will consist of any provided resource
30    request directives, along with their assigned values. Examples include:
31    PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
32    PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
33    from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
34    the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
35    requested fabric allocation. NOTE: the array contents may differ from those requested,
36    especially if PMIX_INFO_REQD was not set in the request.

37  PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)

```

```

1 Request that the allocation include a fabric security key for the spawned job.
2 PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
3 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
4 PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
5 ID string for the fabric plane to be used for the requested allocation.
6 PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
7 Number of endpoints to allocate per process in the job.
8 PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
9 Number of endpoints to allocate per node for the job.
10 PMIX_PROC_MAP "pmix.pmap" (char*)
11 Regular expression describing processes on each node in the specified realm - see 16.2.3.2
12 for an explanation of its generation.
13 PMIX_NODE_MAP "pmix.nmap" (char*)
14 Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2
15 for an explanation of its generation.

```

### Optional Attributes

PMIx libraries that support this operation may support the following:

```

17 PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
18 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
19 request.
20 PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
21 Fabric quality of service level for the job being requested in an allocation request.

```

The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the application. The PMIx server library may utilize this information to harvest/forward model-specific environmental variables, record the programming model associated with the application, etc.

- **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (**char\***)  
Programming model being initialized (e.g., "MPI" or "OpenMP").
- **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (**char\***)  
Programming model implementation ID (e.g., "OpenMPI" or "MPICH").
- **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (**char\***)  
Programming model version string (e.g., "2.1.1").

1           **Description**

2         Provide a function by which the RM can request application-specific setup data (e.g., environmental  
3         variables, fabric configuration and security credentials) from supporting PMIx server library  
4         subsystems prior to initiating launch of a job.

5         This is defined as a non-blocking operation in case contributing subsystems need to perform some  
6         potentially time consuming action (e.g., query a remote service) before responding. The returned  
7         data must be distributed by the host environment and subsequently delivered to the local PMIx  
8         server on each node where application processes will execute, prior to initiating execution of those  
9         processes.

10            **Advice to PMIx server hosts** 

11         Host environments are required to execute this operation prior to launching a job. In addition to  
12         supported directives, the *info* array must include a description of the *job* using the **PMIX\_JOB** attribute.  
**PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes.

13         Note that the function can be called on a per-application basis if the **PMIX\_PROC\_MAP** and  
14         **PMIX\_NODE\_MAP** are provided only for the corresponding application (as opposed to the entire  
15         job) each time.

16            **Advice to PMIx library implementers** 

17         Support for harvesting of environmental variables and providing of local configuration information  
by the PMIx implementation is optional.

18         **16.2.11.1 Server Setup Application Callback Function**

19         The **PMIx\_server\_setup\_application** callback function.

20           **Summary**

21         Provide a function by which the resource manager can receive application-specific environmental  
22         variables and other setup data prior to launch of an application.

1                   **Format**

2        *PMIx v2.0*                   C

```

2        typedef void (*pmix_setup_application_cbfunc_t)(
3    pmix_status_t status,
4    pmix_info_t info[], size_t ninfo,
5    void *provided_cbdata,
6    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

7                   **IN status**  
8                    returned status of the request ([pmix\\_status\\_t](#))  
9                   **IN info**  
10                  Array of info structures (array of handles)  
11                   **IN ninfo**  
12                  Number of elements in the *info* array (integer)  
13                   **IN provided\_cbdata**  
14                  Data originally passed to call to [PMIx\\_server\\_setup\\_application](#) (memory  
15                  reference)  
16                   **IN cbfunc**  
17                  [pmix\\_op\\_cbfunc\\_t](#) function to be called when processing completed (function reference)  
18                   **IN cbdata**  
19                  Data to be passed to the *cbfunc* callback function (memory reference)

20                   **Description**  
21                  Define a function to be called by the PMIx server library for return of application-specific setup  
22                  data in response to a request from the host RM. The returned *info* array is owned by the PMIx  
23                  server library and will be free'd when the provided *cbfunc* is called.

24     **16.2.11.2 Server Setup Application Attributes**

25     *PMIx v3.0*   Attributes specifically defined for controlling contents of application setup data.

```

26        PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)  

27                  Harvest and include relevant environmental variables.  

28        PMIX_SETUP_APP_NONENVARS "" "pmix.setup.nenv" (bool)  

29                  Include all relevant data other than environmental variables.  

30        PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)  

31                  Include all relevant data.
```

32     **16.2.12 PMIx\_Register\_attributes**

33                   **Summary**  
34                  Register host environment attribute support for a function.

1           **Format**

2        *pmix\_status\_t*  
3        **PMIx\_Register\_attributes**(*char \*function,*  
4                                    *pmix\_regattr\_t attrs[],*  
5                                    *size\_t nattrs);*

C

C

6        **IN    function**

7        String name of function (string)

8        **IN    attrs**

9        Array of **pmix\_regattr\_t** describing the supported attributes (handle)

10      **IN    nattrs**

11      Number of elements in *attrs* (**size\_t**)

12     Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

13      **Description**

14     The **PMIx\_Register\_attributes** function is used by the host environment to register with  
15     its PMIx server library the attributes it supports for each **pmix\_server\_module\_t** function.

16     The *function* is the string name of the server module function (e.g., "register\_events",

17     "validate\_credential", or "allocate") whose attributes are being registered. See the

18     **pmix\_regattr\_t** entry for a description of the *attrs* array elements.

19     Note that the host environment can also query the library (using the **PMIx\_Query\_info\_nb**  
20     API) for its attribute support both at the server, client, and tool levels once the host has executed  
21     **PMIx\_server\_init** since the server will internally register those values.

22      **Advice to PMIx server hosts**

23     Host environments are strongly encouraged to register all supported attributes immediately after  
initializing the library to ensure that user requests are correctly serviced.

## Advice to PMIx library implementers

1 PMIx implementations are *required* to register all internally supported attributes for each API  
2 during initialization of the library (i.e., when the process calls their respective PMIx init function).  
3 Specifically, the implementation *must not* register supported attributes upon first call to a given API  
4 as this would prevent users from discovering supported attributes prior to first use of an API.

5 It is the implementation's responsibility to associate registered attributes for a given  
6 `pmix_server_module_t` function with their corresponding user-facing API. Supported  
7 attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by  
8 the level (see Section 5.4.6) at which the attribute is supported.

9 Note that attributes can/will be registered on an API for each level. It is *required* that the  
10 implementation support user queries for supported attributes on a per-level basis. Duplicate  
11 registrations at the *same* level for a function *shall* return an error - however, duplicate registrations  
12 at *different* levels *shall* be independently tracked.

### 16.2.12.1 Attribute registration constants

14 Constants supporting attribute registration.

15 `PMIX_ERR_REPEAT_ATTR_REGISTRATION` The attributes for an identical function have  
16 already been registered at the specified level (host, server, or client).

### 16.2.12.2 Attribute registration structure

18 The `pmix_regattr_t` structure is used to register attribute support for a PMIx function.

PMIx v4.0

C

```
19     typedef struct pmix_regattr {
20         char *name;
21         pmix_key_t *string;
22         pmix_data_type_t type;
23         pmix_info_t *info;
24         size_t ninfo;
25         char **description;
26     } pmix_regattr_t;
```

C

27 Note that in this structure:

- 28 • the *name* is the actual name of the attribute - e.g., "PMIX\_MAX\_PROCS"
- 29 • the *string* is the literal string value of the attribute - e.g., "pmix.max.size" for the  
30 `PMIX_MAX_PROCS` attribute
- 31 • *type* must be a PMIx data type identifying the type of data associated with this attribute.

- the *info* array contains machine-useable information regarding the range of accepted values. This may include entries for **PMIX\_MIN\_VALUE**, **PMIX\_MAX\_VALUE**, **PMIX\_ENUM\_VALUE**, or a combination of them. For example, an attribute that supports all positive integers might delineate it by including a **pmix\_info\_t** with a key of **PMIX\_MIN\_VALUE**, type of **PMIX\_INT**, and value of zero. The lack of an entry for **PMIX\_MAX\_VALUE** indicates that there is no ceiling to the range of accepted values.
- ninfo* indicates the number of elements in the *info* array
- The *description* field consists of a **NULL**-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values - e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the *description* and the number of elements in the *info* array is implied or required.

The attribute *name* and *string* fields must be **NULL**-terminated strings composed of standard alphanumeric values supported by common utilities such as *strcmp*.

Although not strictly required, both PMIx library implementers and host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them.

### 16.2.12.3 Attribute registration structure descriptive attributes

The following attributes relate to the nature of the values being reported in the **pmix\_regattr\_t** structures.

**PMIX\_MAX\_VALUE** "pmix.descr.maxval" (**varies**)

Used in **pmix\_regattr\_t** to describe the maximum valid value for the associated attribute.

**PMIX\_MIN\_VALUE** "pmix.descr.minval" (**varies**)

Used in **pmix\_regattr\_t** to describe the minimum valid value for the associated attribute.

**PMIX\_ENUM\_VALUE** "pmix.descr.enum" (**char\***)

Used in **pmix\_regattr\_t** to describe accepted values for the associated attribute.

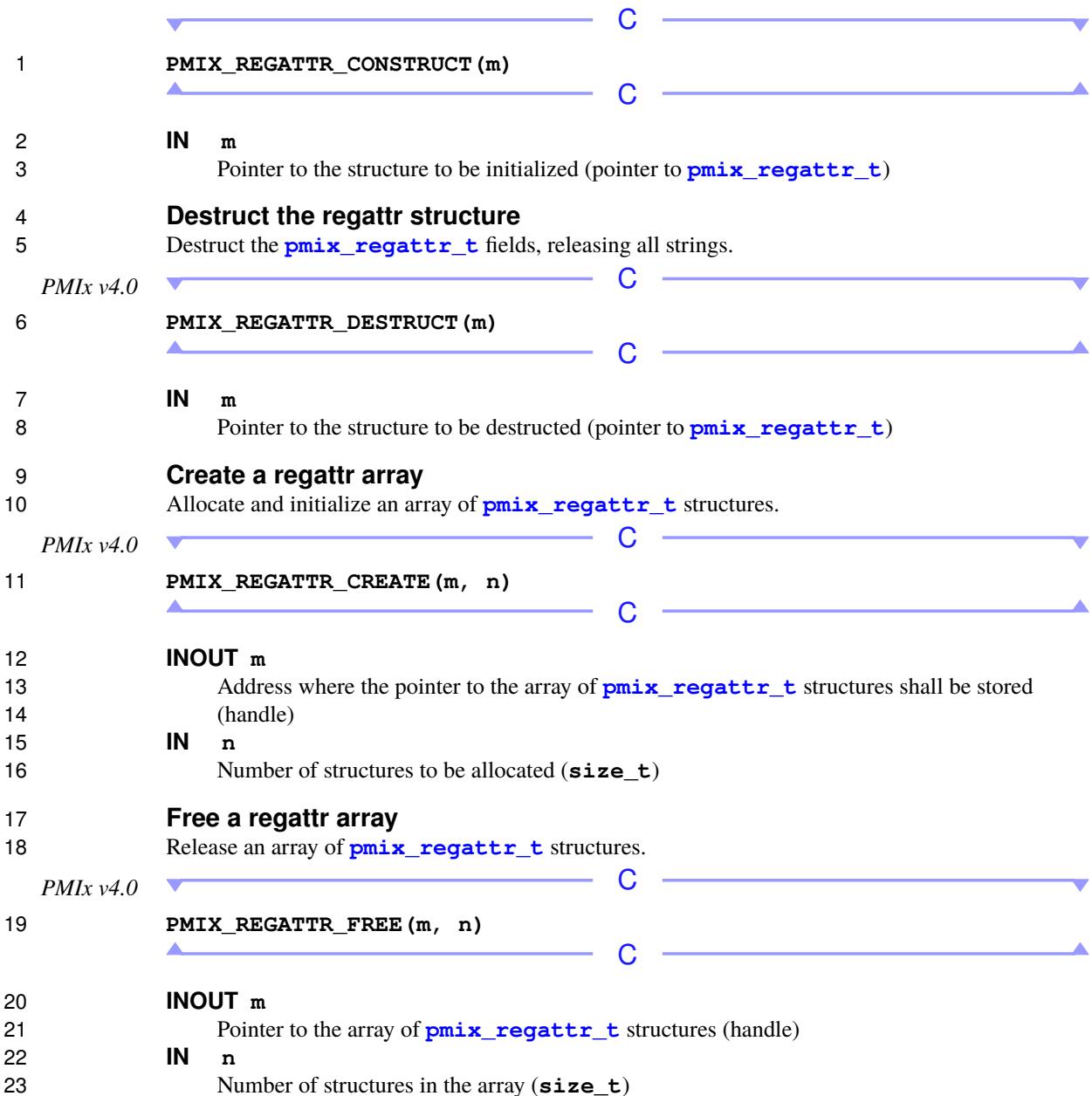
Numerical values shall be presented in a form convertible to the attribute's declared data type. Named values (i.e., values defined by constant names via a typical C-language enum declaration) must be provided as their numerical equivalent.

### 16.2.12.4 Attribute registration structure support macros

The following macros are provided to support the **pmix\_regattr\_t** structure.

#### Initialize the regattr structure

Initialize the **pmix\_regattr\_t** fields



1           **Load a regattr structure**  
2           Load values into a `pmix_regattr_t` structure. The macro can be called multiple times to add as  
3           many strings as desired to the same structure by passing the same address and a `NULL` key to the  
4           macro. Note that the `t` type value must be given each time.

PMIx v4.0

C

5           `PMIX_REGATTR_LOAD(a, n, k, t, ni, v)`

C

6           **IN**    `a`  
7            Pointer to the structure to be loaded (pointer to `pmix_proc_t`)  
8           **IN**    `n`  
9            String name of the attribute (string)  
10          **IN**    `k`  
11          Key value to be loaded (`pmix_key_t`)  
12          **IN**    `t`  
13          Type of data associated with the provided key (`pmix_data_type_t`)  
14          **IN**    `ni`  
15          Number of `pmix_info_t` elements to be allocated in `info` (`size_t`)  
16          **IN**    `v`  
17          One-line description to be loaded (more can be added separately) (string)

18           **Transfer a regattr to another regattr**

19           Non-destructively transfer the contents of a `pmix_regattr_t` structure to another one.

PMIx v4.0

C

20          `PMIX_REGATTR_XFER(m, n)`

C

21          **INOUT** `m`  
22            Pointer to the destination `pmix_regattr_t` structure (handle)  
23          **IN**    `m`  
24            Pointer to the source `pmix_regattr_t` structure (handle)

25        **16.2.13 PMIx\_server\_setup\_local\_support**

26           **Summary**  
27           Provide a function by which the local PMIx server can perform any application-specific operations  
28           prior to spawning local clients of a given application.

```
1 Format  
2 PMIx v2.0  
3 pmix_status_t  
4 PMIx_server_setup_local_support(const pmix_nspace_t nspace,  
5                                     pmix_info_t info[], size_t ninfo,  
6                                     pmix_op_cfunc_t cfunc,  
7                                     void *cbdata);
```

|           |               |                                                                |
|-----------|---------------|----------------------------------------------------------------|
| <b>IN</b> | <b>nspace</b> | Namespace (string)                                             |
| <b>IN</b> | <b>info</b>   | Array of info structures (array of handles)                    |
| <b>IN</b> | <b>ninfo</b>  | Number of elements in the <i>info</i> array ( <b>size_t</b> )  |
| <b>IN</b> | <b>cbfunc</b> | Callback function <b>pmix_op_cbfunc_t</b> (function reference) |
| <b>IN</b> | <b>cbdata</b> | Data to be passed to the callback function (memory reference)  |

Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
  - **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

## Description

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a fabric library might need to setup the local driver for “instant on” addressing. The data provided in the *info* array is the data returned to the host RM by the callback function executed as a result of a call to

## PMIx server setup application.

## Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting any local application processes from the specified namespace if information was obtained from a call to [PMIx\\_server\\_setup\\_application](#).

## 16.2.14 PMIx\_server\_IOF\_deliver

### Summary

Provide a function by which the host environment can pass forwarded Input/Output (IO) to the PMIx server library for distribution to its clients.

### Format

PMIx v3.0

```
6 pmix_status_t
7 PMIx_server_IOF_deliver(const pmix_proc_t *source,
8                           pmix_iof_channel_t channel,
9                           const pmix_byte_object_t *bo,
10                          const pmix_info_t info[], size_t ninfo,
11                          pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

C

#### IN source

Pointer to `pmix_proc_t` identifying source of the IO (handle)

#### IN channel

IO channel of the data (`pmix_iof_channel_t`)

#### IN bo

Pointer to `pmix_byte_object_t` containing the payload to be delivered (handle)

#### IN info

Array of `pmix_info_t` metadata describing the data (array of handles)

#### IN ninfo

Number of elements in the *info* array (`size_t`)

#### IN cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

#### IN cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

1           **Description**

2       Provide a function by which the host environment can pass forwarded IO to the PMIx server library  
3       for distribution to its clients. The PMIx server library is responsible for determining which of its  
4       clients have actually registered for the provided data and delivering it. The *cbfunc* callback function  
5       will be called once the PMIx server library no longer requires access to the provided data.

6   **16.2.15 PMIx\_server\_collect\_inventory**

7           **Summary**

8       Collect inventory of resources on a node.

9           **Format**

PMIx v3.0

C

```
10      pmix_status_t
11     PMIx_server_collect_inventory(const pmix_info_t directives[],
12                                        size_t ndirs,
13                                        pmix_info_cbfunc_t cbfunc,
14                                        void *cbdata);
```

C

15           **IN    directives**

16        Array of [pmix\\_info\\_t](#) directing the request (array of handles)

17           **IN    ndirs**

18        Number of elements in the *directives* array ([size\\_t](#))

19           **IN    cbfunc**

20        Callback function to return collected data ([pmix\\_info\\_cbfunc\\_t](#) function reference)

21           **IN    cbdata**

22        Data to be passed to the callback function (memory reference)

23       Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant. In the event  
24       the function returns an error, the *cbfunc* will not be called.

25           **Description**

26       Provide a function by which the host environment can request its PMIx server library collect an  
27       inventory of local resources. Supported resources depends upon the PMIx implementation, but may  
28       include the local node topology and fabric interfaces.

29           **Advice to PMIx server hosts**

30       This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested  
31       information. Inventory collection is expected to be a rare event – at system startup and upon  
32       command from a system administrator. Inventory updates are expected to initiate a smaller  
33       operation involving only the changed information. For example, replacement of a node would  
34       generate an event to notify the scheduler with an inventory update without invoking a global  
      inventory operation.

## 16.2.16 PMIx\_server\_deliver\_inventory

### 2 Summary

3 Pass collected inventory to the PMIx server library for storage.

### 4 Format

5 *PMIx v3.0*

C

```
6     pmix_status_t
7     PMIx_server_deliver_inventory(const pmix_info_t info[],
8                                     size_t ninfo,
9                                     const pmix_info_t directives[],
10                                    size_t ndirs,
11                                    pmix_op_cbfunc_t cbfunc,
12                                    void *cbdata);
```

C

12 IN **info**

13 Array of **pmix\_info\_t** containing the inventory (array of handles)

14 IN **ninfo**

15 Number of elements in the *info* array (**size\_t**)

16 IN **directives**

17 Array of **pmix\_info\_t** directing the request (array of handles)

18 IN **ndirs**

19 Number of elements in the *directives* array (**size\_t**)

20 IN **cbfunc**

21 Callback function **pmix\_op\_cbfunc\_t** (function reference)

22 IN **cbdata**

23 Data to be passed to the callback function (memory reference)

24 Returns one of the following:

- 25 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
26 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
27 function prior to returning from the API.
- 28 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
29 returned *success* - the *cbfunc* will not be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the *cbfunc* will not be called

1    **Description**

2    Provide a function by which the host environment can pass inventory information obtained from a  
3    node (as a result of a call to [PMIx\\_server\\_collect\\_inventory](#)) to the PMIx server library  
4    for storage. Inventory data is subsequently used by the PMIx server library for allocations in  
5    response to [PMIx\\_server\\_setup\\_application](#), and may be available to the library's host  
6    via the [PMIx\\_Get](#) API (depending upon PMIx implementation). The *cbfunc* callback function  
7    will be called once the PMIx server library no longer requires access to the provided data.

8    **16.2.17 PMIx\_server\_generate\_locality\_string**

9    **Summary**

10   Generate a PMIx locality string from a given cpuset.

11   **Format**

PMIx v4.0

```
12   pmix_status_t
13   PMIx_server_generate_locality_string(const pmix_cpuset_t *cpuset,
14     char **locality);
```

15   **IN cpuset**

16   Pointer to a [pmix\\_cpuset\\_t](#) containing the bitmap of assigned PUs (handle)

17   **OUT locality**

18   String representation of the PMIx locality corresponding to the input bitmap ([char\\*](#))

19   Returns either [PMIX\\_SUCCESS](#) indicating that the returned string contains the locality, or an  
20   appropriate PMIx error constant.

21   **Description**

22   Provide a function by which the host environment can generate a PMIx locality string for inclusion  
23   in the call to [PMIx\\_server\\_register\\_nspace](#). This function shall only be called for local  
24   client processes, with the returned locality included in the job-level information (via the  
25   [PMIX\\_LOCALITY\\_STRING](#) attribute) provided to local clients. Local clients can use these  
26   strings as input to determine the relative locality of their local peers via the  
27   [PMIx\\_Get\\_relative\\_locality](#) API.

28   The function is required to return a string prefixed by the *source* field of the provided *cpuset*  
29   followed by a colon. The remainder of the string shall represent the corresponding locality as  
30   expressed by the underlying implementation.

31   **16.2.18 PMIx\_server\_generate\_cpuset\_string**

32   **Summary**

33   Generate a PMIx string representation of the provided cpuset.

**IN cpuset**  
Pointer to a [pmix\\_cpuset\\_t](#) containing the bitmap of assigned PUs (handle)

**OUT cpuset\_string**  
String representation of the input bitmap (**char\***)

Returns either **PMIX\_SUCCESS** indicating that the returned string contains the representation, or an appropriate PMIx error constant.

## Description

Provide a function by which the host environment can generate a string representation of the cpuset bitmap for inclusion in the call to **PMIx\_server\_register\_nspace**. This function shall only be called for local client processes, with the returned string included in the job-level information (via the **PMIX\_CPUSET** attribute) provided to local clients. Local clients can use these strings as input to obtain their PU bindings via the **PMIx\_Get\_cpuset** API.

The function is required to return a string prefixed by the *source* field of the provided *cpuset* followed by a colon. The remainder of the string shall represent the PUs to which the process is bound as expressed by the underlying implementation.

## 20 16.2.18.1 Cpuset Structure

The `pmix_cpuset_t` structure contains a character string identifying the source of the bitmap (e.g., "hwloc") and a pointer to the corresponding implementation-specific structure (e.g., `hwloc_cpuset_t`).

```
typedef struct pmix_cpuset {
    char *source;
    void *bitmap;
} pmix_cpuset_t;
```

## 28 16.2.18.2 Cpuset support macros

The following macros support the `pmix_cpuset_t` structure.

1           **Initialize the cpuset structure**  
2        Initialize the `pmix_cpuset_t` fields.  
3            PMIx v4.0      C  
4            **IN    m**  
5            Pointer to the structure to be initialized (pointer to `pmix_cpuset_t`)  
6           **Create a cpuset array**  
7        Allocate and initialize a `pmix_cpuset_t` array.  
8            PMIx v4.0      C  
9           **INOUT m**  
10          Address where the pointer to the array of `pmix_cpuset_t` structures shall be stored  
11          (handle)  
12          **IN    n**  
13          Number of structures to be allocated (`size_t`)

## 14 16.2.19 `PMIx_server_define_process_set`

15           **Summary**  
16        Define a PMIx process set.  
17           **Format**  
18            PMIx v4.0      C  
19            `pmix_status_t`  
20            `PMIx_server_define_process_set(const pmix_proc_t members[],`  
21                            `size_t nmembers,`  
22                            `char *pset_name);`  
23           **IN    members**  
24            Pointer to an array of `pmix_proc_t` containing the identifiers of the processes in the  
25            process set (handle)  
26           **IN    nmembers**  
27            Number of elements in `members` (integer)  
28           **IN    pset\_name**  
29            String name of the process set being defined (`char*`)  
30            Returns either `PMIX_SUCCESS` or an appropriate PMIx error constant.

1           **Description**

2       Provide a function by which the host environment can create a process set. The PMIx server shall  
3       alert all local clients of the new process set (including process set name and membership) via the  
4       [PMIX\\_PROCESS\\_SET\\_DEFINE](#) event.

5            **Advice to PMIx server hosts** 

6       The host environment is responsible for ensuring:

- 7
  - 8       • consistent knowledge of process set membership across all involved PMIx servers; and
  - 8       • that process set names do not conflict with system-assigned namespaces within the scope of the
  - 8       set

9   **16.2.20 PMIx\_server\_delete\_process\_set**

10          **Summary**

11       Delete a PMIx process set name

12          **Format**

13       *PMIx v4.0*

14        **C**   
`pmix_status_t  
PMIx_server_delete_process_set(char *pset_name);`  
 **C** 

15          **IN pset\_name**

16       String name of the process set being deleted (`char*`)

17       Returns either [PMIX\\_SUCCESS](#) or an appropriate PMIx error constant.

18          **Description**

19       Provide a function by which the host environment can delete a process set name. The PMIx server  
20       shall alert all local clients of the process set name being deleted via the  
21       [PMIX\\_PROCESS\\_SET\\_DELETE](#) event. Deletion of the name has no impact on the member  
22       processes.

23            **Advice to PMIx server hosts** 

24       The host environment is responsible for ensuring consistent knowledge of process set membership  
across all involved PMIx servers.  


## 1 16.3 Server Function Pointers

2 PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the  
3 protocol. This method allows RMs to implement the server without being burdened with PMIx  
4 internal details. When a request is received from the client, the corresponding server function will  
5 be called with the information.

6 Any functions not supported by the RM can be indicated by a **NULL** for the function pointer. PMIx  
7 implementations are required to return a **PMIX\_ERR\_NOT\_SUPPORTED** status to all calls to  
8 functions that require host environment support and are not backed by a corresponding server  
9 module entry.

10 The host RM will provide the function pointers in a **pmix\_server\_module\_t** structure passed  
11 to **PMIx\_server\_init**. That module structure and associated function references are defined in  
12 this section.

### Advice to PMIx server hosts

13 For performance purposes, the host server is required to return as quickly as possible from all  
14 functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx  
15 server support library to handle multiple client requests as quickly and scalably as possible.

16 All data passed to the host server functions is "owned" by the PMIx server support library and  
17 must not be free'd. Data returned by the host server via callback function is owned by the host  
18 server, which is free to release it upon return from the callback

### 19 16.3.1 pmix\_server\_module\_t Module

#### 20 Summary

21 List of function pointers that a PMIx server passes to **PMIx\_server\_init** during startup.

#### 22 Format

```

1  typedef struct pmix_server_module_4_0_0_t {
2      /* v1x interfaces */
3      pmix_server_client_connected_fn_t    client_connected; // DEPRECATED
4      pmix_server_client_finalized_fn_t   client_finalized;
5      pmix_server_abort_fn_t            abort;
6      pmix_server_fencenb_fn_t          fence_nb;
7      pmix_server_dmodex_req_fn_t       direct_modex;
8      pmix_server_publish_fn_t         publish;
9      pmix_server_lookup_fn_t          lookup;
10     pmix_server_unpublish_fn_t       unpublish;
11     pmix_server_spawn_fn_t          spawn;
12     pmix_server_connect_fn_t        connect;
13     pmix_server_disconnect_fn_t     disconnect;
14     pmix_server_register_events_fn_t register_events;
15     pmix_server_deregister_events_fn_t deregister_events;
16     pmix_server_listener_fn_t       listener;
17     /* v2x interfaces */
18     pmix_server_notify_event_fn_t    notify_event;
19     pmix_server_query_fn_t          query;
20     pmix_server_tool_connection_fn_t tool_connected;
21     pmix_server_log_fn_t           log;
22     pmix_server_alloc_fn_t          allocate;
23     pmix_server_job_control_fn_t    job_control;
24     pmix_server_monitor_fn_t        monitor;
25     /* v3x interfaces */
26     pmix_server_get_cred_fn_t       get_credential;
27     pmix_server_validate_cred_fn_t  validate_credential;
28     pmix_server_iof_fn_t           iof_pull;
29     pmix_server_stdin_fn_t          push_stdin;
30     /* v4x interfaces */
31     pmix_server_grp_fn_t           group;
32     pmix_server_fabric_fn_t        fabric;
33     pmix_server_client_connected2_fn_t client_connected2;
34 } pmix_server_module_t;
```

**Advice to PMIx server hosts**

Note that some PMIx implementations *require* the use of C99-style designated initializers to clearly correlate each provided function pointer with the correct member of the **pmix\_server\_module\_t** structure as the location/ordering of struct members may change over time.

1 **16.3.2 pmix\_server\_client\_connected\_fn\_t**

2 **Summary**

3 Notify the host server that a client connected to this server. This function module entry has been  
4 **DEPRECATED** in favor of [pmix\\_server\\_client\\_connected2\\_fn\\_t](#).

5 **Format**

PMIx v1.0

C

```
6     typedef pmix_status_t (*pmix_server_client_connected_fn_t) (
7         const pmix_proc_t *proc,
8         void* server_object,
9         pmix_op_cbfunc_t cbfunc,
10        void *cbdata);
```

C

11 **IN proc**

12 **pmix\_proc\_t** structure (handle)

13 **IN server\_object**

14 object reference (memory reference)

15 **IN cbfunc**

16 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

17 **IN cbdata**

18 Data to be passed to the callback function (memory reference)

19 Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

27 **Description**

28 This function module entry has been DEPRECATED in favor of  
29 [pmix\\_server\\_client\\_connected2\\_fn\\_t](#). If both functions are provided, the PMIx  
30 library will ignore this function module entry in favor of its replacement.

31 **16.3.3 pmix\_server\_client\_connected2\_fn\_t**

32 **Summary**

33 Notify the host server that a client connected to this server - this version of the original function  
34 definition has been extended to include an array of [pmix\\_info\\_t](#), thereby allowing the PMIx  
35 server library to pass additional information identifying the client to the host environment.

1           **Format**

2       **typedef pmix\_status\_t (\*pmix\_server\_client\_connected2\_fn\_t) (**  
3                           **const pmix\_proc\_t \*proc,**  
4                           **void\* server\_object,**  
5                           **pmix\_info\_t info[], size\_t ninfo,**  
6                           **pmix\_op\_cfunc\_t cfunc,**  
7                           **void \*cbdata)**

8       **IN proc**  
9           **pmix\_proc\_t** structure (handle)  
10      **IN server\_object**  
11       object reference (memory reference)  
12      **IN info**  
13       Array of info structures (array of handles)  
14      **IN ninfo**  
15       Number of elements in the *info* array (integer)  
16      **IN cfunc**  
17       Callback function **pmix\_op\_cfunc\_t** (function reference)  
18      **IN cbdata**  
19       Data to be passed to the callback function (memory reference)

20     Returns one of the following:

- 21     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
22       will be returned in the provided *cfunc*. Note that the host must not invoke the callback function  
23       prior to returning from the API.
- 24     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
25       returned *success* - the *cfunc* will not be called
- 26     • a PMIx error constant indicating either an error in the input or that the request was immediately  
27       processed and failed - the *cfunc* will not be called. The PMIx server library is to immediately  
28       terminate the connection.

29           **Description**

30     Notify the host environment that a client has called **PMIx\_Init**. Note that the client will be in a  
31       blocked state until the host server executes the callback function, thus allowing the PMIx server  
32       support library to release the client. The *server\_object* parameter will be the value of the  
33       *server\_object* parameter passed to **PMIx\_server\_register\_client** by the host server when  
34       registering the connecting client. A host server can choose to not be notified when clients connect  
35       by setting **pmix\_server\_client\_connected2\_fn\_t** to **NULL**.

36     It is possible that only a subset of the clients in a namespace call **PMIx\_Init**. The server's  
37       **pmix\_server\_client\_connected2\_fn\_t** implementation should therefore not depend on

1 being called once per rank in a namespace or delay calling the callback function until all ranks have  
2 connected. However, the host may rely on the `pmix_server_client_connected2_fn_t`  
3 function module entry being called for a given rank prior to any other function module entries  
4 being executed on behalf of that rank.

5 **16.3.4 `pmix_server_client_finalized_fn_t`**

6 **Summary**

7 Notify the host environment that a client called `PMIx_Finalize`.

8 **Format**

9 *PMIx v1.0*

```
10     typedef pmix_status_t (*pmix_server_client_finalized_fn_t)(
11         const pmix_proc_t *proc,
12         void* server_object,
13         pmix_op_cbfunc_t cbfunc,
14         void *cbdata);
```

15 **IN proc**  
16 `pmix_proc_t` structure (handle)  
17 **IN server\_object**  
18 object reference (memory reference)  
19 **IN cbfunc**  
20 Callback function `pmix_op_cbfunc_t` (function reference)  
21 **IN cbdata**  
22 Data to be passed to the callback function (memory reference)

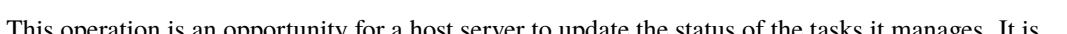
23 Returns one of the following:

- 24
  - `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result  
25 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
prior to returning from the API.
  - `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and  
returned *success* - the *cbfunc* will not be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately  
processed and failed - the *cbfunc* will not be called

1           **Description**

2       Notify the host environment that a client called **PMIx\_Finalize**. Note that the client will be in a  
3       blocked state until the host server executes the callback function, thus allowing the PMIx server  
4       support library to release the client. The **server\_object** parameter will be the value of the  
5       **server\_object** parameter passed to **PMIx\_server\_register\_client** by the host server when  
6       registering the connecting client. If provided, an implementation of  
7       **pmix\_server\_client\_finalized\_fn\_t** is only required to call the callback function  
8       designated. A host server can choose to not be notified when clients finalize by setting  
9       **pmix\_server\_client\_finalized\_fn\_t** to **NULL**.

10      Note that the host server is only being informed that the client has called **PMIx\_Finalize**. The  
11     client might not have exited. If a client exits without calling **PMIx\_Finalize**, the server support  
12     library will not call the **pmix\_server\_client\_finalized\_fn\_t** implementation.

13            **Advice to PMIx server hosts** 

14      This operation is an opportunity for a host server to update the status of the tasks it manages. It is  
also a convenient and well defined time to release resources used to support that client.

15      **16.3.5 pmix\_server\_abort\_fn\_t**

16           **Summary**

17      Notify the host environment that a local client called **PMIx\_Abort**.

18           **Format**

19       C 

PMIx v1.0

```
20      typedef pmix_status_t (*pmix_server_abort_fn_t)(
21           const pmix_proc_t *proc,
22           void *server_object,
23           int status,
24           const char msg[],
25           pmix_proc_t procs[],
26           size_t nprocs,
27           pmix_op_cbfunc_t cbfunc,
28           void *cbdata);
```

```

1   IN  proc
2     pmix_proc_t structure identifying the process requesting the abort (handle)
3   IN  server_object
4     object reference (memory reference)
5   IN  status
6     exit status (integer)
7   IN  msg
8     exit status message (string)
9   IN  procs
10    Array of pmix_proc_t structures identifying the processes to be terminated (array of
11    handles)
12  IN  nprocs
13    Number of elements in the procs array (integer)
14  IN  cbfunc
15    Callback function pmix_op_cbfunc_t (function reference)
16  IN  cbdata
17    Data to be passed to the callback function (memory reference)

18 Returns one of the following:
19
20  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
21    will be returned in the provided cbfunc. Note that the host must not invoke the callback function
22    prior to returning from the API.
23
24  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
25    returned success - the cbfunc will not be called
26
27  • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
28    request, even though the function entry was provided in the server module - the cbfunc will not
    be called
29
30  • a PMIx error constant indicating either an error in the input or that the request was immediately
31    processed and failed - the cbfunc will not be called

```

## 29 Description

30 A local client called **PMIx\_Abort**. Note that the client will be in a blocked state until the host  
31 server executes the callback function, thus allowing the PMIx server library to release the client.  
32 The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all  
33 processes in the client's namespace are to be terminated.

## 34 16.3.6 pmix\_server\_fencenb\_fn\_t

### 35 Summary

36 At least one client called either **PMIx\_Fence** or **PMIx\_Fence\_nb**.

1           **Format**2       *PMIx v1.0*

C

```
3       typedef pmix_status_t (*pmix_server_fencenb_fn_t)(
4                           const pmix_proc_t procs[],
5                           size_t nprocs,
6                           const pmix_info_t info[],
7                           size_t ninfo,
8                           char *data, size_t ndata,
9                           pmix_modex_cbfunc_t cbfunc,
                         void *cbdata);
```

C

10      **IN procs**11        Array of **pmix\_proc\_t** structures identifying operation participants(array of handles)12      **IN nprocs**13        Number of elements in the *procs* array (integer)14      **IN info**

15        Array of info structures (array of handles)

16      **IN ninfo**17        Number of elements in the *info* array (integer)18      **IN data**

19        (string)

20      **IN ndata**

21        (integer)

22      **IN cbfunc**23        Callback function **pmix\_modex\_cbfunc\_t** (function reference)24      **IN cbdata**

25        Data to be passed to the callback function (memory reference)

26        Returns one of the following:

- 27        • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
28           will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
29           prior to returning from the API.
- 30        • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
31           returned *success* - the *cbfunc* will not be called
- 32        • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
33           request, even though the function entry was provided in the server module - the *cbfunc* will not  
34           be called
- 35        • a PMIx error constant indicating either an error in the input or that the request was immediately  
36           processed and failed - the *cbfunc* will not be called

## Required Attributes

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.

2 The following attributes are required to be supported by all host environments:

3 **PMIX\_COLLECT\_DATA** "pmix.collect" (bool)

4 Collect all data posted by the participants using **PMIx\_Put** that has been committed via  
5 **PMIx\_Commit**, making the collection locally available to each participant at the end of the  
6 operation. By default, this will include all job-level information that was locally generated  
7 by PMIx servers unless excluded using the **PMIX\_COLLECT\_GENERATED\_JOB\_INFO**  
8 attribute.

## Optional Attributes

9 The following attributes are optional for host environments:

10 **PMIX\_TIMEOUT** "pmix.timeout" (int)

11 Time in seconds before the specified operation should time out (zero indicating infinite) and  
12 return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
13 caused by multiple layers (client, server, and host) simultaneously timing the operation.

## Advice to PMIx server hosts

14 Host environment are required to return **PMIX\_ERR\_NOT\_SUPPORTED** if passed an attributed  
15 marked as **PMIX\_INFO\_REQD** that they do not support, even if support for that attribute is  
16 optional.

## 1           Description

2       All local clients in the provided array of *procs* called either **PMIx\_Fence** or **PMIx\_Fence\_nb**.  
3       In either case, the host server will be called via a non-blocking function to execute the specified  
4       operation once all participating local processes have contributed. All processes in the specified  
5       *procs* array are required to participate in the **PMIx\_Fence/PMIx\_Fence\_nb** operation. The  
6       callback is to be executed once every daemon hosting at least one participant has called the host  
7       server's **pmix\_server\_fencenb\_fn\_t** function.

8       The provided data is to be collectively shared with all PMIx servers involved in the fence operation,  
9       and returned in the modeX *cbfunc*. A **NULL** data value indicates that the local processes had no data  
10      to contribute.

11      The array of *info* structs is used to pass user-requested options to the server. This can include  
12      directives as to the algorithm to be used to execute the fence operation. The directives are optional  
13      unless the **PMIX\_INFO REQD** flag has been set - in such cases, the host RM is required to return  
14      an error if the directive cannot be met.

### Advice to PMIx library implementers

15      The PMIx server library is required to aggregate participation by local clients, passing the request  
16      to the host environment once all local participants have executed the API.

### Advice to PMIx server hosts

17      The host will receive a single call for each collective operation. It is the responsibility of the host to  
18      identify the nodes containing participating processes, execute the collective across all participating  
19      nodes, and notify the local PMIx server library upon completion of the global collective. Data  
20      received from each node must be simply concatenated to form an aggregated unit, as shown in the  
21      following example:

```
22      uint8_t *blob1, *blob2, *total;
23      size_t sz_blob1, sz_blob2, sz_total;
24
25      sz_total = sz_blob1 + sz_blob2;
26      total = (uint8_t*)malloc(sz_total);
27      memcpy(total, blob1, sz_blob1);
28      memcpy(&total[sz_blob1], blob2, sz_blob2);
```

29      Note that the ordering of the data blobs does not matter. The host is responsible for free'ing the  
30      *data* object passed to it by the PMIx server library.

## 16.3.6.1 Modex Callback Function

### Summary

The `pmix_modex_cbfunc_t` is used by the `pmix_server_fencenb_fn_t` and `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex Business Card Exchange (BCX) data.

PMIx v1.0

C

```
6     typedef void (*pmix_modex_cbfunc_t)
7         (pmix_status_t status,
8          const char *data, size_t ndata,
9           void *cbdata,
10          pmix_release_cbfunc_t release_fn,
11          void *release_cbdata);
```

C

#### IN `status`

Status associated with the operation (handle)

#### IN `data`

Data to be passed (pointer)

#### IN `ndata`

size of the data (`size_t`)

#### IN `cbdata`

Callback data passed to original API call (memory reference)

#### IN `release_fn`

Callback for releasing `data` (function pointer)

#### IN `release_cbdata`

Pointer to be passed to `release_fn` (memory reference)

### Description

A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data in response to “fence” and “get” operations. The returned blob contains the data collected from each server participating in the operation.

## 16.3.7 `pmix_server_dmodex_req_fn_t`

### Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified process to obtain and return a direct modex blob for that process.

1           **Format**

2        *PMIx v1.0*

```
3        typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
4                           const pmix_proc_t *proc,
5                           const pmix_info_t info[],
6                           size_t ninfo,
7                           pmix_modex_cfunc_t cfunc,
8                           void *cbdata);
```

C

8        **IN proc**  
9            *pmix\_proc\_t* structure identifying the process whose data is being requested (handle)  
10      **IN info**  
11        Array of info structures (array of handles)  
12      **IN ninfo**  
13        Number of elements in the *info* array (integer)  
14      **IN cfunc**  
15        Callback function *pmix\_modex\_cfunc\_t* (function reference)  
16      **IN cbdata**  
17        Data to be passed to the callback function (memory reference)

18     Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the *cfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cfunc* will not be called

27           **Required Attributes**

28     PMIx libraries are required to pass any provided attributes to the host environment for processing.

29           **Optional Attributes**

30     The following attributes are optional for host environments that support this operation:

31      **PMIX\_TIMEOUT "pmix.timeout" (int)**

32        Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

1           **Description**

2           Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
3           hosts the specified proc to obtain and return any information that process posted via calls to  
4           **PMIx\_Put** and **PMIx\_Commit**.

5           The array of *info* structs is used to pass user-requested options to the server. This can include a  
6           timeout to preclude an indefinite wait for data that may never become available. The directives are  
7           optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an  
8           error if the directive cannot be met.

9         **16.3.7.1 Dmodex attributes**

10           **PMIX\_REQUIRED\_KEY "pmix.req.key" (char\*)**

11           Key the user needs prior to responding from a dmodex request.

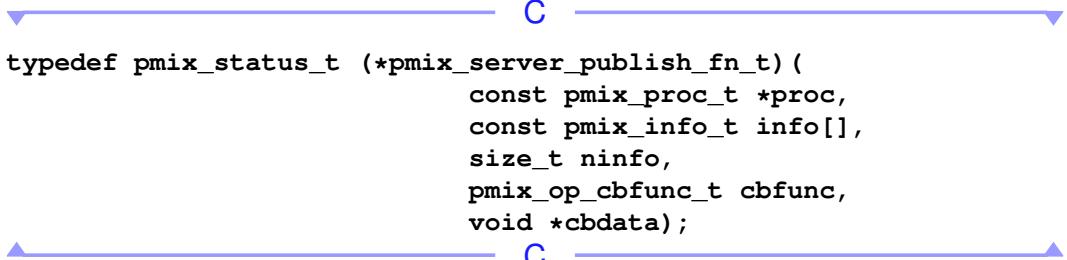
12         **16.3.8 pmix\_server\_publish\_fn\_t**

13           **Summary**

14           Publish data per the PMIx API specification.

15           **Format**

PMIx v1.0



```
16         typedef pmix_status_t (*pmix_server_publish_fn_t)(
17                                 const pmix_proc_t *proc,
18                                 const pmix_info_t info[],
19                                 size_t ninfo,
20                                 pmix_op_cbfunc_t cbfunc,
21                                 void *cbdata);
```

22         **IN proc**

23           **pmix\_proc\_t** structure of the process publishing the data (handle)

24         **IN info**

25           Array of info structures (array of handles)

26         **IN ninfo**

27           Number of elements in the *info* array (integer)

28         **IN cbfunc**

29           Callback function **pmix\_op\_cbfunc\_t** (function reference)

30         **IN cbdata**

31           Data to be passed to the callback function (memory reference)

32           Returns one of the following:

- 33           • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
34           will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
35           prior to returning from the API.

- 1     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
2        returned *success* - the *cbfunc* will not be called  
3     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
4        request, even though the function entry was provided in the server module - the *cbfunc* will not  
5        be called  
6     • a PMIx error constant indicating either an error in the input or that the request was immediately  
7        processed and failed - the *cbfunc* will not be called

### Required Attributes

8     PMIx libraries are required to pass any provided attributes to the host environment for processing.  
9     In addition, the following attributes are required to be included in the passed *info* array:

- 10    **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
11       Effective user ID of the connecting process.  
12    **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
13       Effective group ID of the connecting process.

---

15     Host environments that implement this entry point are required to support the following attributes:

- 16    **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)  
17       Define constraints on the processes that can access the provided data. Only processes that  
18        meet the constraints are allowed to access it.  
19    **PMIX\_PERSISTENCE** "pmix.persist" (**pmix\_persistence\_t**)  
20       Declare how long the datastore shall retain the provided data. The datastore is to delete the  
21        data upon reaching the persistence criterion.

### Optional Attributes

22     The following attributes are optional for host environments that support this operation:

- 23    **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
24       Time in seconds before the specified operation should time out (zero indicating infinite) and  
25        return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
26        caused by multiple layers (client, server, and host) simultaneously timing the operation.

1           **Description**

2       Publish data per the **PMIx\_Publish** specification. The callback is to be executed upon  
3       completion of the operation. The default data range is left to the host environment, but expected to  
4       be **PMIX\_RANGE\_SESSION**, and the default persistence **PMIX\_PERSIST\_SESSION** or their  
5       equivalent. These values can be specified by including the respective attributed in the *info* array.

6       The persistence indicates how long the server should retain the data.

7            **Advice to PMIx server hosts**

8       The host environment is not required to guarantee support for any specific range - i.e., the  
9       environment does not need to return an error if the data store doesn't support a specified range so  
10      long as it is covered by some internally defined range. However, the server must return an error (a)  
11      if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of  
12      published info by the original publisher - it is left to the discretion of the host environment to allow  
      info-key-based flags to modify this behavior.

13     The **PMIX\_USERID** and **PMIX\_GRPID** of the publishing process will be provided to support  
14     authorization-based access to published information and must be returned on any subsequent  
15     lookup request.

16           

16   **16.3.9 pmix\_server\_lookup\_fn\_t**

17           **Summary**

18       Lookup published data.

19           **Format**

PMIx v1.0            C

```
20       typedef pmix_status_t (*pmix_server_lookup_fn_t)(
21                   const pmix_proc_t *proc,
22                   char ***keys,
23                   const pmix_info_t info[],
24                   size_t ninfo,
25                   pmix_lookup_cbfunc_t cbfunc,
26                   void *cbdata);
```

```

1   IN  proc
2     pmix_proc_t structure of the process seeking the data (handle)
3   IN  keys
4     (array of strings)
5   IN  info
6     Array of info structures (array of handles)
7   IN  ninfo
8     Number of elements in the info array (integer)
9   IN  cbfunc
10    Callback function pmix_lookup_cbfunc_t (function reference)
11   IN  cbdata
12    Data to be passed to the callback function (memory reference)

13 Returns one of the following:
14
15  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
16    will be returned in the provided cbfunc. Note that the host must not invoke the callback function
17    prior to returning from the API.
18
19  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
20    returned success - the cbfunc will not be called
21
22  • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
23    request, even though the function entry was provided in the server module - the cbfunc will not
    be called
24
25  • a PMIx error constant indicating either an error in the input or that the request was immediately
    processed and failed - the cbfunc will not be called

```

### Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

```

26  PMIX_USERID "pmix.euid" (uint32_t)
27    Effective user ID of the connecting process.
28
29  PMIX_GRPID "pmix.egid" (uint32_t)
30    Effective group ID of the connecting process.

```

---

Host environments that implement this entry point are required to support the following attributes:

```

32  PMIX_RANGE "pmix.range" (pmix_data_range_t)
33    Define constraints on the processes that can access the provided data. Only processes that
    meet the constraints are allowed to access it.
34

```

```
1 PMIX_WAIT "pmix.wait" (int)
2     Caller requests that the PMIx server wait until at least the specified number of values are
3     found (a value of zero indicates all and is the default).
```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

```
5 PMIX_TIMEOUT "pmix.timeout" (int)
6     Time in seconds before the specified operation should time out (zero indicating infinite) and
7     return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions
8     caused by multiple layers (client, server, and host) simultaneously timing the operation.
```

## Description

Lookup published data. The host server will be passed a **NULL**-terminated array of string keys identifying the data being requested.

The array of *info* structs is used to pass user-requested options to the server. The default data range is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**. This can include a wait flag to indicate that the server should wait for all data to become available before executing the callback function, or should immediately callback with whatever data is available. In addition, a timeout can be specified on the wait to preclude an indefinite wait for data that may never be published.

## Advice to PMIx server hosts

The **PMIX\_USERID** and **PMIX\_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

### 16.3.10 pmix\_server\_unpublish\_fn\_t

#### Summary

Delete data from the data store.

1           **Format**

2        *PMIx v1.0*

C

```
2        typedef pmix_status_t (*pmix_server_unpublish_fn_t) (
3                            const pmix_proc_t *proc,
4                            char **keys,
5                            const pmix_info_t info[],
6                            size_t ninfo,
7                            pmix_op_cbfunc_t cbfunc,
8                            void *cbdata);
```

C

9        **IN proc**

10            *pmix\_proc\_t* structure identifying the process making the request (handle)

11        **IN keys**

12            (array of strings)

13        **IN info**

14            Array of info structures (array of handles)

15        **IN ninfo**

16            Number of elements in the *info* array (integer)

17        **IN cbfunc**

18            Callback function *pmix\_op\_cbfunc\_t* (function reference)

19        **IN cbdata**

20            Data to be passed to the callback function (memory reference)

21        Returns one of the following:

- 22        • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23            will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24            prior to returning from the API.
- 25        • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26            returned *success* - the *cbfunc* will not be called
- 27        • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28            request, even though the function entry was provided in the server module - the *cbfunc* will not  
29            be called
- 30        • a PMIx error constant indicating either an error in the input or that the request was immediately  
31            processed and failed - the *cbfunc* will not be called

32           **Required Attributes**

33        PMIx libraries are required to pass any provided attributes to the host environment for processing.  
34        In addition, the following attributes are required to be included in the passed *info* array:

35        **PMIX\_USERID "pmix.euid" (uint32\_t)**

            Effective user ID of the connecting process.

1   **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2       Effective group ID of the connecting process.

---

4       Host environments that implement this entry point are required to support the following attributes:

5   **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)  
6       Define constraints on the processes that can access the provided data. Only processes that  
7       meet the constraints are allowed to access it.



### Optional Attributes

8       The following attributes are optional for host environments that support this operation:

9   **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
10      Time in seconds before the specified operation should time out (zero indicating infinite) and  
11      return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
12      caused by multiple layers (client, server, and host) simultaneously timing the operation.



### Description

14       Delete data from the data store. The host server will be passed a **NULL**-terminated array of string  
15       keys, plus potential directives such as the data range within which the keys should be deleted. The  
16       default data range is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**.  
17       The callback is to be executed upon completion of the delete procedure.

### Advice to PMIx server hosts

18       The **PMIX\_USERID** and **PMIX\_GRP\_ID** of the requesting process will be provided to support  
19       authorization-based access to published information. The host environment is not required to  
20       guarantee support for any specific range - i.e., the environment does not need to return an error if  
21       the data store doesn't support a specified range so long as it is covered by some internally defined  
22       range.



## 16.3.11 pmix\_server\_spawn\_fn\_t

### Summary

24       Spawn a set of applications/processes as per the **PMIX\_Spawn** API.

## 1 Format

PMIx v1.0

C

```
2     typedef pmix_status_t (*pmix_server_spawn_fn_t)(  
3         const pmix_proc_t *proc,  
4         const pmix_info_t job_info[],  
5         size_t ninfo,  
6         const pmix_app_t apps[],  
7         size_t napps,  
8         pmix_spawn_cbfunc_t cbfunc,  
9         void *cbdata);
```

C

10 **IN proc**

**pmix\_proc\_t** structure of the process making the request (handle)

11 **IN job\_info**

Array of info structures (array of handles)

12 **IN ninfo**

Number of elements in the *jobinfo* array (integer)

13 **IN apps**

Array of **pmix\_app\_t** structures (array of handles)

14 **IN napps**

Number of elements in the *apps* array (integer)

15 **IN cbfunc**

Callback function **pmix\_spawn\_cbfunc\_t** (function reference)

16 **IN cbdata**

Data to be passed to the callback function (memory reference)

24 Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

## Required Attributes

1 PMIx server libraries are required to pass any provided attributes to the host environment for  
2 processing. In addition, the following attributes are required to be included in the passed *info* array:

3   **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
4         Effective user ID of the connecting process.  
5   **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
6         Effective group ID of the connecting process.  
7   **PMIX\_SPAWNED** "pmix.spawned" (**bool**)  
8         **true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return  
9         status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.  
10   **PMIX\_PARENT\_ID** "pmix.parent" (**pmix\_proc\_t**)  
11         Process identifier of the parent process of the specified process - typically used to identify  
12         the application process that caused the job containing the specified process to be spawned  
13         (e.g., the process that called **PMIx\_Spawn**).  
14   **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (**bool**)  
15         The requesting process is a PMIx tool.  
16   **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (**bool**)  
17         The requesting process is a PMIx client.

---

18  
19 Host environments that provide this module entry point are required to pass the **PMIX\_SPAWNED**  
20 and **PMIX\_PARENT\_ID** attributes to all PMIx servers launching new child processes so those  
21 values can be returned to clients upon connection to the PMIx server. In addition, they are required  
22 to support the following attributes when present in either the *job\_info* or the *info* array of an  
23 element of the *apps* array:

24   **PMIX\_WDIR** "pmix.wdir" (**char\***)  
25         Working directory for spawned processes.  
26   **PMIX\_SET\_SESSION\_CWD** "pmix.ssncwd" (**bool**)  
27         Set the current working directory to the session working directory assigned by the RM - can  
28         be assigned to the entire job (by including attribute in the *job\_info* array) or on a  
29         per-application basis in the *info* array for each **pmix\_app\_t**.  
30   **PMIX\_PREFIX** "pmix.prefix" (**char\***)  
31         Prefix to use for starting spawned processes - i.e., the directory where the executables can be  
32         found.  
33   **PMIX\_HOST** "pmix.host" (**char\***)  
34         Comma-delimited list of hosts to use for spawned processes.  
35   **PMIX\_HOSTFILE** "pmix.hostfile" (**char\***)

1 Hostfile to use for spawned processes.

## Optional Attributes

2 The following attributes are optional for host environments that support this operation:

3 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (**char\***)  
4 Hostfile containing hosts to add to existing allocation.

5 **PMIX\_ADD\_HOST** "pmix.addhost" (**char\***)  
6 Comma-delimited list of hosts to add to the allocation.

7 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (**bool**)  
8 Preload executables onto nodes prior to executing launch procedure.

9 **PMIX\_PRELOAD\_FILES** "pmix.loadfiles" (**char\***)  
10 Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

11 **PMIX\_PERSONALITY** "pmix.pers" (**char\***)  
12 Name of personality corresponding to programming model used by application - supported  
13 values depend upon PMIx implementation.

14 **PMIX\_MAPPER** "pmix.mapper" (**char\***)  
15 Mapping mechanism to use for placing spawned processes - when accessed using  
16 **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping  
17 mechanism used for the provided namespace.

18 **PMIX\_DISPLAY\_MAP** "pmix.dispmapper" (**bool**)  
19 Display process mapping upon spawn.

20 **PMIX\_PPR** "pmix.ppr" (**char\***)  
21 Number of processes to spawn on each identified resource.

22 **PMIX\_MAPBY** "pmix.mapby" (**char\***)  
23 Process mapping policy - when accessed using **PMIx\_Get**, use the  
24 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
25 provided namespace. Supported values are launcher specific.

26 **PMIX\_RANKBY** "pmix.rankby" (**char\***)  
27 Process ranking policy - when accessed using **PMIx\_Get**, use the  
28 **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
29 provided namespace. Supported values are launcher specific.

30 **PMIX\_BINDTO** "pmix.bindto" (**char\***)  
31 Process binding policy - when accessed using **PMIx\_Get**, use the  
32 **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
33 provided namespace. Supported values are launcher specific.

34 **PMIX\_NON\_PMI** "pmix.nonpmi" (**bool**)

```

1      Spawns processes will not call PMIx_Init.
2      PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
3          Spawns process rank that is to receive any forwarded stdin.
4      PMIX_FWD_STDIN "pmix.fwd.stdin" (pmix_rank_t)
5          The requester intends to push information from its stdin to the indicated process. The
6          local spawn agent should, therefore, ensure that the stdin channel to that process remains
7          available. A rank of PMIX_RANK_WILDCARD indicates that all processes in the spawned
8          job are potential recipients. The requester will issue a call to PMIx_IOF_push to initiate
9          the actual forwarding of information to specified targets - this attribute simply requests that
10         the Intermediate Launcher (IL) retain the ability to forward the information to the designated
11         targets.
12      PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
13          Requests that the ability to forward the stdout of the spawned processes be maintained.
14          The requester will issue a call to PMIx_IOF_pull to specify the callback function and
15          other options for delivery of the forwarded output.
16      PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
17          Requests that the ability to forward the stderr of the spawned processes be maintained.
18          The requester will issue a call to PMIx_IOF_pull to specify the callback function and
19          other options for delivery of the forwarded output.
20      PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)
21          Included in the pmix_info_t array of a pmix_app_t, this attribute declares that the
22          application consists of debugger daemons and shall be governed accordingly. If used as the
23          sole pmix_app_t in a PMIx_Spawn request, then the PMIX_DEBUG_TARGET attribute
24          must also be provided (in either the job_info or in the info array of the pmix_app_t) to
25          identify the namespace to be debugged so that the launcher can determine where to place the
26          spawned daemons. If neither PMIX_DEBUG_DAEMONS_PER_PROC nor
27          PMIX_DEBUG_DAEMONS_PER_NODE is specified, then the launcher shall default to a
28          placement policy of one daemon per process in the target job.
29      PMIX_TAG_OUTPUT "pmix.tagout" (bool)
30          Tag stdout/stderr with the identity of the source process - can be assigned to the entire
31          job (by including attribute in the job_info array) or on a per-application basis in the info
32          array for each pmix_app_t.
33      PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
34          Timestamp output - can be assigned to the entire job (by including attribute in the job_info
35          array) or on a per-application basis in the info array for each pmix_app_t.
36      PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
37          Merge stdout and stderr streams - can be assigned to the entire job (by including
38          attribute in the job_info array) or on a per-application basis in the info array for each
39          pmix_app_t.

```

```

1   PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
2     Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be
3     assigned to the entire job (by including attribute in the job_info array) or on a per-application
4     basis in the info array for each pmix_app_t.
5   PMIX_INDEX_ARGV "pmix.indxargv" (bool)
6     Mark the argv with the rank of the process.
7   PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
8     Number of PUs to assign to each rank - when accessed using PMIx_Get, use the
9     PMIX_RANK_WILDCARD value for the rank to discover the PUs/process assigned to the
10    provided namespace.
11  PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
12    Do not place processes on the head node.
13  PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
14    Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
15    node.
16  PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
17    Report bindings of the individual processes.
18  PMIX_CPU_LIST "pmix.cpulist" (char*)
19    List of PUs to use for this job - when accessed using PMIx_Get, use the
20    PMIX_RANK_WILDCARD value for the rank to discover the PU list used for the provided
21    namespace.
22  PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
23    Application supports recoverable operations.
24  PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
25    Application is continuous, all failed processes should be immediately restarted.
26  PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
27    Maximum number of times to restart a process - when accessed using PMIx_Get, use the
28    PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
29    namespace.
30  PMIX_TIMEOUT "pmix.timeout" (int)
31    Time in seconds before the specified operation should time out (zero indicating infinite) and
32    return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions
33    caused by multiple layers (client, server, and host) simultaneously timing the operation.

```



1           **Description**

2       Spawn a set of applications/processes as per the [PMIx\\_Spawn](#) API. Note that applications are not  
3       required to be MPI or any other programming model. Thus, the host server cannot make any  
4       assumptions as to their required support. The callback function is to be executed once all processes  
5       have been started. An error in starting any application or process in this request shall cause all  
6       applications and processes in the request to be terminated, and an error returned to the originating  
7       caller.

8       Note that a timeout can be specified in the job\_info array to indicate that failure to start the  
9       requested job within the given time should result in termination to avoid hangs.

10      **16.3.11.1 Server spawn attributes**

11      **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)  
12       The requesting process is a PMIx tool.  
13      **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)  
14       The requesting process is a PMIx client.

15      **16.3.12 pmix\_server\_connect\_fn\_t**

16           **Summary**

17       Record the specified processes as *connected*.

18           **Format**

PMIx v1.0

19      **typedef pmix\_status\_t (\*pmix\_server\_connect\_fn\_t)(**  
20            **const pmix\_proc\_t procs[],**  
21            **size\_t nprocs,**  
22            **const pmix\_info\_t info[],**  
23            **size\_t ninfo,**  
24            **pmix\_op\_cbfunc\_t cbfunc,**  
25            **void \*cbdata);**

C

C

26      **IN procs**

27       Array of [pmix\\_proc\\_t](#) structures identifying participants (array of handles)

28      **IN nprocs**

29       Number of elements in the *procs* array (integer)

30      **IN info**

31       Array of info structures (array of handles)

32      **IN ninfo**

33       Number of elements in the *info* array (integer)

34      **IN cbfunc**

35       Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

1   **IN cbdata**

2         Data to be passed to the callback function (memory reference)

3         Returns one of the following:

- 4             • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
5                 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
6                 prior to returning from the API.
- 7             • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
8                 returned *success* - the *cbfunc* will not be called
- 9             • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
10                 request, even though the function entry was provided in the server module - the *cbfunc* will not  
11                 be called
- 12             • a PMIx error constant indicating either an error in the input or that the request was immediately  
13                 processed and failed - the *cbfunc* will not be called

14              **Required Attributes** 

15         PMIx libraries are required to pass any provided attributes to the host environment for processing.  


16              **Optional Attributes** 

17         The following attributes are optional for host environments that support this operation:

18             **PMIX\_TIMEOUT "pmix.timeout" (int)**

19                 Time in seconds before the specified operation should time out (zero indicating infinite) and  
               return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
               caused by multiple layers (client, server, and host) simultaneously timing the operation.  


1           **Description**

2       Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The  
3       callback is to be executed once every daemon hosting at least one participant has called the host  
4       server's **pmix\_server\_connect\_fn\_t** function, and the host environment has completed any  
5       supporting operations required to meet the terms of the PMIx definition of *connected* processes.

6            **Advice to PMIx library implementers** 

7       The PMIx server library is required to aggregate participation by local clients, passing the request  
to the host environment once all local participants have executed the API.  


8            **Advice to PMIx server hosts** 

9       The host will receive a single call for each collective operation. It is the responsibility of the host to  
10      identify the nodes containing participating processes, execute the collective across all participating  
nodes, and notify the local PMIx server library upon completion of the global collective.  


11     **16.3.13 pmix\_server\_disconnect\_fn\_t**

12           **Summary**

13       Disconnect a previously connected set of processes.

## 1 Format

PMIx v1.0

C

```

2     typedef pmix_status_t (*pmix_server_disconnect_fn_t) (
3         const pmix_proc_t procs[],
4         size_t nprocs,
5         const pmix_info_t info[],
6         size_t ninfo,
7         pmix_op_cbfunc_t cbfunc,
8         void *cbdata);

```

C

## 9 IN procs

10 Array of **pmix\_proc\_t** structures identifying participants (array of handles)

## 11 IN nprocs

12 Number of elements in the *procs* array (integer)

## 13 IN info

14 Array of info structures (array of handles)

## 15 IN ninfo

16 Number of elements in the *info* array (integer)

## 17 IN cbfunc

18 Callback function **pmix\_op\_cbfunc\_t** (function reference)

## 19 IN cbdata

20 Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24 prior to returning from the API.
- 25 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26 returned *success* - the *cbfunc* will not be called
- 27 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28 request, even though the function entry was provided in the server module - the *cbfunc* will not  
29 be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the *cbfunc* will not be called


**Required Attributes**

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2   **PMIX\_TIMEOUT "pmix.timeout" (int)**

3   Time in seconds before the specified operation should time out (zero indicating infinite) and  
4   return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
5   caused by multiple layers (client, server, and host) simultaneously timing the operation.

## 6 Description

7 Disconnect a previously connected set of processes. The callback is to be executed once every  
8 daemon hosting at least one participant has called the host server's has called the  
9 **pmix\_server\_disconnect\_fn\_t** function, and the host environment has completed any  
10 required supporting operations.

## Advice to PMIx library implementers

11 The PMIx server library is required to aggregate participation by local clients, passing the request  
12 to the host environment once all local participants have executed the API.

## Advice to PMIx server hosts

13 The host will receive a single call for each collective operation. It is the responsibility of the host to  
14 identify the nodes containing participating processes, execute the collective across all participating  
15 nodes, and notify the local PMIx server library upon completion of the global collective.

16 A **PMIX\_ERR\_INVALID\_OPERATION** error must be returned if the specified set of *procs* was  
17 not previously *connected* via a call to the **pmix\_server\_connect\_fn\_t** function.

## 16.3.14 **pmix\_server\_register\_events\_fn\_t**

### Summary

Register to receive notifications for the specified events.

## 1 Format

PMIx v1.0

```

2     typedef pmix_status_t (*pmix_server_register_events_fn_t)(
3         pmix_status_t *codes,
4         size_t ncodes,
5         const pmix_info_t info[],
6         size_t ninfo,
7         pmix_op_cbfunc_t cbfunc,
8         void *cbdata);
```

C

9 **IN codes**10 Array of `pmix_status_t` values (array of handles)11 **IN ncodes**12 Number of elements in the *codes* array (integer)13 **IN info**

14 Array of info structures (array of handles)

15 **IN ninfo**16 Number of elements in the *info* array (integer)17 **IN cbfunc**18 Callback function `pmix_op_cbfunc_t` (function reference)19 **IN cbdata**

20 Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24 prior to returning from the API.
- 25 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26 returned *success* - the *cbfunc* will not be called
- 27 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28 request, even though the function entry was provided in the server module - the *cbfunc* will not  
29 be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the *cbfunc* will not be called

**Required Attributes**

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
 33 In addition, the following attributes are required to be included in the passed *info* array:

34 **PMIX\_USERID "pmix.euid" (uint32\_t)**

35 Effective user ID of the connecting process.

1           **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2                 Effective group ID of the connecting process.

3           **Description**

4           Register to receive notifications for the specified status codes. The *info* array included in this API is  
5           reserved for possible future directives to further steer notification.

6           **Advice to PMIx library implementers**

7           The PMIx server library must track all client registrations for subsequent notification. This module  
8           function shall only be called when:

- 9           • the client has requested notification of an environmental code (i.e., a PMIx codes in the range  
10              between **PMIX\_EVENT\_SYS\_BASE** and **PMIX\_EVENT\_SYS\_OTHER**, inclusive) or codes that  
11              lies outside the defined PMIx range of constants; and  
12           • the PMIx server library has not previously requested notification of that code - i.e., the host  
13              environment is to be contacted only once a given unique code value

14           **Advice to PMIx server hosts**

15           The host environment is required to pass to its PMIx server library all non-environmental events  
16           that directly relate to a registered namespace without the PMIx server library explicitly requesting  
17           them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in  
18           the range between **PMIX\_EVENT\_SYS\_BASE** and **PMIX\_EVENT\_SYS\_OTHER** (inclusive).

17           **16.3.15 pmix\_server\_deregister\_events\_fn\_t**

18           **Summary**

19           Deregister to receive notifications for the specified events.

1                   **Format**

2        *PMIx v1.0*      C

```
3        typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(
4                                  pmix_status_t *codes,
5                                  size_t ncodes,
6                                  pmix_op_cfunc_t cfunc,
7                                  void *cbdata);
```

8                   **IN codes**  
9                   Array of `pmix_status_t` values (array of handles)

10                  **IN ncodes**  
11                  Number of elements in the *codes* array (integer)

12                  **IN cfunc**  
13                  Callback function `pmix_op_cfunc_t` (function reference)

14                  **IN cbdata**  
15                  Data to be passed to the callback function (memory reference)

16                  Returns one of the following:

- 17                  • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
18                  will be returned in the provided *cfunc*. Note that the host must not invoke the callback function  
prior to returning from the API.
- 19                  • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
20                  returned *success* - the *cfunc* will not be called
- 21                  • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
22                  request, even though the function entry was provided in the server module - the *cfunc* will not  
23                  be called
- 24                  • a PMIx error constant indicating either an error in the input or that the request was immediately  
25                  processed and failed - the *cfunc* will not be called

26                  **Description**  
27                  Deregister to receive notifications for the specified events to which the PMIx server has previously  
28                  registered.

29                   **Advice to PMIx library implementers**

30                  The PMIx server library must track all client registrations. This module function shall only be  
31                  called when:

- 32                   the library is deregistering environmental codes (i.e., a PMIx codes in the range between  
33                   **PMIX\_EVENT\_SYS\_BASE** and **PMIX\_EVENT\_SYS\_OTHER**, inclusive) or codes that lies  
                 outside the defined PMIx range of constants; and

- 1      • no client (including the server library itself) remains registered for notifications on any included  
2      code - i.e., a code should be included in this call only when no registered notifications against it  
3      remain.
- 

4    **16.3.16 pmix\_server\_notify\_event\_fn\_t**

5    **Summary**

6    Notify the specified processes of an event.

7    **Format**

PMIx v2.0

C

```
8    typedef pmix_status_t (*pmix_server_notify_event_fn_t)(
9                          pmix_status_t code,
10                         const pmix_proc_t *source,
11                         pmix_data_range_t range,
12                         pmix_info_t info[],
13                         size_t ninfo,
14                         pmix_op_cbfunc_t cbfunc,
15                         void *cbdata);
```

C

16    **IN code**

17       The **pmix\_status\_t** event code being referenced structure (handle)

18    **IN source**

19       **pmix\_proc\_t** of process that generated the event (handle)

20    **IN range**

21       **pmix\_data\_range\_t** range over which the event is to be distributed (handle)

22    **IN info**

23       Optional array of **pmix\_info\_t** structures containing additional information on the event  
24       (array of handles)

25    **IN ninfo**

26       Number of elements in the *info* array (integer)

27    **IN cbfunc**

28       Callback function **pmix\_op\_cbfunc\_t** (function reference)

29    **IN cbdata**

30       Data to be passed to the callback function (memory reference)

31    Returns one of the following:

- 32    • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
33       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
34       prior to returning from the API.

- 1     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
2        returned *success* - the *cbfunc* will not be called  
  
3     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
4        request, even though the function entry was provided in the server module - the *cbfunc* will not  
5        be called  
  
6     • a PMIx error constant indicating either an error in the input or that the request was immediately  
7        processed and failed - the *cbfunc* will not be called

### Required Attributes

8     PMIx libraries are required to pass any provided attributes to the host environment for processing.

9     Host environments that provide this module entry point are required to support the following  
10      attributes:

#### **PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t)**

12       Define constraints on the processes that can access the provided data. Only processes that  
13        meet the constraints are allowed to access it.

### Description

15     Notify the specified processes (described through a combination of *range* and attributes provided in  
16        the *info* array) of an event generated either by the PMIx server itself or by one of its local clients.  
17     The process generating the event is provided in the *source* parameter, and any further descriptive  
18        information is included in the *info* array.

19     Note that the PMIx server library is not allowed to echo any event given to it by its host via the  
20        **PMIx\_Notify\_event** API back to the host through the  
21        **pmix\_server\_notify\_event\_fn\_t** server module function.

### Advice to PMIx server hosts

22     The callback function is to be executed once the host environment no longer requires that the PMIx  
23        server library maintain the provided data structures. It does not necessarily indicate that the event  
24        has been delivered to any process, nor that the event has been distributed for delivery

## 25    16.3.17 **pmix\_server\_listener\_fn\_t**

### 26    Summary

27     Register a socket the host server can monitor for connection requests.

1                   **Format**

2        *PMIx v1.0*      C

```
3        typedef pmix_status_t (*pmix_server_listener_fn_t) (
4                    int listening_sd,
5                    pmix_connection_cbfunc_t cbfunc,
6                    void *cbdata);
```

6        **IN incoming\_sd**  
7                    (integer)  
8        **IN cbfunc**  
9                    Callback function **pmix\_connection\_cbfunc\_t** (function reference)  
10      **IN cbdata**  
11                    (memory reference)

12      Returns **PMIX\_SUCCESS** indicating that the request is accepted, or a negative value corresponding  
13      to a PMIx error constant indicating that the request has been rejected.

14                   **Description**

15      Register a socket the host environment can monitor for connection requests, harvest them, and then  
16      call the PMIx server library's internal callback function for further processing. A listener thread is  
17      essential to efficiently harvesting connection requests from large numbers of local clients such as  
18      occur when running on large SMPs. The host server listener is required to call accept on the  
19      incoming connection request, and then pass the resulting socket to the provided cbfunc. A **NULL**  
20      for this function will cause the internal PMIx server to spawn its own listener thread.

21     **16.3.17.1 PMIx Client Connection Callback Function**

22                   **Summary**

23      Callback function for incoming connection request from a local client.

24                   **Format**

25        *PMIx v1.0*      C

```
26        typedef void (*pmix_connection_cbfunc_t) (
27                    int incoming_sd, void *cbdata);
```

27        **IN incoming\_sd**  
28                    (integer)  
29        **IN cbdata**  
30                    (memory reference)

31                   **Description**

32      Callback function for incoming connection requests from local clients - only used by host  
33      environments that wish to directly handle socket connection requests.

## 16.3.18 pmix\_server\_query\_fn\_t

### 2 Summary

3 Query information from the resource manager.

### 4 Format

5 PMIx v2.0

C

```
6     typedef pmix_status_t (*pmix_server_query_fn_t) (
7             pmix_proc_t *proct,
8             pmix_query_t *queries,
9             size_t nqueries,
10            pmix_info_cfunc_t cbfunc,
11            void *cbdata);
```

C

11 IN proct

12 pmix\_proc\_t structure of the requesting process (handle)

13 IN queries

14 Array of pmix\_query\_t structures (array of handles)

15 IN nqueries

16 Number of elements in the queries array (integer)

17 IN cbfunc

18 Callback function pmix\_info\_cfunc\_t (function reference)

19 IN cbdata

20 Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22 • PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result  
23 will be returned in the provided cbfunc. Note that the host must not invoke the callback function  
24 prior to returning from the API.
- 25 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and  
26 returned success - the cbfunc will not be called
- 27 • PMIX\_ERR\_NOT\_SUPPORTED, indicating that the host environment does not support the  
28 request, even though the function entry was provided in the server module - the cbfunc will not  
29 be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the cbfunc will not be called

## Required Attributes

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
2 In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
4     Effective user ID of the connecting process.  
5 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
6     Effective group ID of the connecting process.

## Optional Attributes

7 The following attributes are optional for host environments that support this operation:

8 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (**char\***)  
9     Request a comma-delimited list of active namespaces. NO QUALIFIERS.  
10 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (**pmix\_status\_t**)  
11     Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX\_NSPACE**  
12       indicating the namespace whose status is being queried.  
13 **PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (**char\***)  
14     Request a comma-delimited list of scheduler queues. NO QUALIFIERS.  
15 **PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (**char\***)  
16     Returns status of a specified scheduler queue, expressed as a string. OPTIONAL  
17       QUALIFIERS: **PMIX\_ALLOC\_QUEUE** naming specific queue whose status is being  
18       requested.  
19 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.ptable" (**char\***)  
20     Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
21       process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
22       **PMIX\_NSPACE** indicating the namespace whose process table is being queried.  
23 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lptable" (**char\***)  
24     Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
25       process in the specified namespace executing on the same node as the requester, ordered by  
26       process job rank. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the namespace  
27       whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME**  
28       indicating the host whose local process table is being queried. By default, the query assumes  
29       that the host upon which the request was made is to be used.  
30 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (**bool**)  
31     Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.  
32 **PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (**bool**)  
33     Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

```
1 PMIX_QUERY_MEMORY_USAGE "pmixqry.mem" (bool)
2     Return information on memory usage for the processes indicated in the qualifiers.
3     OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
4     specific process(es) whose memory usage is being requested.
5 PMIX_QUERY_LOCAL_ONLY "pmixqry.local" (bool)
6     Constrain the query to local information only. NO QUALIFIERS.
7 PMIX_QUERY_REPORT_AVG "pmixqry.avg" (bool)
8     Report only average values for sampled information. NO QUALIFIERS.
9 PMIX_QUERY_REPORT_MINMAX "pmixqry.minmax" (bool)
10    Report minimum and maximum values. NO QUALIFIERS.
11 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
12    String identifier of the allocation whose status is being requested. NO QUALIFIERS.
13 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
14    Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
15    OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
16    requested (defaults to allocation containing the caller).
```



## 17 **Description**

18 Query information from the host environment. The query will include the namespace/rank of the  
19 process that is requesting the info, an array of **pmix\_query\_t** describing the request, and a  
20 callback function/data for the return.

### Advice to PMIx library implementers

21 The PMIx server library should not block in this function as the host environment may, depending  
22 upon the information being requested, require significant time to respond.



## 23 **16.3.19 pmix\_server\_tool\_connection\_fn\_t**

### 24 **Summary**

25 Register that a tool has connected to the server.

1      **Format**

2      *PMIx v2.0*

C

```
2      typedef void (*pmix_server_tool_connection_fn_t)(
3                          pmix_info_t info[], size_t ninfo,
4                          pmix_tool_connection_cfunc_t cfunc,
5                          void *cbdata);
```

C

6      **IN info**

7      Array of `pmix_info_t` structures (array of handles)

8      **IN ninfo**

9      Number of elements in the *info* array (integer)

10     **IN cfunc**

11     Callback function `pmix_tool_connection_cfunc_t` (function reference)

12     **IN cbdata**

13     Data to be passed to the callback function (memory reference)

▼----- Required Attributes -----▼

14     PMIx libraries are required to pass the following attributes in the *info* array:

15     **PMIX\_USERID "pmix.euid" (uint32\_t)**

16       Effective user ID of the connecting process.

17     **PMIX\_GRP\_ID "pmix.egid" (uint32\_t)**

18       Effective group ID of the connecting process.

19     **PMIX\_TOOL\_NSPACE "pmix.tool.nspace" (char\*)**

20       Name of the namespace to use for this tool. This must be included only if the tool already  
21       has an assigned namespace.

22     **PMIX\_TOOL\_RANK "pmix.tool.rank" (uint32\_t)**

23       Rank of this tool. This must be included only if the tool already has an assigned rank.

24     **PMIX\_CREDENTIAL "pmix.cred" (char\*)**

25       Security credential assigned to the process.



## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2   **PMIX\_FWD\_STDOUT "pmix.fwd.stdout" (bool)**

3   Requests that the ability to forward the **stdout** of the spawned processes be maintained.  
4   The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
5   other options for delivery of the forwarded output.

6   **PMIX\_FWD\_STDERR "pmix.fwd.stderr" (bool)**

7   Requests that the ability to forward the **stderr** of the spawned processes be maintained.  
8   The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
9   other options for delivery of the forwarded output.

10   **PMIX\_FWD\_STDIN "pmix.fwd.stdin" (pmix\_rank\_t)**

11   The requester intends to push information from its **stdin** to the indicated process. The  
12   local spawn agent should, therefore, ensure that the **stdin** channel to that process remains  
13   available. A rank of **PMIX\_RANK\_WILDCARD** indicates that all processes in the spawned  
14   job are potential recipients. The requester will issue a call to **PMIx\_IOF\_push** to initiate  
15   the actual forwarding of information to specified targets - this attribute simply requests that  
16   the IL retain the ability to forward the information to the designated targets.

17   **PMIX\_VERSION\_INFO "pmix.version" (char\*)**

18   PMIx version of the library being used by the connecting process.

## 19   Description

20   Register that a tool has connected to the server, possibly requesting that the tool be assigned a  
21   namespace/rank identifier for further interactions. The **pmix\_info\_t** array is used to pass  
22   qualifiers for the connection request, including the effective uid and gid of the calling tool for  
23   authentication purposes.

24   If the tool already has an assigned process identifier, then this must be indicated in the *info* array.  
25   The host is responsible for checking that the provided namespace does not conflict with any  
26   currently known assignments, returning an appropriate error in the callback function if a conflict is  
27   found.

28   The host environment is solely responsible for authenticating and authorizing the connection using  
29   whatever means it deems appropriate. If certificates or other authentication information are  
30   required, then the tool must provide them. The conclusion of those operations shall be  
31   communicated back to the PMIx server library via the callback function.

32   Approval or rejection of the connection request shall be returned in the *status* parameter of the  
33   **pmix\_tool\_connection\_cbfunc\_t**. If the connection is refused, the PMIx server library  
34   must terminate the connection attempt. The host must not execute the callback function prior to  
35   returning from the API.

### 16.3.19.1 Tool connection attributes

Attributes associated with tool connections.

```
PMIX_USERID "pmix.euid" (uint32_t)  
    Effective user ID of the connecting process.  
PMIX_GRPID "pmix.egid" (uint32_t)  
    Effective group ID of the connecting process.  
PMIX_VERSION_INFO "pmix.version" (char*)  
    PMIx version of the library being used by the connecting process.
```

### 16.3.19.2 PMIx Tool Connection Callback Function

#### Summary

Callback function for incoming tool connections.

#### Format

PMIx v2.0

```
typedef void (*pmix_tool_connection_cbfunc_t) (  
    pmix_status_t status,  
    pmix_proc_t *proc, void *cbdata);
```

IN **status**  
 pmix\_status\_t value (handle)  
IN **proc**  
 pmix\_proc\_t structure containing the identifier assigned to the tool (handle)  
IN **cbdata**  
 Data to be passed (memory reference)

#### Description

Callback function for incoming tool connections. The host environment shall provide a namespace/rank identifier for the connecting tool.

#### Advice to PMIx server hosts

It is assumed that **rank=0** will be the normal assignment, but allow for the future possibility of a parallel set of tools connecting, and thus each process requiring a unique rank.

### 16.3.20 pmix\_server\_log\_fn\_t

#### Summary

Log data on behalf of a client.

1      **Format**

2      **PMIx v2.0**      C  
3      **typedef void (\*pmix\_server\_log\_fn\_t)(**  
4                          **const pmix\_proc\_t \*client,**  
5                          **const pmix\_info\_t data[], size\_t ndata,**  
6                          **const pmix\_info\_t directives[], size\_t ndirs,**  
7                          **pmix\_op\_cbfunc\_t cbfunc, void \*cbdata);**

8      **IN client**  
9                  **pmix\_proc\_t** structure (handle)  
10     **IN data**  
11        Array of info structures (array of handles)  
12     **IN ndata**  
13        Number of elements in the *data* array (integer)  
14     **IN directives**  
15        Array of info structures (array of handles)  
16     **IN ndirs**  
17        Number of elements in the *directives* array (integer)  
18     **IN cbfunc**  
19        Callback function **pmix\_op\_cbfunc\_t** (function reference)  
20     **IN cbdata**  
21        Data to be passed to the callback function (memory reference)

22      **Required Attributes**

23      PMIx libraries are required to pass any provided attributes to the host environment for processing.  
24      In addition, the following attributes are required to be included in the passed *info* array:

25     **PMIX\_USERID "pmix.euid" (uint32\_t)**  
26        Effective user ID of the connecting process.  
27     **PMIX\_GRPID "pmix.egid" (uint32\_t)**  
28        Effective group ID of the connecting process.

29      Host environments that provide this module entry point are required to support the following  
30      attributes:

31     **PMIX\_LOG\_STDERR "pmix.log.stderr" (char\*)**  
32        Log string to **stderr**.  
33     **PMIX\_LOG\_STDOUT "pmix.log.stdout" (char\*)**  
34        Log string to **stdout**.  
35     **PMIX\_LOG\_SYSLOG "pmix.log.syslog" (char\*)**

1 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
2 otherwise to local syslog.

### Optional Attributes

3 The following attributes are optional for host environments that support this operation:

4 **PMIX\_LOG\_MSG** "pmix.log.msg" (**pmix\_byte\_object\_t**)

5 Message blob to be sent somewhere.

6 **PMIX\_LOG\_EMAIL** "pmix.log.email" (**pmix\_data\_array\_t**)

7 Log via email based on **pmix\_info\_t** containing directives.

8 **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (**char\***)

9 Comma-delimited list of email addresses that are to receive the message.

10 **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (**char\***)

11 Subject line for email.

12 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (**char\***)

13 Message to be included in email.

### Description

15 Log data on behalf of a client. This function is not intended for output of computational results, but  
16 rather for reporting status and error messages. The host must not execute the callback function prior  
17 to returning from the API.

## 16.3.21 **pmix\_server\_alloc\_fn\_t**

### Summary

Request allocation operations on behalf of a client.

1      **Format**

C

```
2     typedef pmix_status_t (*pmix_server_alloc_fn_t)(
3         const pmix_proc_t *client,
4         pmix_alloc_directive_t directive,
5         const pmix_info_t data[],
6         size_t ndata,
7         pmix_info_cbfunc_t cbfunc,
8         void *cbdata);
```

C

9      **IN client**

10     **pmix\_proc\_t** structure of process making request (handle)

11     **IN directive**

12     Specific action being requested (**pmix\_alloc\_directive\_t**)

13     **IN data**

14     Array of info structures (array of handles)

15     **IN ndata**

16     Number of elements in the *data* array (integer)

17     **IN cbfunc**

18     Callback function **pmix\_info\_cbfunc\_t** (function reference)

19     **IN cbdata**

20     Data to be passed to the callback function (memory reference)

21     Returns one of the following:

- 22     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24       prior to returning from the API.
- 25     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26       returned *success* - the *cbfunc* will not be called
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28       request, even though the function entry was provided in the server module - the *cbfunc* will not  
29       be called
- 30     • a PMIx error constant indicating either an error in the input or that the request was immediately  
31       processed and failed - the *cbfunc* will not be called

▼----- Required Attributes -----▼

32     PMIx libraries are required to pass any provided attributes to the host environment for processing.  
33     In addition, the following attributes are required to be included in the passed *info* array:

34     **PMIX\_USERID "pmix.euid" (uint32\_t)**

35       Effective user ID of the connecting process.

```
1 PMIX_GRP_ID "pmix.egid" (uint32_t)
2 Effective group ID of the connecting process.
3
4 Host environments that provide this module entry point are required to support the following
5 attributes:
6 PMIX_ALLOC_ID "pmix.alloc.id" (char*)
7 A string identifier (provided by the host environment) for the resulting allocation which can
8 later be used to reference the allocated resources in, for example, a call to PMIX_Spawn.
9 PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
10 The number of nodes being requested in an allocation request.
11 PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
12 Number of PUs being requested in an allocation request.
13 PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
14 Total session time (in seconds) being requested in an allocation request.
```

## Optional Attributes

```
15 The following attributes are optional for host environments that support this operation:
16 PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
17 Regular expression of the specific nodes being requested in an allocation request.
18 PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
19 Regular expression of the number of PUs for each node being requested in an allocation
20 request.
21 PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
22 Regular expression of the specific PUs being requested in an allocation request.
23 PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
24 Number of Megabytes[base2] of memory (per process) being requested in an allocation
25 request.
26 PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
27 Array of pmix_info_t describing requested fabric resources. This must include at least:
28 PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
29 PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
30 PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
```

1       The key to be used when accessing this requested fabric allocation. The fabric allocation  
2       will be returned/stored as a `pmix_data_array_t` of `pmix_info_t` whose first  
3       element is composed of this key and the allocated resource description. The type of the  
4       included value depends upon the fabric support. For example, a TCP allocation might  
5       consist of a comma-delimited string of socket ranges such as "**32000-32100,**  
6       **33005, 38123-38146**". Additional array entries will consist of any provided resource  
7       request directives, along with their assigned values. Examples include:  
8       **PMIX\_ALLOC\_FABRIC\_TYPE** - the type of resources provided;  
9       **PMIX\_ALLOC\_FABRIC\_PLANE** - if applicable, what plane the resources were assigned  
10      from; **PMIX\_ALLOC\_FABRIC\_QOS** - the assigned QoS; **PMIX\_ALLOC\_BANDWIDTH** -  
11      the allocated bandwidth; **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** - a security key for the  
12      requested fabric allocation. NOTE: the array contents may differ from those requested,  
13      especially if **PMIX\_INFO\_REQD** was not set in the request.

14      **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (**float**)  
15      Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
16      request.  
17      **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (**char\***)  
18      Fabric quality of service level for the job being requested in an allocation request.



## 19     **Description**

20     Request new allocation or modifications to an existing allocation on behalf of a client. Several  
21     broad categories are envisioned, including the ability to:

- 22     • Request allocation of additional resources, including memory, bandwidth, and compute for an  
23       existing allocation. Any additional allocated resources will be considered as part of the current  
24       allocation, and thus will be released at the same time.  
25     • Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not  
26       affiliated with) the allocation of the requestor - thus the termination of one allocation will not  
27       impact the other.  
28     • Extend the reservation on currently allocated resources, subject to scheduling availability and  
29       priorities.  
30     • Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to  
31       the scheduler with a promise to return them upon subsequent request.

32     The callback function provides a *status* to indicate whether or not the request was granted, and to  
33     provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of  
34     `pmix_info_t` structures.

### 35    **16.3.22 pmix\_server\_job\_control\_fn\_t**

#### 36    **Summary**

37    Execute a job control action on behalf of a client.

1      **Format**

PMIx v2.0

C

```
2      typedef pmix_status_t (*pmix_server_job_control_fn_t)(
3               const pmix_proc_t *requestor,
4               const pmix_proc_t targets[],
5               size_t ntargs,
6               const pmix_info_t directives[],
7               size_t ndirs,
8               pmix_info_cbfunc_t cbfunc,
9               void *cbdata);
```

C

```
10     IN requestor  
11       pmix_proc_t structure of requesting process (handle)  
12     IN targets  
13       Array of proc structures (array of handles)  
14     IN ntargs  
15       Number of elements in the targets array (integer)  
16     IN directives  
17       Array of info structures (array of handles)  
18     IN ndirs  
19       Number of elements in the info array (integer)  
20     IN cbfunc  
21       Callback function pmix_op_cbfunc_t (function reference)  
22     IN cbdata  
23       Data to be passed to the callback function (memory reference)
```

24     Returns one of the following:

- 25     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
26       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
27       prior to returning from the API.
- 28     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
29       returned *success* - the *cbfunc* will not be called
- 30     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
31       request, even though the function entry was provided in the server module - the *cbfunc* will not  
32       be called
- 33     • a PMIx error constant indicating either an error in the input or that the request was immediately  
34       processed and failed - the *cbfunc* will not be called

## Required Attributes

1 PMIx libraries are required to pass any attributes provided by the client to the host environment for  
2 processing. In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
4       Effective user ID of the connecting process.  
5 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
6       Effective group ID of the connecting process.

---

7  
8 Host environments that provide this module entry point are required to support the following  
9 attributes:

10 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (**char\***)  
11      Provide a string identifier for this request. The user can provide an identifier for the  
12      requested operation, thus allowing them to later request status of the operation or to  
13      terminate it. The host, therefore, shall track it with the request for future reference.  
14 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (**bool**)  
15      Pause the specified processes.  
16 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (**bool**)  
17      Resume ("un-pause") the specified processes.  
18 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (**bool**)  
19      Forcibly terminate the specified processes and cleanup.  
20 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (**int**)  
21      Send given signal to specified processes.  
22 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (**bool**)  
23      Politely terminate the specified processes.

## Optional Attributes

24 The following attributes are optional for host environments that support this operation:

25 **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (**char\***)  
26      Cancel the specified request - the provided request ID must match the  
27      **PMIX\_JOB\_CTRL\_ID** provided to a previous call to **PMIx\_Job\_control**. An ID of  
28      **NULL** implies cancel all requests from this requestor.  
29 **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (**char\***)  
30      Restart the specified processes using the given checkpoint ID.  
31 **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (**char\***)  
32      Checkpoint the specified processes and assign the given ID to it.

```
1   PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
2     Use event notification to trigger a process checkpoint.
3   PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
4     Use the given signal to trigger a process checkpoint.
5   PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
6     Time in seconds to wait for a checkpoint to complete.
7   PMIX_JOB_CTRL_CHECKPOINT_METHOD
8     "pmix.jctrl.ckmethod" (pmix_data_array_t)
9     Array of pmix_info_t declaring each method and value supported by this application.
10  PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
11    Regular expression identifying nodes that are to be provisioned.
12  PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
13    Name of the image that is to be provisioned.
14  PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
15    Indicate that the job can be pre-empted.
```

## 16 Description

17 Execute a job control action on behalf of a client. The *targets* array identifies the processes to  
18 which the requested job control action is to be applied. A **NULL** value can be used to indicate all  
19 processes in the caller's namespace. The use of **PMIX\_RANK\_WILDCARD** can also be used to  
20 indicate that all processes in the given namespace are to be included.

21 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The callback  
22 function provides a *status* to indicate whether or not the request was granted, and to provide some  
23 information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
24 **pmix\_info\_t** structures.

## 25 16.3.23 pmix\_server\_monitor\_fn\_t

### 26 Summary

27 Request that a client be monitored for activity.

## 1 Format

PMIx v2.0

C

```
2     typedef pmix_status_t (*pmix_server_monitor_fn_t) (
3         const pmix_proc_t *requestor,
4         const pmix_info_t *monitor,
5         pmix_status_t error,
6         const pmix_info_t directives[],
7         size_t ndirs,
8         pmix_info_cfunc_t cbfunc,
9         void *cbdata);
```

C

10 **IN requestor**  
11     **pmix\_proc\_t** structure of requesting process (handle)  
12 **IN monitor**  
13     **pmix\_info\_t** identifying the type of monitor being requested (handle)  
14 **IN error**  
15     Status code to use in generating event if alarm triggers (integer)  
16 **IN directives**  
17     Array of info structures (array of handles)  
18 **IN ndirs**  
19     Number of elements in the *info* array (integer)  
20 **IN cbfunc**  
21     Callback function **pmix\_op\_cfunc\_t** (function reference)  
22 **IN cbdata**  
23     Data to be passed to the callback function (memory reference)

24 Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

35 This entry point is only called for monitoring requests that are not directly supported by the PMIx  
36 server library itself.

## Required Attributes

If supported by the PMIx server library, then the library must not pass any supported attributes to the host environment. Any attributes provided by the client that are not directly supported by the server library must be passed to the host environment if it provides this module entry. In addition, the following attributes are required to be included in the passed *info* array:

5     **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
6         Effective user ID of the connecting process.  
  
7     **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
8         Effective group ID of the connecting process.

9     Host environments are not required to support any specific monitoring attributes.

## Optional Attributes

10   The following attributes may be implemented by a host environment.

11   **PMIX\_MONITOR\_ID** "pmix.monitor.id" (**char\***)  
12      Provide a string identifier for this request.  
  
13   **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (**char\***)  
14      Identifier to be canceled (**NULL** means cancel all monitoring for this process).  
  
15   **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (**bool**)  
16      The application desires to control the response to a monitoring event - i.e., the application is  
17      requesting that the host environment not take immediate action in response to the event (e.g.,  
18      terminating the job).  
  
19   **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (**void**)  
20      Register to have the PMIx server monitor the requestor for heartbeats.  
  
21   **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (**uint32\_t**)  
22      Time in seconds before declaring heartbeat missed.  
  
23   **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (**uint32\_t**)  
24      Number of heartbeats that can be missed before generating the event.  
  
25   **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (**char\***)  
26      Register to monitor file for signs of life.  
  
27   **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (**bool**)  
28      Monitor size of given file is growing to determine if the application is running.  
  
29   **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (**char\***)  
30      Monitor time since last access of given file to determine if the application is running.  
  
31   **PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (**char\***)  
32      Monitor time since last modified of given file to determine if the application is running.

```
1 PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
2     Time in seconds between checking the file.
3 PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
4     Number of file checks that can be missed before generating the event.
```

## 5 Description

6 Request that a client be monitored for activity.

### 7 16.3.24 pmix\_server\_get\_cred\_fn\_t

#### 8 Summary

9 Request a credential from the host environment.

#### 10 Format

PMIx v3.0

C

```
11 typedef pmix_status_t (*pmix_server_get_cred_fn_t) (
12     const pmix_proc_t *proc,
13     const pmix_info_t directives[],
14     size_t ndirs,
15     pmix_credential_cbfunc_t cbfunc,
16     void *cbdata);
```

C

- 17 IN proc  
18 `pmix_proc_t` structure of requesting process (handle)
- 19 IN directives  
20 Array of info structures (array of handles)
- 21 IN ndirs  
22 Number of elements in the *info* array (integer)
- 23 IN cbfunc  
24 Callback function to return the credential (`pmix_credential_cbfunc_t` function  
25 reference)
- 26 IN cbdata  
27 Data to be passed to the callback function (memory reference)
- 28 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
29 will be returned in the provided *cbfunc*
- 30 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
31 request, even though the function entry was provided in the server module - the *cbfunc* will not  
32 be called
- 33 • a PMIx error constant indicating either an error in the input or that the request was immediately  
34 processed and failed - the *cbfunc* will not be called

## Required Attributes

If the PMIx library does not itself provide the requested credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

```
1 PMIX_USERID "pmix.euid" (uint32_t)
2   Effective user ID of the connecting process.
3
4 PMIX_GRP_ID "pmix.egid" (uint32_t)
5   Effective group ID of the connecting process.
```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

```
9 PMIX_CRED_TYPE "pmix.sec.ctype" (char*)
10  When passed in PMIx_Get_credential, a prioritized, comma-delimited list of desired
11    credential types for use in environments where multiple authentication mechanisms may be
12    available. When returned in a callback function, a string identifier of the credential type.
13
14 PMIX_TIMEOUT "pmix.timeout" (int)
15  Time in seconds before the specified operation should time out (zero indicating infinite) and
16    return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions
    caused by multiple layers (client, server, and host) simultaneously timing the operation.
```

### 17 **Description**

18 Request a credential from the host environment.

### 19 **16.3.24.1 Credential callback function**

#### 20 **Summary**

21 Callback function to return a requested security credential

1           **Format**

2        `typedef void (*pmix_credential_cbfunc_t) (`  
3                                      `pmix_status_t status,`  
4                                      `pmix_byte_object_t *credential,`  
5                                      `pmix_info_t info[], size_t ninfo,`  
6                                      `void *cbdata);`

7        **IN    status**

8                  `pmix_status_t` value (handle)

9        **IN    credential**

10                 `pmix_byte_object_t` structure containing the security credential (handle)

11       **IN    info**

12                 Array of provided by the system to pass any additional information about the credential - e.g.,  
13                 the identity of the issuing agent. (handle)

14       **IN    ninfo**

15                 Number of elements in `info` (`size_t`)

16       **IN    cbdata**

17                 Object passed in original request (memory reference)

18           **Description**

19         Define a callback function to return a requested security credential. Information provided by the  
20         issuing agent can subsequently be used by the application for a variety of purposes. Examples  
21         include:

- 22
  - checking identified authorizations to determine what requests/operations are feasible as a means  
23                 to steering [workflows](#)
  - compare the credential type to that of the local SMS for compatibility

24           **Advice to users**

25         The credential is opaque and therefore understandable only by a service compatible with the issuer.  
26         The `info` array is owned by the PMIx library and is not to be released or altered by the receiving  
27         party.

28     **16.3.25 pmix\_server\_validate\_cred\_fn\_t**

29           **Summary**

30             Request validation of a credential.

1      **Format**

C

```
2      typedef pmix_status_t (*pmix_server_validate_cred_fn_t)(
3           const pmix_proc_t *proc,
4           const pmix_byte_object_t *cred,
5           const pmix_info_t directives[],
6           size_t ndirs,
7           pmix_validation_cbfunc_t cbfunc,
8           void *cbdata);
```

C

9      **IN proc**

10     **pmix\_proc\_t** structure of requesting process (handle)

11     **IN cred**

12     Pointer to **pmix\_byt<sub>e</sub>\_object\_t** containing the credential (handle)

13     **IN directives**

14     Array of info structures (array of handles)

15     **IN ndirs**

16     Number of elements in the *info* array (integer)

17     **IN cbfunc**

18     Callback function to return the result (**pmix\_validation\_cbfunc\_t** function reference)

19     **IN cbdata**

20     Data to be passed to the callback function (memory reference)

21     Returns one of the following:

- 22     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*
- 24     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
25       returned *success* - the *cbfunc* will not be called
- 26     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
27       request, even though the function entry was provided in the server module - the *cbfunc* will not  
28       be called
- 29     • a PMIx error constant indicating either an error in the input or that the request was immediately  
30       processed and failed - the *cbfunc* will not be called

31     **Required Attributes**

32     If the PMIx library does not itself validate the credential, then it is required to pass any attributes  
33     provided by the client to the host environment for processing. In addition, it must include the  
34     following attributes in the passed *info* array:

35     **PMIX\_USERID "pmix.euid" (uint32\_t)**

            Effective user ID of the connecting process.

1           **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2            Effective group ID of the connecting process.

---

3  
4           Host environments are not required to support any specific attributes.

▲-----▼      **Optional Attributes**      ▼-----▲

5           The following attributes are optional for host environments that support this operation:

6           **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
7           Time in seconds before the specified operation should time out (zero indicating infinite) and  
8           return the **PMIX\_ERR\_TIMEOUT** error. Care should be taken to avoid race conditions  
9           caused by multiple layers (client, server, and host) simultaneously timing the operation.

10          **Description**

11          Request validation of a credential obtained from the host environment via a prior call to the  
12         **pmix\_server\_get\_cred\_fn\_t** module entry.

13     **16.3.26 Credential validation callback function**

14          **Summary**

15          Callback function for security credential validation.

1           **Format**

C

```
2       typedef void (*pmix_validation_cbfunc_t)(
3                           pmix_status_t status,
4                           pmix_info_t info[], size_t ninfo,
5                           void *cbdata);
```

6       **IN status**

7           **pmix\_status\_t** value (handle)

8       **IN info**

9           Array of **pmix\_info\_t** provided by the system to pass any additional information about the  
10          authentication - e.g., the effective userid and group id of the certificate holder, and any related  
11          authorizations (handle)

12       **IN ninfo**

13           Number of elements in *info* (**size\_t**)

14       **IN cbdata**

15           Object passed in original request (memory reference)

16          The returned status shall be one of the following:

- 17       • **PMIX\_SUCCESS**, indicating that the request was processed and returned *success* (i.e., the  
18          credential was both valid and any information it contained was successfully processed). Details  
19          of the result will be returned in the *info* array
- 20       • a PMIx error constant indicating either an error in the parsing of the credential or that the request  
21          was refused

22           **Description**

23          Define a validation callback function to indicate if a provided credential is valid, and any  
24          corresponding information regarding authorizations and other security matters.

25           **Advice to users**

26          The precise contents of the array will depend on the host environment and its associated security  
27          system. At the minimum, it is expected (but not required) that the array will contain entries for the  
28          **PMIX\_USERID** and **PMIX\_GRPID** of the client described in the credential. The *info* array is  
            owned by the PMIx library and is not to be released or altered by the receiving party.

29       **16.3.27 pmix\_server\_iof\_fn\_t**

30           **Summary**

31          Request the specified IO channels be forwarded from the given array of processes.

1      **Format**

PMIx v3.0

C

```
2      typedef pmix_status_t (*pmix_server_iof_fn_t)(
3               const pmix_proc_t procs[],
4               size_t nprocs,
5               const pmix_info_t directives[],
6               size_t ndirs,
7               pmix_iof_channel_t channels,
8               pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

9      **IN procs**

10     Array **pmix\_proc\_t** identifiers whose IO is being requested (handle)

11     **IN nprocs**

12     Number of elements in *procs* (**size\_t**)

13     **IN directives**

14     Array of **pmix\_info\_t** structures further defining the request (array of handles)

15     **IN ndirs**

16     Number of elements in the *info* array (integer)

17     **IN channels**

18     Bitmask identifying the channels to be forwarded (**pmix\_iof\_channel\_t**)

19     **IN cbfunc**

20     Callback function **pmix\_op\_cbfunc\_t** (function reference)

21     **IN cbdata**

22     Data to be passed to the callback function (memory reference)

23     Returns one of the following:

- 24     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
25       will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
26       function prior to returning from the API.
- 27     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
28       returned *success* - the *cbfunc* will not be called
- 29     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
30       request, even though the function entry was provided in the server module - the *cbfunc* will not  
31       be called
- 32     • a PMIx error constant indicating either an error in the input or that the request was immediately  
33       processed and failed - the *cbfunc* will not be called

## Required Attributes

1 The following attributes are required to be included in the passed *info* array:

2 **PMIX\_USERID** "pmix.euid" (`uint32_t`)

3 Effective user ID of the connecting process.

4 **PMIX\_GRP\_ID** "pmix.egid" (`uint32_t`)

5 Effective group ID of the connecting process.

6

---

7 Host environments that provide this module entry point are required to support the following  
8 attributes:

9 **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csizes" (`uint32_t`)

10 The requested size of the PMIx server cache in bytes for each specified channel. By default,  
11 the server is allowed (but not required) to drop all bytes received beyond the max size.

12 **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (`bool`)

13 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
14 cache.

15 **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (`bool`)

16 In an overflow situation, the PMIx server is to drop any new bytes received until room  
17 becomes available in the cache (default).

## Optional Attributes

18 The following attributes may be supported by a host environment.

19 **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (`uint32_t`)

20 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
21 specified number of bytes is collected to avoid being called every time a block of IO arrives.  
22 The PMIx tool library will execute the callback and reset the collection counter whenever the  
23 specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
24 the callback upon a call to deregister the respective channel.

25 **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (`uint32_t`)

26 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
27 size, this prevents IO from being held indefinitely while waiting for another payload to  
28 arrive.

1           **Description**  
2         Request the specified IO channels be forwarded from the given array of processes. An error shall be  
3         returned in the callback function if the requested service from any of the requested processes cannot  
4         be provided.

5            **Advice to PMIx library implementers** 

6         The forwarding of stdin is a *push* process - processes cannot request that it be *pulled* from some  
7         other source. Requests including the **PMIX\_FWD\_STDIN\_CHANNEL** channel will return a  
**PMIX\_ERR\_NOT\_SUPPORTED** error.  


8    **16.3.27.1 IOF delivery function**

9           **Summary**  
10          Callback function for delivering forwarded IO to a process.

11           **Format**

PMIx v3.0

12          **C**   
13         **typedef void (\*pmix\_iof\_cbfunc\_t)(**  
14                 **size\_t iofhdlr, pmix\_iof\_channel\_t channel,**  
15                 **pmix\_proc\_t \*source, char \*payload,**  
               **pmix\_info\_t info[], size\_t ninfo);**

16         **IN iofhdlr**  
17         Registration number of the handler being invoked (**size\_t**)  
18         **IN channel**  
19         bitmask identifying the channel the data arrived on (**pmix\_iof\_channel\_t**)  
20         **IN source**  
21         Pointer to a **pmix\_proc\_t** identifying the namespace/rank of the process that generated the  
22         data (**char\***)  
23         **IN payload**  
24         Pointer to character array containing the data.  
25         **IN info**  
26         Array of **pmix\_info\_t** provided by the source containing metadata about the payload. This  
27         could include **PMIX\_IOF\_COMPLETE** (handle)  
28         **IN ninfo**  
29         Number of elements in *info* (**size\_t**)

1           **Description**  
2       Define a callback function for delivering forwarded IO to a process. This function will be called  
3       whenever data becomes available, or a specified buffering size and/or time has been met.

4           **Advice to users**  
5       Multiple strings may be included in a given *payload*, and the *payload* may *not* be **NULL** terminated.  
6       The user is responsible for releasing the *payload* memory. The *info* array is owned by the PMIx  
library and is not to be released or altered by the receiving party.

7     **16.3.28 pmix\_server\_stdin\_fn\_t**

8           **Summary**  
9       Pass standard input data to the host environment for transmission to specified recipients.

10          **Format**

PMIx v3.0

C

```
11       typedef pmix_status_t (*pmix_server_stdin_fn_t)(
12                           const pmix_proc_t *source,
13                           const pmix_proc_t targets[],
14                           size_t ntargs,
15                           const pmix_info_t directives[],
16                           size_t ndirs,
17                           const pmix_byte_object_t *bo,
18                           pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

```
19       IN source
20           pmix_proc_t structure of source process (handle)
21       IN targets
22           Array of pmix_proc_t target identifiers (handle)
23       IN ntargs
24           Number of elements in the targets array (integer)
25       IN directives
26           Array of info structures (array of handles)
27       IN ndirs
28           Number of elements in the info array (integer)
29       IN bo
30           Pointer to pmix_byte_object_t containing the payload (handle)
31       IN cbfunc
32           Callback function pmix_op_cbfunc_t (function reference)
33       IN cbdata
34           Data to be passed to the callback function (memory reference)
```

1        Returns one of the following:

- 2        • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
3           will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
4           function prior to returning from the API.
- 5        • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
6           returned *success* - the *cbfunc* will not be called
- 7        • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
8           request, even though the function entry was provided in the server module - the *cbfunc* will not  
9           be called
- 10      • a PMIx error constant indicating either an error in the input or that the request was immediately  
11           processed and failed - the *cbfunc* will not be called

### Required Attributes

12     The following attributes are required to be included in the passed *info* array:

- 13     **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
14        Effective user ID of the connecting process.
- 15     **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
16        Effective group ID of the connecting process.

### Description

17     Passes stdin to the host environment for transmission to specified recipients. The host environment  
18     is responsible for forwarding the data to all locations that host the specified *targets* and delivering  
19     the payload to the PMIx server library connected to those clients.

## 16.3.29 pmix\_server\_grp\_fn\_t

### Summary

22     Request group operations (construct, destruct, etc.) on behalf of a set of processes.  
23

1      **Format**2      *PMIx v4.0*

C

```

2     typedef pmix_status_t (*pmix_server_grp_fn_t)(
3             pmix_group_operation_t op,
4             char grp[],
5             const pmix_proc_t procs[],
6             size_t nprocs,
7             const pmix_info_t directives[],
8             size_t ndirs,
9             pmix_info_cbfunc_t cbfunc,
10            void *cbdata);

```

C

11     **IN** **op**  
12        *pmix\_group\_operation\_t* value indicating operation the host is requested to perform  
13        (integer)

14     **IN** **grp**  
15        Character string identifying the group (string)

16     **IN** **procs**  
17        Array of *pmix\_proc\_t* identifiers of participants (handle)

18     **IN** **nprocs**  
19        Number of elements in the *procs* array (integer)

20     **IN** **directives**  
21        Array of info structures (array of handles)

22     **IN** **ndirs**  
23        Number of elements in the *info* array (integer)

24     **IN** **cbfunc**  
25        Callback function *pmix\_info\_cbfunc\_t* (function reference)

26     **IN** **cbdata**  
27        Data to be passed to the callback function (memory reference)

28     Returns one of the following:

- 29     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
30        will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
31        function prior to returning from the API.
- 32     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
33        returned *success* - the *cbfunc* will not be called
- 34     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
35        request, even though the function entry was provided in the server module - the *cbfunc* will not  
36        be called
- 37     • a PMIx error constant indicating either an error in the input or that the request was immediately  
38        processed and failed - the *cbfunc* will not be called

## Optional Attributes

1 The following attributes may be supported by a host environment.

2 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

3 Requests that the RM assign a new context identifier to the newly created group. The  
4 identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
5 specified in the request. Thus, the value serves as a means of identifying the group within  
6 that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

7 **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

8 Group operation only involves local processes. PMIx implementations are *required* to  
9 automatically scan an array of group members for local vs remote processes - if only local  
10 processes are detected, the implementation need not execute a global collective for the  
11 operation unless a context ID has been requested from the host environment. This can result  
12 in significant time savings. This attribute can be used to optimize the operation by indicating  
13 whether or not only local processes are represented, thus allowing the implementation to  
14 bypass the scan.

15 **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

16 Data collected during group construction to ensure communication between group members  
17 is supported upon completion of the operation.

18 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

19 Participation is optional - do not return an error if any of the specified processes terminate  
20 without having joined. The default is **false**.

21 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

22 Define constraints on the processes that can access the provided data. Only processes that  
23 meet the constraints are allowed to access it.

24 The following attributes may be included in the host's response:

25 **PMIX\_GROUP\_ID** "pmix.grp.id" (char\*)

26 User-provided group identifier - as the group identifier may be used in PMIx operations, the  
27 user is required to ensure that the provided ID is unique within the scope of the host  
28 environment (e.g., by including some user-specific or application-specific prefix or suffix to  
29 the string).

30 **PMIX\_GROUP\_MEMBERSHIP** "pmix.grp.mbrs" (pmix\_data\_array\_t\*)

31 Array **pmix\_proc\_t** identifiers identifying the members of the specified group.

32 **PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (size\_t)

33 Context identifier assigned to the group by the host RM.

34 **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

35 Data collected during group construction to ensure communication between group members  
36 is supported upon completion of the operation.

## 1 Description

2 Perform the specified operation across the identified processes, plus any special actions included in  
3 the directives. Return the result of any special action requests in the callback function when the  
4 operation is completed. Actions may include a request (**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID**)  
5 that the host assign a unique numerical (size\_t) ID to this group - if given, the **PMIX\_RANGE**  
6 attribute will specify the range across which the ID must be unique (default to  
7 **PMIX\_RANGE\_SESSION**).

### 8 16.3.29.1 Group Operation Constants

9 *PMIx v4.0* The **pmix\_group\_operation\_t** structure is a **uint8\_t** value for specifying group  
10 operations. All values were originally defined in version 4 of the standard unless otherwise marked.

11 **PMIX\_GROUP\_CONSTRUCT** Construct a group composed of the specified processes - used by  
12 a PMIx server library to direct host operation.

13 **PMIX\_GROUP\_DESTRUCT** Destruct the specified group - used by a PMIx server library to  
14 direct host operation.

### 15 16.3.30 pmix\_server\_fabric\_fn\_t

#### 16 Summary

17 Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

#### 18 Format

19 *PMIx v4.0*

C

```
20     typedef pmix_status_t (*pmix_server_fabric_fn_t)(
21         const pmix_proc_t *requestor,
22         pmix_fabric_operation_t op,
23         const pmix_info_t directives[],
24         size_t ndirs,
25         pmix_info_cfunc_t cfunc,
26         void *cbdata);
```

C

26 **IN requestor**  
27     **pmix\_proc\_t** identifying the requestor (handle)  
28 **IN op**  
29     **pmix\_fabric\_operation\_t** value indicating operation the host is requested to perform  
30     (integer)  
31 **IN directives**  
32     Array of info structures (array of handles)  
33 **IN ndirs**  
34     Number of elements in the *info* array (integer)

1   **IN cbfunc**  
2    Callback function `pmix_info_cbfunc_t` (function reference)  
3   **IN cldata**  
4    Data to be passed to the callback function (memory reference)  
5   Returns one of the following:  
6    

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result  
7      will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
8      function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and  
9      returned *success* - the *cbfunc* will not be called
- `PMIX_ERR_NOT_SUPPORTED`, indicating that the host environment does not support the  
10     request, even though the function entry was provided in the server module - the *cbfunc* will not  
11     be called
- a PMIx error constant indicating either an error in the input or that the request was immediately  
12     processed and failed - the *cbfunc* will not be called

### Required Attributes

16   The following directives are required to be supported by all hosts to aid users in identifying the  
17    fabric and (if applicable) the device to whom the operation references:

18   **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (`string`)  
19     Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.  
20   **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (`string`)  
21     An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).  
22   **PMIX\_FABRIC\_PLANE** "pmix.fab.plane" (`string`)  
23     ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
24     for information, specifies the plane whose information is to be returned. When used directly  
25     as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
26     planes in the overall system.  
27   **PMIX\_FABRIC\_DEVICE\_INDEX** "pmix.fabdev.idx" (`uint32_t`)  
28     Index of the device within an associated communication cost matrix.

### Description

29   Perform the specified operation. Return the result of any requests in the callback function when the  
30    operation is completed. Operations may, for example, include a request for fabric information. See  
31    `pmix_fabric_t` for a list of expected information to be included in the response. Note that  
32    requests for device index are to be returned in the callback function's array of `pmix_info_t`  
33    using the `PMIX_FABRIC_DEVICE_INDEX` attribute.

## CHAPTER 17

# Tools and Debuggers

---

The term *tool* widely refers to programs executed by the user or system administrator on a command line. Tools frequently interact with either the SMS, user applications, or both to perform administrative and support functions. For example, a debugger tool might be used to remotely control the processes of a parallel application, monitoring their behavior on a step-by-step basis. Historically, such tools were custom-written for each specific host environment due to the customized and/or proprietary nature of the environment's interfaces.

The advent of PMIx offers the possibility for creating portable tools capable of interacting with multiple RMs without modification. Possible use-cases include:

- querying the status of scheduling queues and estimated allocation time for various resource options
- job submission and allocation requests
- querying job status for executing applications
- launching and monitoring applications

Enabling these capabilities requires some extensions to the PMIx Standard (both in terms of APIs and attributes), and utilization of client-side APIs for more tool-oriented purposes.

This chapter defines specific APIs related to tools, provides tool developers with an overview of the support provided by PMIx, and serves to guide RM vendors regarding roles and responsibilities of RMs to support tools. As the number of tool-specific APIs and attributes is fairly small, the bulk of the chapter serves to provide a "theory of operation" for tools and debuggers. Description of the APIs themselves is therefore deferred to the Section 17.5 later in the chapter.

## 17.1 Connection Mechanisms

The key to supporting tools lies in providing mechanisms by which a tool can connect to a PMIx server. Application processes are able to connect because their local RM daemon provides them with the necessary contact information upon execution. A command-line tool, however, isn't spawned by an RM daemon, and therefore lacks the information required for rendezvous with a PMIx server.

Once a tool has started, it initializes PMIx as a tool (via `PMIx_tool_init`) if its access is restricted to PMIx-based informational services such as `PMIx_Query_info`. However, if the

1 tool intends to start jobs, then it must include the **PMIX\_LAUNCHER** attribute to inform the library  
2 of that intent so that the library can initialize and provide access to the corresponding support.

3 Support for tools requires that the PMIx server be initialized with an appropriate attribute  
4 indicating that tool connections are to be allowed. Separate attributes are provided to "fine-tune"  
5 this permission by allowing the environment to independently enable (or disable) connections from  
6 tools executing on nodes other than the one hosting the server itself. The PMIx server library shall  
7 provide an opportunity for the host environment to authenticate and approve each connection  
8 request from a specific tool by calling the **pmix\_server\_tool\_connection\_fn\_t** "hook"  
9 provided in the server module for that purpose. Servers in environments that do not provide this  
10 "hook" shall automatically reject all tool connection requests.

11 Tools can connect to any local or remote PMIx server provided they are either explicitly given the required connection information, or are able to discover it via one of several defined rendezvous  
12 protocols. Connection discovery centers around the existence of *rendezvous files* containing the  
13 necessary connection information, as illustrated in Fig. 17.1.  
14

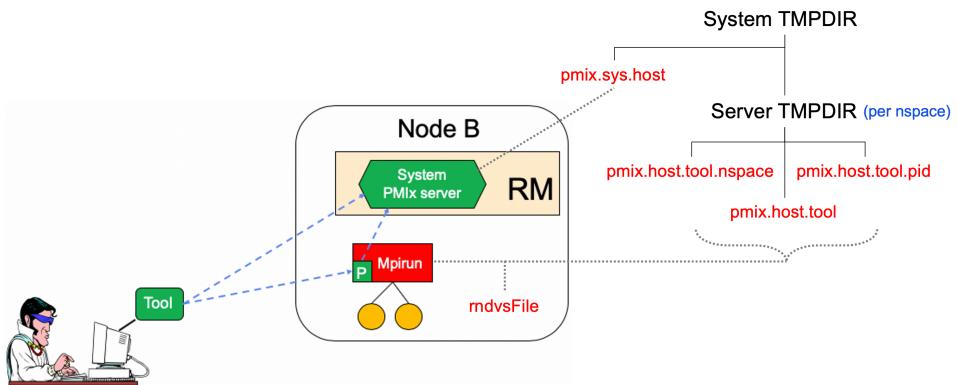


Figure 17.1.: Tool rendezvous files

15 The contents of each rendezvous file are specific to a given PMIx implementation, but should at  
16 least contain the namespace and rank of the server along with its connection URI. Note that tools  
17 linked to one PMIx implementation are therefore unlikely to successfully connect to PMIx server  
18 libraries from another implementation.

19 The top of the directory tree is defined by either the **PMIX\_SYSTEM\_TMPDIR** attribute (if given)  
20 or the **TMPDIR** environmental variable. PMIx servers that are designated as *system servers* by  
21 including the **PMIX\_SERVER\_SYSTEM\_SUPPORT** attribute when calling  
22 **PMIx\_server\_init** will create a rendezvous file in this top-level directory. The filename will  
23 be of the form *pmix.sys.hostname*, where *hostname* is the string returned by the **gethostname**  
24 system call. Note that only one PMIx server on a node can be designated as the system server.

25 Non-system PMIx servers will create a set of three rendezvous files in the directory defined by  
26 either the **PMIX\_SERVER\_TMPDIR** attribute or the **TMPDIR** environmental variable:

- *pmix.host.tool.nspace* where *host* is the string returned by the **gethostname** system call and *nspace* is the namespace of the server.
- *pmix.host.tool.pid* where *host* is the string returned by the **gethostname** system call and *pid* is the PID of the server.
- *pmix.host.tool* where *host* is the string returned by the **gethostname** system call. Note that servers which are not given a namespace-specific **PMIX\_SERVER\_TMPDIR** attribute may not generate this file due to conflicts should multiple servers be present on the node.

The files are identical and may be implemented as symlinks to a single instance. The individual file names are composed so as to aid the search process should a tool wish to connect to a server identified by its namespace or PID.

Servers will additionally provide a rendezvous file in any given location if the path (either absolute or relative) and filename is specified either during **PMIx\_server\_init** using the **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** attribute, or by the **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** environmental variable prior to executing the process containing the server. This latter mechanism may be the preferred mechanism for tools such as debuggers that need to fork/exec a launcher (e.g., "mpiexec") and then rendezvous with it. This is described in more detail in Section 17.2.2.

Rendezvous file ownerships are set to the UID and GID of the server that created them, with permissions set according to the desires of the implementation and/or system administrator policy. All connection attempts are first governed by read access privileges to the target rendezvous file - thus, the combination of permissions, UID, and GID of the rendezvous files act as a first-level of security for tool access.

A tool may connect to as many servers at one time as the implementation supports, but is limited to designating only one such connection as its *primary* server. This is done to avoid confusion when the tool calls an API as to which server should service the request. The first server the tool connects to is automatically designated as the *primary* server.

Tools are allowed to change their primary server at any time via the **PMIx\_tool\_set\_server** API, and to connect/disconnect from a server as many times as desired. Note that standing requests (e.g., event registrations) with the current primary server may be lost and/or may not be transferred when transitioning to another primary server - PMIx implementors are not required to maintain or transfer state across tool-server connections.

Tool process identifiers are assigned by one of the following methods:

- If **PMIX\_TOOL\_NSPACE** is given, then the namespace of the tool will be assigned that value.
  - If **PMIX\_TOOL\_RANK** is also given, then the rank of the tool will be assigned that value.
  - If **PMIX\_TOOL\_RANK** is not given, then the rank will be set to a default value of zero.
- If a process ID has not been provided, then one will be assigned by the host environment upon connection to a server. Users should note that the tool's process ID will be *invalid* until a

connection has been established.

Tool process identifiers remain constant across servers. Thus, it is critical that a system-wide unique namespace be provided if the tool itself sets the identifier, and that host environments provide a system-wide unique identifier in the case where the identifier is set by the server upon connection. The host environment is required to reject any connection request that fails to meet this criterion.

For simplicity, the following descriptions will refer to the:

- **PMIX\_SYSTEM\_TMPDIR** as the directory specified by either the **PMIX\_SYSTEM\_TMPDIR** attribute (if given) or the **TMPDIR** environmental variable.
- **PMIX\_SERVER\_TMPDIR** as the directory specified by either the **PMIX\_SERVER\_TMPDIR** attribute or the **TMPDIR** environmental variable.

The rendezvous methods are automatically employed for the initial tool connection during **PMIx\_tool\_init** unless the **PMIX\_TOOL\_DO\_NOT\_CONNECT** attribute is specified, and on all subsequent calls to **PMIx\_tool\_connect\_to\_server**.

### 17.1.1 Rendezvousing with a local server

Connection to a local PMIx server is pursued according to the following precedence chain based on attributes contained in the call to the **PMIx\_tool\_init** or **PMIx\_tool\_connect\_to\_server** APIs. Servers to which the tool already holds a connection will be ignored. Except where noted, the PMIx library will return an error if the specified file cannot be found, the caller lacks permissions to read it, or the server specified within the file does not respond to or accept the connection — the library will not proceed to check for other connection options as the user specified a particular one to use.

- If **PMIX\_TOOL\_ATTACHMENT\_FILE** is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it. The format of the attachment file is identical to the rendezvous files described in earlier in this section.
- If **PMIX\_SERVER\_URI** or **PMIX\_TCP\_URI** is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. **PMIX\_SERVER\_URI** is the preferred method as it is more generalized — **PMIX\_TCP\_URI** is provided for those cases where the user specifically wants to use a TCP transport for the connection and wants to error out if one isn't available or cannot be used.
- If **PMIX\_SERVER\_PIDINFO** was provided, then the tool will search for a rendezvous file created by a PMIx server of the given PID in the **PMIX\_SERVER\_TMPDIR** directory.
- If **PMIX\_SERVER\_NSPACE** is given, then the tool will search for a rendezvous file created by a PMIx server of the given namespace in the **PMIX\_SERVER\_TMPDIR** directory.
- If **PMIX\_CONNECT\_TO\_SYSTEM** is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the **PMIX\_SYSTEM\_TMPDIR** directory.

- If **PMIX\_CONNECT\_SYSTEM\_FIRST** is given, then the tool will look for a system-level rendezvous file created by a PMIx server in the **PMIX\_SYSTEM\_TMPDIR** directory. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused — the PMIx library will silently continue to the next option.
- By default, the tool will search the directory tree under the **PMIX\_SERVER\_TMPDIR** directory for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error.

Note that there can be multiple local servers - one from the system plus others from launchers and active jobs. The PMIx tool connection search method is not guaranteed to pick a particular server unless directed to do so. Tools can obtain a list of servers available on their local node using the **PMIx\_Query\_info** APIs with the **PMIX\_QUERY\_AVAIL\_SERVERS** key.

### 17.1.2 Connecting to a remote server

Connecting to remote servers is complicated due to the lack of access to the previously-described rendezvous files. Two methods are required to be supported, both based on the caller having explicit knowledge of either connection information or a path to a local file that contains such information:

- If **PMIX\_TOOL\_ATTACHMENT\_FILE** is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it. The format of the attachment file is identical to the rendezvous files described in earlier in this section.
- If **PMIX\_SERVER\_URI** or **PMIX\_TCP\_URI** is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. **PMIX\_SERVER\_URI** is the preferred method as it is more generalized — **PMIX\_TCP\_URI** is provided for those cases where the user specifically wants to use the TCP transport for the connection and wants to error out if it isn't available or cannot be used.

Additional methods may be provided by particular PMIx implementations. For example, the tool may use *ssh* to launch a *probe* process onto the remote node so that the probe can search the **PMIX\_SYSTEM\_TMPDIR** and **PMIX\_SERVER\_TMPDIR** directories for rendezvous files, relaying the discovered information back to the requesting tool. If sufficient information is found to allow for remote connection, then the tool can use it to establish the connection. Note that this method is not required to be supported - it is provided here as an example and left to the discretion of PMIx implementors.

### 17.1.3 Attaching to running jobs

When attaching to a running job, the tool must connect to a PMIx server that is associated with that job - e.g., a server residing in the host environment's local daemon that spawned one or more of the job's processes, or the server residing in the launcher that is overseeing the job. Identifying an

1 appropriate server can sometimes prove challenging, particularly in an environment where multiple  
2 job launchers may be in operation, possibly under control of the same user.

3 In cases where the user has only the one job of interest in operation on the local node (e.g., when  
4 engaged in an interactive session on the node from which the launcher was executed), the normal  
5 rendezvous file discovery method can often be used to successfully connect to the target job, even  
6 in the presence of jobs executed by other users. The permissions and security authorizations can, in  
7 many cases, reliably ensure that only the one connection can be made. However, this is not  
8 guaranteed in all cases.

9 The most common method, therefore, for attaching to a running job is to specify either the PID of  
10 the job's launcher or the namespace of the launcher's job (note that the launcher's namespace  
11 frequently differs from the namespace of the job it has launched). Unless the application processes  
12 themselves act as PMIx servers, connection must be to the servers in the daemons that oversee the  
13 application. This is typically either daemons specifically started by the job's launcher process, or  
14 daemons belonging to the host environment, that are responsible for starting the application's  
15 processes and oversee their execution.

16 Identifying the correct PID or namespace can be accomplished in a variety of ways, including:

- 17 • Using typical OS or host environment tools to obtain a listing of active jobs and perusing those to  
18 find the target launcher.
- 19 • Using a PMIx-based tool attached to a system-level server to query the active jobs and their  
20 command lines, thereby identifying the application of interest and its associated launcher.
- 21 • Manually recording the PID of the launcher upon starting the job.

22 Once the namespace and/or PID of the target server has been identified, either of the previous  
23 methods can be used to connect to it.

#### 24 17.1.4 Tool initialization attributes

25 The following attributes are passed to the `PMIx_tool_init` API for use when initializing the  
26 PMIx library.

27 `PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)`  
28     Name of the namespace to use for this tool.  
29 `PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)`  
30     Rank of this tool.  
31 `PMIX_LAUNCHER "pmix.tool.launcher" (bool)`  
32     Tool is a launcher and needs to create rendezvous files.

#### 33 17.1.5 Tool initialization environmental variables

34 The following environmental variables are used during `PMIx_tool_init` to control various  
35 rendezvous-related operations when the tool is started manually (e.g., on a command line) or by a  
36 fork/exec-like operation.

37 `PMIX_LAUNCHER_PAUSE_FOR_TOOL`

1       Upon completing **PMIx\_server\_init**, the spawned launcher is to generate the  
2       **PMIX\_LAUNCHER\_READY** event and then pause until the spawning tool can connect to it  
3       and provide directives.  
4       **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE**

5       Absolute path of file where the launcher is to store its connection information so that the  
6       spawning tool can connect to it.

## 7     17.1.6 Tool connection attributes

8       These attributes are defined to assist PMIx-enabled tools to connect with a PMIx server by passing  
9       them into either the **PMIx\_tool\_init** or the **PMIx\_tool\_connect\_to\_server** APIs -  
10      thus, they are not typically accessed via the **PMIx\_Get** API.

11      **PMIX\_SERVER\_PIDINFO "pmix.srvr.pidinfo" (pid\_t)**  
12        PID of the target PMIx server for a tool.  
13      **PMIX\_CONNECT\_TO\_SYSTEM "pmix.cnct.sys" (bool)**  
14        The requester requires that a connection be made only to a local, system-level PMIx server.  
15      **PMIX\_CONNECT\_SYSTEM\_FIRST "pmix.cnct.sys.first" (bool)**  
16        Preferentially, look for a system-level PMIx server first.  
17      **PMIX\_SERVER\_URI "pmix.srvr.uri" (char\*)**  
18        URI of the PMIx server to be contacted.  
19      **PMIX\_SERVER\_HOSTNAME "pmix.srvr.host" (char\*)**  
20        Host where target PMIx server is located.  
21      **PMIX\_CONNECT\_MAX\_RETRIES "pmix.tool.mretries" (uint32\_t)**  
22        Maximum number of times to try to connect to PMIx server - the default value is  
23        implementation specific.  
24      **PMIX\_CONNECT\_RETRY\_DELAY "pmix.tool.retry" (uint32\_t)**  
25        Time in seconds between connection attempts to a PMIx server - the default value is  
26        implementation specific.  
27      **PMIX\_TOOL\_DO\_NOT\_CONNECT "pmix.tool.nocon" (bool)**  
28        The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.  
29      **PMIX\_TOOL\_CONNECT\_OPTIONAL "pmix.tool.conopt" (bool)**  
30        The tool shall connect to a server if available, but otherwise continue to operate unconnected.  
31      **PMIX\_TOOL\_ATTACHMENT\_FILE "pmix.tool.attach" (char\*)**  
32        Pathname of file containing connection information to be used for attaching to a specific  
33        server.  
34      **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE "pmix.tool.lncrnd" (char\*)**  
35        Pathname of file where the launcher is to store its connection information so that the  
36        spawning tool can connect to it.  
37      **PMIX\_PRIMARY\_SERVER "pmix.pri.srvr" (bool)**  
38        The server to which the tool is connecting shall be designated the *primary* server once  
39        connection has been accomplished.

## 17.2 Launching Applications with Tools

Tool-directed launches require that the tool include the `PMIX_LAUNCHER` attribute when calling `PMIx_tool_init`. Two launch modes are supported:

- *Direct launch* where the tool itself is directly responsible for launching all processes, including debugger daemons, using either the RM or daemons launched by the tool – i.e., there is no *intermediate launcher* (IL) such as *mpixexec*. The case where the tool is self-contained (i.e., uses its own daemons without interacting with an external entity such as the RM) lies outside the scope of this Standard; and
- *Indirect launch* where all processes are started via an IL such as *mpixexec* and the tool itself is not directly involved in launching application processes or debugger daemons. Note that the IL may utilize the RM to launch processes and/or daemons under the tool’s direction.

Either of these methods can be executed interactively or by a batch script. Note that not all host environments may support the direct launch method.

### 17.2.1 Direct launch

In the direct-launch use-case (Fig. 17.2), the tool itself performs the role of the launcher. Once invoked, the tool connects to an appropriate PMIx server - e.g., a system-level server hosted by the RM. The tool is responsible for assembling the description of the application to be launched (e.g., by parsing its command line) into a spawn request containing an array of `pmix_app_t` applications and `pmix_info_t` job-level information. An allocation of resources may or may not have been made in advance – if not, then the spawn request must include allocation request information.

In addition to the attributes described in `PMIx_Spawn`, the tool may optionally wish to include the following tool-specific attributes in the `job_info` argument to that API (the debugger-related attributes are discussed in more detail in Section 17.4):

- `PMIX_FWD_STDIN "pmix.fwd.stdin" (pmix_rank_t)`

The requester intends to push information from its `stdin` to the indicated process. The local spawn agent should, therefore, ensure that the `stdin` channel to that process remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the spawned job are potential recipients. The requester will issue a call to `PMIx_IOF_push` to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

- `PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)`

Requests that the ability to forward the `stdout` of the spawned processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and other options for delivery of the forwarded output.

- `PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)`

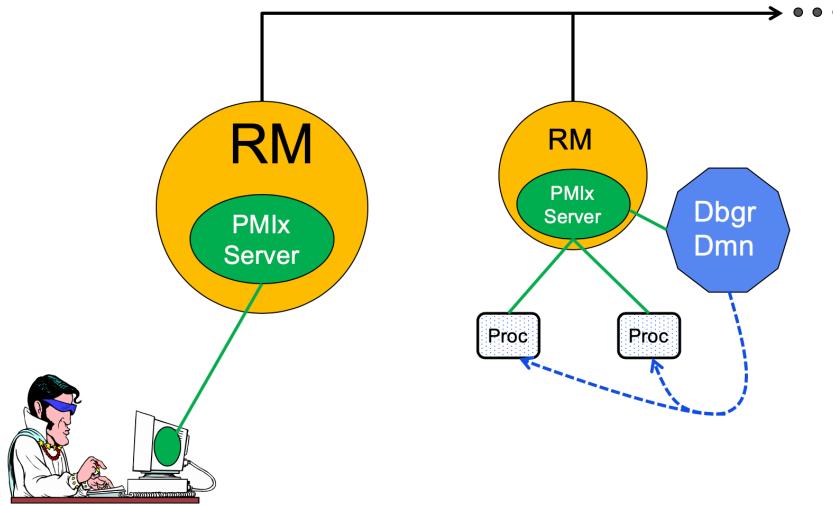


Figure 17.2.: Direct Launch

1 Requests that the ability to forward the `stderr` of the spawned processes be maintained.  
 2 The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and  
 3 other options for delivery of the forwarded output.

4 • **`PMIX_FWD_STDDIAG`** "pmix.fwd.stddiag" (bool)

5 Requests that the ability to forward the diagnostic channel (if it exists) of the spawned  
 6 processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify  
 7 the callback function and other options for delivery of the forwarded output.

8 • **`PMIX_IOF_CACHE_SIZE`** "pmix.iof.cszie" (uint32\_t)

9 The requested size of the PMIx server cache in bytes for each specified channel. By  
 10 default, the server is allowed (but not required) to drop all bytes received beyond the max  
 11 size.

12 • **`PMIX_IOF_DROP_OLDEST`** "pmix.iof.old" (bool)

13 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
 14 cache.

15 • **`PMIX_IOF_DROP_NEWEST`** "pmix.iof.new" (bool)

16 In an overflow situation, the PMIx server is to drop any new bytes received until room  
 17 becomes available in the cache (default).

18 • **`PMIX_IOF_BUFFERING_SIZE`** "pmix.iof.bsize" (uint32\_t)

19 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until  
 20 the specified number of bytes is collected to avoid being called every time a block of IO  
 21 arrives. The PMIx tool library will execute the callback and reset the collection counter.

1 whenever the specified number of bytes becomes available. Any remaining buffered data  
2 will be *flushed* to the callback upon a call to deregister the respective channel.

3 • **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (**uint32\_t**)

4 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
5 size, this prevents IO from being held indefinitely while waiting for another payload to  
6 arrive.

7 • **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (**bool**)

8 Requests that output be prefixed with the nspace,rank of the source and a string  
9 identifying the channel (**stdout**, **stderr**, etc.).

10 • **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (**bool**)

11 Requests that output be marked with the time at which the data was received by the tool -  
12 note that this will differ from the time at which the data was collected from the source.

13 • **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (**bool**)

14 Requests that output be formatted in XML.

15 • **PMIX\_NOHUP** "pmix.nohup" (**bool**)

16 Any processes started on behalf of the calling tool (or the specified namespace, if such  
17 specification is included in the list of attributes) should continue after the tool disconnects  
18 from its server.

19 • **PMIX\_NOTIFY\_JOB\_EVENTS** "pmix.note.jev" (**bool**)

20 Requests that the launcher generate the **PMIX\_EVENT\_JOB\_START**,  
21 **PMIX\_LAUNCH\_COMPLETE**, and **PMIX\_EVENT\_JOB\_END** events. Each event is to  
22 include at least the namespace of the corresponding job and a  
23 **PMIX\_EVENT\_TIMESTAMP** indicating the time the event occurred. Note that the  
24 requester must register for these individual events, or capture and process them by  
25 registering a default event handler instead of individual handlers and then process the  
26 events based on the returned status code. Another common method is to register one event  
27 handler for all job-related events, with a separate handler for non-job events - see  
28 [PMIx\\_Register\\_event\\_handler](#) for details.

29 • **PMIX\_NOTIFY\_COMPLETION** "pmix.notecomp" (**bool**)

30 Requests that the launcher generate the **PMIX\_EVENT\_JOB\_END** event for normal or  
31 abnormal termination of the spawned job. The event shall include the returned status code  
32 (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job; the identity (**PMIX\_PROCID**)  
33 and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if applicable; and a  
34 **PMIX\_EVENT\_TIMESTAMP** indicating the time the termination occurred. Note that the  
35 requester must register for the event or capture and process it within a default event  
36 handler.

37 • **PMIX\_LOG\_JOB\_EVENTS** "pmix.log.jev" (**bool**)

- 1 Requests that the launcher log the **PMIX\_EVENT\_JOB\_START**,  
2 **PMIX\_LAUNCH\_COMPLETE**, and **PMIX\_EVENT\_JOB\_END** events using **PMIx\_Log**,  
3 subject to the logging attributes of Section 12.4.3.
- 4 • **PMIX\_LOG\_COMPLETION "pmix.logcomp" (bool)**  
5 Requests that the launcher log the **PMIX\_EVENT\_JOB\_END** event for normal or  
6 abnormal termination of the spawned job using **PMIx\_Log**, subject to the logging  
7 attributes of Section 12.4.3. The event shall include the returned status code  
8 (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job; the identity (**PMIX\_PROCID**)  
9 and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if applicable; and a  
10 **PMIX\_EVENT\_TIMESTAMP** indicating the time the termination occurred.
- 11 • **PMIX\_DEBUG\_STOP\_ON\_EXEC "pmix.dbg.exec" (bool)**  
12 Included in either the **pmix\_info\_t** array in a **pmix\_app\_t** description (if the  
13 directive applies only to that application) or in the **job\_info** array if it applies to all  
14 applications in the given spawn request. Indicates that the application is being spawned  
15 under a debugger, and that the local launch agent is to pause the resulting application  
16 processes on first instruction for debugger attach. The launcher (RM or IL) is to generate  
17 the **PMIX\_LAUNCH\_COMPLETE** event when all processes are stopped at the exec point.
- 18 • **PMIX\_DEBUG\_STOP\_IN\_INIT "pmix.dbg.init" (bool)**  
19 Included in either the **pmix\_info\_t** array in a **pmix\_app\_t** description (if the  
20 directive applies only to that application) or in the **job\_info** array if it applies to all  
21 applications in the given spawn request. Indicates that the specified application is being  
22 spawned under a debugger. The PMIx client library in each resulting application process  
23 shall notify its PMIx server that it is pausing and then pause during **PMIx\_Init** of the  
24 spawned processes until receipt of the **PMIX\_DEBUGGER\_RELEASE** event. The launcher  
25 (RM or IL) is responsible for generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY**  
26 event when all processes have reached the pause point.
- 27 • **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY "pmix.dbg.notify" (bool)**  
28 Included in either the **pmix\_info\_t** array in a **pmix\_app\_t** description (if the  
29 directive applies only to that application) or in the **job\_info** array if it applies to all  
30 applications in the given spawn request. Indicates that the specified application is being  
31 spawned under a debugger. The resulting application processes are to notify their server  
32 (by generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event) when they reach  
33 some application-determined location and pause at that point until receipt of the  
34 **PMIX\_DEBUGGER\_RELEASE** event. The launcher (RM or IL) is responsible for  
35 generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event when all processes have  
36 indicated they are at the pause point.

37 The tool then calls the **PMIx\_Spawn** API so that the PMIx library can communicate the spawn  
38 request to the server.

Upon receipt, the PMIx server library passes the spawn request to its host RM daemon for processing via the `pmix_server_spawn_fn_t` server module function. If this callback was not provided, then the PMIx server library will return the `PMIX_ERR_NOT_SUPPORTED` error status.

If an allocation must be made, then the host environment is responsible for communicating the request to its associated scheduler. Once resources are available, the host environment initiates the launch process to start the job. The host environment must parse the spawn request for relevant directives, returning an error if any required directive cannot be supported. Optional directives may be ignored if they cannot be supported.

Any error while executing the spawn request must be returned by `PMIx_Spawn` to the requester. Once the spawn request has succeeded in starting the specified processes, the request will return `PMIX_SUCCESS` back to the requester along with the namespace of the started job. Upon termination of the spawned job, the host environment must generate a `PMIX_EVENT_JOB_END` event for normal or abnormal termination if requested to do so. The event shall include:

- the returned status code (`PMIX_JOB_TERM_STATUS`) for the corresponding job;
- the identity (`PMIX_PROCID`) and exit status (`PMIX_EXIT_CODE`) of the first failed process, if applicable;
- a `PMIX_EVENT_TIMESTAMP` indicating the time the termination occurred; plus
- any other info provided by the host environment.

## 17.2.2 Indirect launch

In the indirect launch use-case, the application processes are started via an intermediate launcher (e.g., `mpirexec`) that is itself started by the tool (see Fig 17.3). Thus, at a high level, this is a two-stage launch procedure to start the application: the tool starts the IL, which then starts the applications. In practice, additional steps may be involved if, for example, the IL starts its own daemons to shepherd the application processes.

A key aspect of this operational mode is the avoidance of any requirement that the tool parse and/or understand the command line of the IL. Instead, the indirect launch support relies on `PMIx_Spawn` API to abstract the tool-launcher interaction required to start the IL and connect to it, and then uses a second call to `PMIx_Spawn` to request that the IL spawn the actual job.

The tool spawns the IL using the same procedure for launching an application - it assembles the description of the IL (e.g., by parsing its command line) into a spawn request containing an array of `pmix_app_t` and `pmix_info_t` job-level information. An allocation of resources for the IL itself may or may not be required – if it is, then the allocation must be made in advance or the spawn request must include allocation request information.

The tool may optionally wish to include the following tool-specific attributes in the `job_info` argument to `PMIx_Spawn` - note that these attributes refer to the behavior of the IL itself and not the eventual job to be launched:

- `PMIX_FWD_STDIN "pmix.fwd.stdin"` (`pmix_rank_t`)

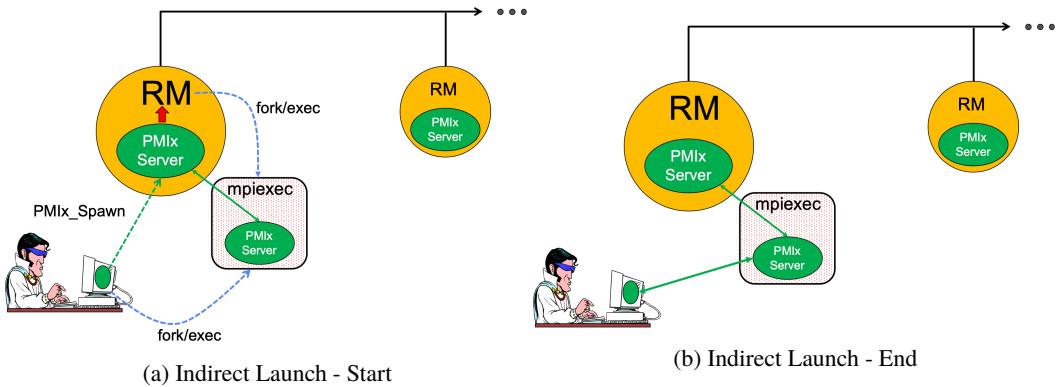


Figure 17.3.: Indirect launch procedure

1           The requester intends to push information from its `stdin` to the indicated process. The  
 2           local spawn agent should, therefore, ensure that the `stdin` channel to that process  
 3           remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the  
 4           spawned job are potential recipients. The requester will issue a call to `PMIx_IOF_push`  
 5           to initiate the actual forwarding of information to specified targets - this attribute simply  
 6           requests that the IL retain the ability to forward the information to the designated targets.

- 7     • **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
       Requests that the ability to forward the `stdout` of the spawned processes be maintained.  
       The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and  
       other options for delivery of the forwarded output.
- 11    • **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
       Requests that the ability to forward the `stderr` of the spawned processes be maintained.  
       The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and  
       other options for delivery of the forwarded output.
- 15    • **PMIX\_FWD\_STDDIAG** "pmix.fwd.stddiag" (bool)  
       Requests that the ability to forward the diagnostic channel (if it exists) of the spawned  
       processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify  
       the callback function and other options for delivery of the forwarded output.
- 19    • **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.cszie" (uint32\_t)  
       The requested size of the PMIx server cache in bytes for each specified channel. By  
       default, the server is allowed (but not required) to drop all bytes received beyond the max  
       size.
- 23    • **PMIX\_IOF\_DROP\_OLEDEST** "pmix.iof.old" (bool)  
       In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
       cache.

- **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)  
In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).
- **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)  
Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be *flushed* to the callback upon a call to deregister the respective channel.
- **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)  
Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
- **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (bool)  
Requests that output be prefixed with the nspace,rank of the source and a string identifying the channel (**stdout**, **stderr**, etc.).
- **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (bool)  
Requests that output be marked with the time at which the data was received by the tool - note that this will differ from the time at which the data was collected from the source.
- **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)  
Requests that output be formatted in XML.
- **PMIX\_NOHUP** "pmix.nohup" (bool)  
Any processes started on behalf of the calling tool (or the specified namespace, if such specification is included in the list of attributes) should continue after the tool disconnects from its server.
- **PMIX\_LAUNCHER\_DAEMON** "pmix.lnch.dmn" (char\*)  
Path to executable that is to be used as the backend daemon for the launcher. This replaces the launcher's own daemon with the specified executable. Note that the user is therefore responsible for ensuring compatibility of the specified executable and the host launcher.
- **PMIX\_FORKEXEC\_AGENT** "pmix.frkex.agnt" (char\*)  
Path to executable that the launcher's backend daemons are to fork/exec in place of the actual application processes. The fork/exec agent shall connect back (as a PMIx tool) to the launcher's daemon to receive its spawn instructions, and is responsible for starting the actual application process it replaced. See Section 17.4.3 for details.
- **PMIX\_EXEC\_AGENT** "pmix.exec.agnt" (char\*)  
Path to executable that the launcher's backend daemons are to fork/exec in place of the actual application processes. The launcher's daemon shall pass the full command line of the application on the command line of the exec agent, which shall not connect back to the

1 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
2 process in its own place. See Section 17.4.3 for details.

3 The tool then calls the **PMIx\_Spawn** API so that the PMIx library can either communicate the  
4 spawn request to the server (if connected to one), or locally spawn the IL itself if not connected to a  
5 server and the PMIx implementation includes self-spawn support. **PMIx\_Spawn** shall return an  
6 error if neither of these conditions is met, or if the tool fails to successfully connect to the IL after  
7 spawning it.

8 Upon successful return from **PMIx\_Spawn**:

- 9   • The namespace of the IL shall be returned in the *nspace* parameter of the **PMIx\_Spawn** API (or  
10    in the **pmix\_spawn\_cbfunc\_t** for the non-blocking form of that API).  
11   • The tool shall be connected to the IL with the IL acting as the tool's *primary* server.

12 Once **PMIx\_Spawn** has returned, the tool can proceed to spawn the actual application according  
13 to the procedure described in Section 17.2.1.

## 14 17.2.3 Tool spawn-related attributes

15 Tools are free to utilize the spawn attributes available to applications (see 11.2.4) when  
16 constructing a spawn request, but can also utilize the following attributes that are specific to  
17 tool-based spawn operations:

18 **PMIX\_FWD\_STDIN "pmix.fwd.stdin"** (**pmix\_rank\_t**)

19   The requester intends to push information from its **stdin** to the indicated process. The  
20   local spawn agent should, therefore, ensure that the **stdin** channel to that process remains  
21   available. A rank of **PMIX\_RANK\_WILDCARD** indicates that all processes in the spawned  
22   job are potential recipients. The requester will issue a call to **PMIx\_IOF\_push** to initiate  
23   the actual forwarding of information to specified targets - this attribute simply requests that  
24   the IL retain the ability to forward the information to the designated targets.

25 **PMIX\_FWD\_STDOUT "pmix.fwd.stdout"** (**bool**)

26   Requests that the ability to forward the **stdout** of the spawned processes be maintained.  
27   The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
28   other options for delivery of the forwarded output.

29 **PMIX\_FWD\_STDERR "pmix.fwd.stderr"** (**bool**)

30   Requests that the ability to forward the **stderr** of the spawned processes be maintained.  
31   The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
32   other options for delivery of the forwarded output.

33 **PMIX\_FWD\_STDDIAG "pmix.fwd.stddiag"** (**bool**)

34   Requests that the ability to forward the diagnostic channel (if it exists) of the spawned  
35   processes be maintained. The requester will issue a call to **PMIx\_IOF\_pull** to specify the  
36   callback function and other options for delivery of the forwarded output.

37 **PMIX\_NOHUP "pmix.nohup"** (**bool**)

1 Any processes started on behalf of the calling tool (or the specified namespace, if such  
2 specification is included in the list of attributes) should continue after the tool disconnects  
3 from its server.

4 **PMIX\_LAUNCHER\_DAEMON** "pmix.lnch.dmn" (**char\***)

5 Path to executable that is to be used as the backend daemon for the launcher. This replaces  
6 the launcher's own daemon with the specified executable. Note that the user is therefore  
7 responsible for ensuring compatibility of the specified executable and the host launcher.

8 **PMIX\_FORKEXEC\_AGENT** "pmix.frkex.agnt" (**char\***)

9 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
10 application processes. The fork/exec agent shall connect back (as a PMIx tool) to the  
11 launcher's daemon to receive its spawn instructions, and is responsible for starting the actual  
12 application process it replaced. See Section 17.4.3 for details.

13 **PMIX\_EXEC\_AGENT** "pmix.exec.agnt" (**char\***)

14 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
15 application processes. The launcher's daemon shall pass the full command line of the  
16 application on the command line of the exec agent, which shall not connect back to the  
17 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
18 process in its own place. See Section 17.4.3 for details.

## 19 17.2.4 Tool rendezvous-related events

20 These constants refer to events relating to rendezvous of a tool and launcher during spawn of the IL.  
21 The events serve as a form of event-driven "handshake" between the two processes, thereby  
22 allowing the tool to provide directives to the IL prior to the IL launching any processes.

23 **PMIX\_LAUNCH\_DIRECTIVE** When a launcher wishes to capture directives sent to it by a  
24 PMIx-enabled tool, it shall register to receive this event. The specified callback function will  
25 be invoked when the event is received from the tool, thereby allowing the launcher to harvest  
26 the directives for its purposes.

27 **PMIX\_LAUNCHER\_READY** When an application launcher (e.g., *mpieexec*) is ready to receive  
28 directives from a PMIx-enabled tool, it shall generate this event to signal the tool to send the  
29 directives. Directives are conveyed to the launcher via the tool generating the  
30 **PMIX\_LAUNCH\_DIRECTIVE** event - see [PMIx\\_Notify\\_event](#) for details.

## 31 17.3 IO Forwarding

32 Underlying the operation of many tools is a common need to forward **stdin** from the tool to  
33 targeted processes, and to return **stdout/stderr** from those processes to the tool (e.g., for  
34 display on the user's console). Historically, each tool developer was responsible for creating their  
35 own IO forwarding subsystem. However, the introduction of PMIx as a standard mechanism for  
36 interacting between applications and the host environment has made it possible to relieve tool  
37 developers of this burden.

38 This section defines functions by which tools can request forwarding of input/output to/from other  
39 processes and serves as a design guide to:

- provide tool developers with an overview of the expected behavior of the PMIx IO forwarding support;
- guide RM vendors regarding roles and responsibilities expected of the RM to support IO forwarding; and
- provide insight into the thinking of the PMIx community behind the definition of the PMIx IO forwarding APIs.

Note that the forwarding of IO via PMIx requires that both the host environment and the tool support PMIx, but does not impose any similar requirements on the application itself.

The responsibility of the host environment in forwarding of IO falls into the following areas:

- Capturing output from specified processes.
- Forwarding that output to the host of the PMIx server library that requested it.
- Delivering that payload to the PMIx server library via the `PMIx_server_IOF_deliver` API for final dispatch to the requesting tool.

It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the requesting client. This may require caching of output until a forwarding registration is received, as governed by the corresponding IO forwarding attributes of Section 17.3.5 that are supported by the implementation.

### 17.3.1 Forwarding stdout/stderr

At an appropriate point in its operation (usually during startup), a tool will utilize the `PMIx_tool_init` function to connect to a PMIx server. The PMIx server can be hosted by an RM daemon or could be embedded in a library-provided starter program such as *mpexec* - in terms of IO forwarding, the operations remain the same either way. For purposes of this discussion, we will assume the server is in an RM daemon and that the application processes are directly launched by the RM, as shown in Fig 17.4.

Once the tool has connected to the target server, it can request that processes be spawned on its behalf or that output from a specified set of existing processes in a given executing application be forwarded to it. Requests to spawn processes should include the `PMIX_FWD_STDIN`, `PMIX_FWD_STDOUT`, and/or `PMIX_FWD_STDERR` attributes if the tool intends to request that the corresponding streams be forwarded at some point during execution.

Note that requests to capture output from existing processes via the `PMIx_IOF_pull` API, and/or to forward input to specified processes via the `PMIx_IOF_push` API, can only succeed if the required attributes to retain that ability were passed when the corresponding job was spawned. The host is required to return an error for all such requests in cases where this condition is not met.

Two modes are supported when requesting that the host forward standard output/error via the `PMIx_IOF_pull` API - these can be controlled by including one of the following attributes in the *info* array passed to that function:

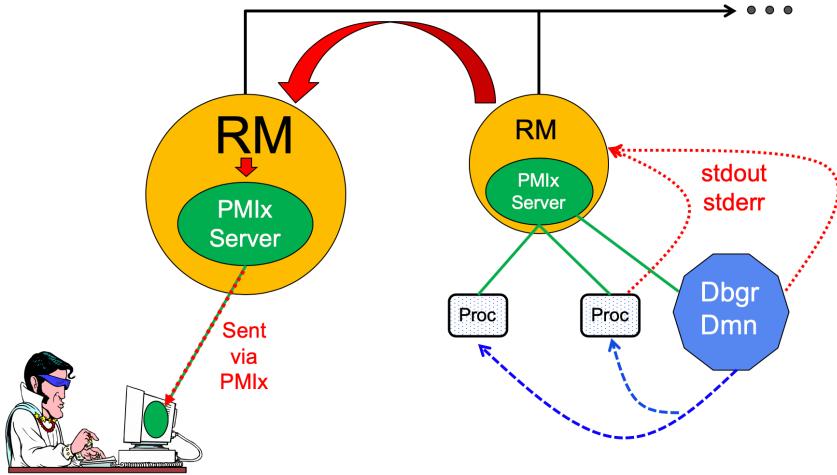


Figure 17.4.: Forwarding stdout/stderr

1     • **PMIX\_IOF\_COPY "pmix.iof.cpy" (bool)**

2         Requests that the host environment deliver a copy of the specified output stream(s) to the  
 3         tool, letting the stream(s) continue to also be delivered to the default location. This allows  
 4         the tool to tap into the output stream(s) without redirecting it from its current final  
 5         destination.

6     • **PMIX\_IOF\_REDIRECT "pmix.iof.redir" (bool)**

7         Requests that the host environment intercept the specified output stream(s) and deliver it  
 8         to the requesting tool instead of its current final destination. This might be used, for  
 9         example, during a debugging procedure to avoid injection of debugger-related output into  
 10        the application's results file. The original output stream(s) destination is restored upon  
 11        termination of the tool. This is the default mode of operation.

12       When requesting to forward **stdout/stderr**, the tool can specify several formatting options to  
 13       be used on the resulting output stream. These include:

14     • **PMIX\_IOF\_TAG\_OUTPUT "pmix.iof.tag" (bool)**

15         Requests that output be prefixed with the nspace,rank of the source and a string  
 16         identifying the channel (**stdout**, **stderr**, etc.).

17     • **PMIX\_IOF\_TIMESTAMP\_OUTPUT "pmix.iof.ts" (bool)**

18         Requests that output be marked with the time at which the data was received by the tool -  
 19         note that this will differ from the time at which the data was collected from the source.

20     • **PMIX\_IOF\_XML\_OUTPUT "pmix.iof.xml" (bool)**

21         Requests that output be formatted in XML.

1       The PMIx client in the tool is responsible for formatting the output stream. Note that output from  
2       multiple processes will often be interleaved due to variations in arrival time - ordering of output is  
3       not guaranteed across processes and/or nodes.

#### 4     17.3.2 Forwarding `stdin`

5       A tool is not necessarily a child of the RM as it may have been started directly from the command  
6       line. Thus, provision must be made for the tool to collect its `stdin` and pass it to the host RM (via  
7       the PMIx server) for forwarding. Two methods of support for forwarding of `stdin` are defined:

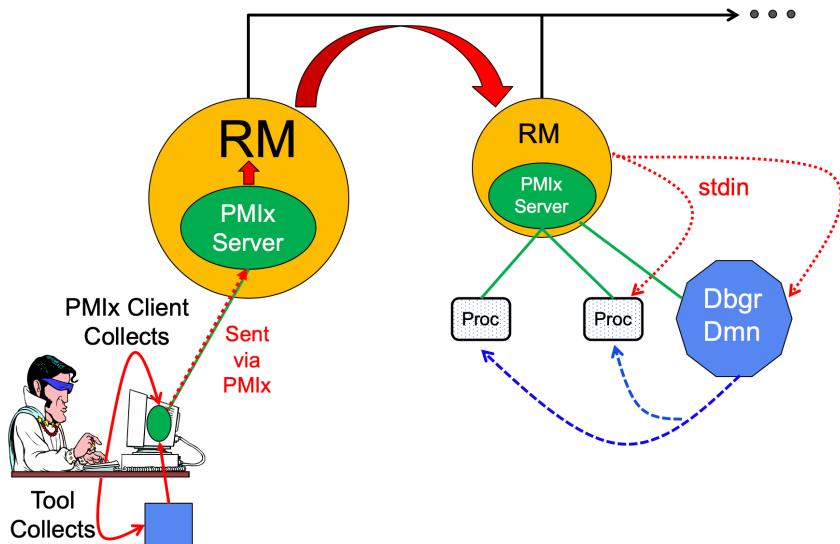


Figure 17.5.: Forwarding `stdin`

- 8       internal collection by the PMIx tool library itself. This is requested via the  
9       `PMIX_IOF_PUSH_STDIN` attribute in the `PMIx_IOF_push` call. When this mode is  
10      selected, the tool library begins collecting all `stdin` data and internally passing it to the local  
11      server for distribution to the specified target processes. All collected data is sent to the same  
12      targets until `stdin` is closed, or a subsequent call to `PMIx_IOF_push` is made that includes  
13      the `PMIX_IOF_COMPLETE` attribute indicating that forwarding of `stdin` is to be terminated.
- 14      external collection directly by the tool. It is assumed that the tool will provide its own  
15      code/mechanism for collecting its `stdin` as the tool developers may choose to insert some  
16      filtering and/or editing of the stream prior to forwarding it. In addition, the tool can directly  
17      control the targets for the data on a per-call basis – i.e., each call to `PMIx_IOF_push` can  
18      specify its own set of target recipients for that particular *blob* of data. Thus, this method provides  
19      maximum flexibility, but requires that the tool developer provide their own code to capture  
20      `stdin`.

1 Note that it is the responsibility of the RM to forward data to the host where the target process(es)  
2 are executing, and for the host daemon on that node to deliver the data to the **stdin** of target  
3 process(es). The PMIx server on the remote node is not involved in this process. Systems that do  
4 not support forwarding of **stdin** shall return **PMIX\_ERR\_NOT\_SUPPORTED** in response to a  
5 forwarding request.

#### Advice to users

6 Scalable forwarding of **stdin** represents a significant challenge. Most environments will at least  
7 handle a *send-to-1* model whereby **stdin** is forwarded to a single identified process, and  
8 occasionally an additional *send-to-all* model where **stdin** is forwarded to all processes in the  
9 application. Users are advised to check their host environment for available support as the  
10 distribution method lies outside the scope of PMIx.

11 **Stdin** buffering by the RM and/or PMIx library can be problematic. If any targeted recipient is  
12 slow reading data (or decides never to read data), then the data must be buffered in some  
13 intermediate daemon or the PMIx tool library itself. Thus, piping a large amount of data into  
14 **stdin** can result in a very large memory footprint in the system management stack or the tool.  
15 Best practices, therefore, typically focus on reading of input files by application processes as  
16 opposed to forwarding of **stdin**.

### 17.3.3 IO Forwarding Channels

18 *PMIx v3.0* The **pmix\_ifc\_channel\_t** structure is a **uint16\_t** type that defines a set of bit-mask flags  
19 for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple  
20 channels.

21 **PMIX\_FWD\_NO\_CHANNELS** Forward no channels.  
22 **PMIX\_FWD\_STDIN\_CHANNEL** Forward **stdin**.  
23 **PMIX\_FWD\_STDOUT\_CHANNEL** Forward **stdout**.  
24 **PMIX\_FWD\_STDERR\_CHANNEL** Forward **stderr**.  
25 **PMIX\_FWD\_STDDIAG\_CHANNEL** Forward **stddiag**, if available.  
26 **PMIX\_FWD\_ALL\_CHANNELS** Forward all available channels.

### 17.3.4 IO Forwarding constants

28 **PMIX\_ERR\_IOF\_FAILURE** An IO forwarding operation failed - the affected channel will be  
29 included in the notification.  
30 **PMIX\_ERR\_IOF\_COMPLETE** IO forwarding of the standard input for this process has  
31 completed - i.e., the **stdin** file descriptor has closed.

## 17.3.5 IO Forwarding attributes

The following attributes are used to control IO forwarding behavior at the request of tools. Use of the attributes is optional - any option not provided will revert to some implementation-specific value.

**PMIX\_IOF\_CACHE\_SIZE "pmix.iof.csizes" (uint32\_t)**  
The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

**PMIX\_IOF\_DROP\_OLDEST "pmix.iof.old" (bool)**  
In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.

**PMIX\_IOF\_DROP\_NEWEST "pmix.iof.new" (bool)**  
In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).

**PMIX\_IOF\_BUFFERING\_SIZE "pmix.iof.bsize" (uint32\_t)**  
Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be *flushed* to the callback upon a call to deregister the respective channel.

**PMIX\_IOF\_BUFFERING\_TIME "pmix.iof.btime" (uint32\_t)**  
Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.

**PMIX\_IOF\_COMPLETE "pmix.iof.cmp" (bool)**  
Indicates that the specified IO channel has been closed by the source.

**PMIX\_IOF\_TAG\_OUTPUT "pmix.iof.tag" (bool)**  
Requests that output be prefixed with the nspace,rank of the source and a string identifying the channel (**stdout**, **stderr**, etc.).

**PMIX\_IOF\_TIMESTAMP\_OUTPUT "pmix.iof.ts" (bool)**  
Requests that output be marked with the time at which the data was received by the tool - note that this will differ from the time at which the data was collected from the source.

**PMIX\_IOF\_XML\_OUTPUT "pmix.iof.xml" (bool)**  
Requests that output be formatted in XML.

**PMIX\_IOF\_PUSH\_STDIN "pmix.iof.stdin" (bool)**  
Requests that the PMIx library collect the **stdin** of the requester and forward it to the processes specified in the **PMIX\_IOF\_push** call. All collected data is sent to the same targets until **stdin** is closed, or a subsequent call to **PMIX\_IOF\_push** is made that includes the **PMIX\_IOF\_COMPLETE** attribute indicating that forwarding of **stdin** is to be terminated.

**PMIX\_IOF\_COPY "pmix.iof.cpy" (bool)**  
Requests that the host environment deliver a copy of the specified output stream(s) to the tool, letting the stream(s) continue to also be delivered to the default location. This allows the tool to tap into the output stream(s) without redirecting it from its current final destination.

```
1 PMIX_IOF_REDIRECT "pmix.iof.redir" (bool)
2 Requests that the host environment intercept the specified output stream(s) and deliver it to
3 the requesting tool instead of its current final destination. This might be used, for example,
4 during a debugging procedure to avoid injection of debugger-related output into the
5 application's results file. The original output stream(s) destination is restored upon
6 termination of the tool.
```

## 17.4 Debugger Support

```
8 Debuggers are a class of tool that merits special consideration due to their particular requirements
9 for access to job-related information and control over process execution. The primary advantage of
10 using PMIx for these purposes lies in the resulting portability of the debugger as it can be used with
11 any system and/or programming model that supports PMIx. In addition to the general tool support
12 described above, debugger support includes:
```

- Co-location, co-spawn, and communication wireup of debugger daemons for scalable launch.  
This includes providing debugger daemons with endpoint connection information across the  
daemons themselves.
- Identification of the job that is to be debugged. This includes automatically providing debugger  
daemons with the job-level information for their target job.

```
18 Debuggers can also utilize the options in the PMIx_Spawn API to exercise a degree of control
19 over spawned jobs for debugging purposes. For example, a debugger can utilize the environmental
20 parameter attributes of Section 11.2.4 to request LD_PRELOAD of a memory interceptor library
21 prior to spawning an application process, or interject a custom fork/exec agent to shepherd the
22 application process.
```

```
23 A key element of the debugging process is the ability of the debugger to require that processes
24 pause at some well-defined point, thereby providing the debugger with an opportunity to attach and
25 control execution. The actual implementation of the pause lies outside the scope of PMIx - it
26 typically requires either the launcher or the application itself to implement the necessary
27 operations. However, PMIx does provide several standard attributes by which the debugger can
28 specify the desired attach point:
```

- **PMIX\_DEBUG\_STOP\_ON\_EXEC** "pmix.dbg.exec" (bool)  
Included in either the **pmix\_info\_t** array in a **pmix\_app\_t** description (if the  
directive applies only to that application) or in the **job\_info** array if it applies to all  
applications in the given spawn request. Indicates that the application is being spawned  
under a debugger, and that the local launch agent is to pause the resulting application  
processes on first instruction for debugger attach. The launcher (RM or IL) is to generate  
the **PMIX\_LAUNCH\_COMPLETE** event when all processes are stopped at the exec point.  
Launchers that cannot support this operation shall return an error from the **PMIx\_Spawn**  
API if this behavior is requested.
- **PMIX\_DEBUG\_STOP\_IN\_INIT** "pmix.dbg.init" (bool)

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The PMIx client library in each resulting application process shall notify its PMIx server that it is pausing and then pause during `PMIx_Init` of the spawned processes until receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have reached the pause point. PMIx implementations that do not support this operation shall return an error from `PMIx_Init` if this behavior is requested. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

- `PMIX_DEBUG_WAIT_FOR_NOTIFY "pmix.dbg.notify" (bool)`

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The resulting application processes are to notify their server (by generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some application-determined location and pause at that point until receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they are at the pause point. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

Note that there is no mechanism by which the PMIx library or the launcher can verify that an application will recognize and support the `PMIX_DEBUG_WAIT_FOR_NOTIFY` request. Debuggers utilizing this attachment method must, therefore, be prepared to deal with the case where the application fails to recognize and/or honor the request.

If the PMIx implementation and/or the host environment support it, debuggers can utilize the `PMIx_Query_info` API to determine which features are available via the `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute.

- `PMIX_DEBUG_STOP_IN_INIT` by checking `PMIX_CLIENT_ATTRIBUTES` for the `PMIx_Init` API.
- `PMIX_DEBUG_STOP_ON_EXEC` by checking `PMIX_HOST_ATTRIBUTES` for the `PMIx_Spawn` API.

The target namespace or process (as given by the debugger in the spawn request) shall be provided to each daemon in its job-level information via the `PMIX_DEBUG_TARGET` attribute. Debugger daemons are responsible for self-determining their specific target process(es), and can then utilize the `PMIx_Query_info` API to obtain information about them (see Fig 17.6) - e.g., to obtain the PIDs of the local processes to which they need to attach. PMIx provides the `pmix_proc_info_t` structure for organizing information about a process' PID, location, and state. Debuggers may request information on a given job at two levels:

- ```

1   • PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
2     Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
3     process in the specified namespace, ordered by process job rank. REQUIRED
4     QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being
5     queried.
6   • PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
7     Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
8     process in the specified namespace executing on the same node as the requester, ordered
9     by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the
10    namespace whose local process table is being queried. OPTIONAL QUALIFIER:
11    PMIX_HOSTNAME indicating the host whose local process table is being queried. By
12    default, the query assumes that the host upon which the request was made is to be used.
13
14  Note that the information provided in the returned proctable represents a snapshot in time. Any
15  process, regardless of role (tool, client, debugger, etc.) can obtain the proctable of a given
16  namespace so long as it has the system-determined authorizations to do so. The list of namespaces
17  available via a given server can be obtained using the PMIx_Query_info API with the
PMIX_QUERY_NAMESPACES key.

```

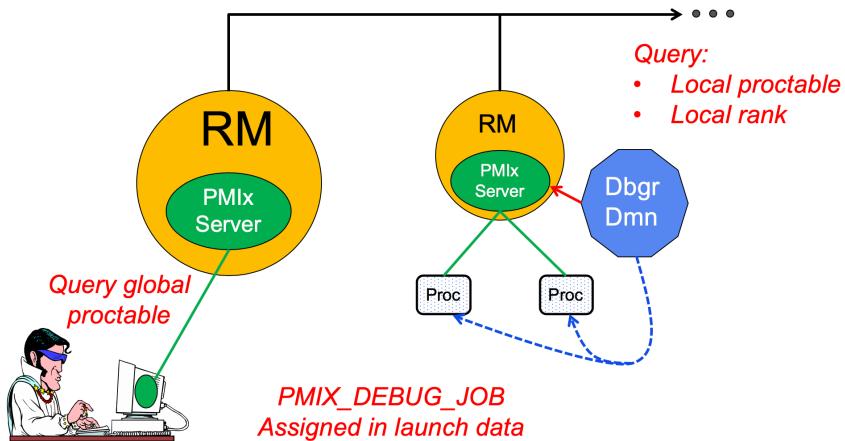


Figure 17.6.: Obtaining proctables

```

18
19  Debugger daemons can be started in two ways - either at the same time the application is spawned,
or separately at a later time.

```

## 20 17.4.1 Co-Location of Debugger Daemons

```

21  Debugging operations typically require the use of daemons that are located on the same node as the
22  processes they are attempting to debug. The debugger can, of course, specify its own mapping
23  method when issuing its spawn request or utilize its own internal launcher to place the daemons.

```

1 However, when attaching to a running job, PMIx provides debuggers with a simplified method for  
2 requesting that the launcher associated with the job *co-locate* the required daemons. Debuggers can  
3 request *co-location* of their daemons by adding the following attributes to the **PMIx\_Spawn** used  
4 to spawn them:

- 5 • **PMIX\_DEBUGGER\_DAEMONS** - indicating that the launcher is being asked to spawn debugger  
6 daemons.
- 7 • **PMIX\_DEBUG\_TARGET** - indicating the job or process that is to be debugged. This allows the  
8 launcher to identify the processes to be debugged and their location. Note that the debugger job  
9 shall be assigned its own namespace (different from that of the job it is being spawned to debug)  
10 and each daemon will be assigned a unique rank within that namespace.
- 11 • **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** - specifies the number of debugger daemons to be  
12 co-located per target process.
- 13 • **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** - specifies the number of debugger daemons to be  
14 co-located per node where at least one target process is executing.

15 Debugger daemons spawned in this manner shall be provided with the typical PMIx information for  
16 their own job plus the target they are to debug via the **PMIX\_DEBUG\_TARGET** attribute. The  
17 debugger daemons spawned on a given node are responsible for self-determining their specific  
18 target process(es) - e.g., by referencing their own **PMIX\_LOCAL\_RANK** in the daemon debugger  
19 job versus the corresponding **PMIX\_LOCAL\_RANK** of the target processes on the node. Note that  
20 the debugger will be attaching to the application processes at some arbitrary point in the  
21 application's execution unless some method for pausing the application (e.g., by providing a PMIx  
22 directive at time of launch, or via a tool using the **PMIx\_Job\_control** API to direct that the  
23 process be paused) has been employed.

---

### Advice to users

---

24 Note that the tool calling **PMIx\_Spawn** to request the launch of the debugger daemons is *not*  
25 included in the resulting job - i.e., the debugger daemons do not inherit the namespace of the tool.  
26 Thus, collective operations and notifications that target the debugger daemon job will not include  
27 the tool unless the namespace/rank of the tool is explicitly included.

---

## 17.4.2 Co-Spawn of Debugger Daemons

In the case where a job is being spawned under the control of a debugger, PMIx provides a shortcut method for spawning the debugger's daemons in parallel with the job. This requires that the debugger be specified as one of the `pmix_app_t` in the same spawn command used to start the job. The debugger application must include at least the `PMIX_DEBUGGER_DAEMONS` attribute identifying itself as a debugger, and may utilize either a mapping option to direct daemon placement, or one of the `PMIX_DEBUG_DAEMONS_PER_PROC` or `PMIX_DEBUG_DAEMONS_PER_NODE` directives.

The launcher must not include information regarding the debugger daemons in the job-level info provided to the rest of the `pmix_app_t`s, nor in any calculated rank values (e.g., `PMIX_NODE_RANK` or `PMIX_LOCAL_RANK`) in those applications. The debugger job is to be assigned its own namespace and each debugger daemon shall receive a unique rank - i.e., the debugger application is to be treated as a completely separate PMIx job that is simply being started in parallel with the user's applications. The launcher is free to implement the launch as a single operation for both the applications and debugger daemons (preferred), or may stage the launches as required. The launcher shall not return from the `PMIx_Spawn` command until all included applications and the debugger daemons have been started.

Attributes that apply to both the debugger daemons and the application processes can be specified in the `job_info` array passed into the `PMIx_Spawn` API. Attributes that either (a) apply solely to the debugger daemons or to one of the applications included in the spawn request, or (b) have values that differ from those provided in the `job_info` array, should be specified in the `info` array in the corresponding `pmix_app_t`. Note that PMIx job `pause` attributes (e.g., `PMIX_DEBUG_STOP_IN_INIT`) do not apply to applications (defined in `pmix_app_t`) where the `PMIX_DEBUGGER_DAEMONS` attribute is set to `true`.

Debugger daemons spawned in this manner shall be provided with the typical PMIx information for their own job plus the target they are to debug via the `PMIX_DEBUG_TARGET` attribute. The debugger daemons spawned on a given node are responsible for self-determining their specific target process(es) - e.g., by referencing their own `PMIX_LOCAL_RANK` in the daemon debugger job versus the corresponding `PMIX_LOCAL_RANK` of the target processes on the node.

### Advice to users

Note that the tool calling `PMIx_Spawn` to request the launch of the debugger daemons is *not* included in the resulting job - i.e., the debugger daemons do not inherit the namespace of the tool. Thus, collective operations and notifications that target the debugger daemon job will not include the tool unless the namespace/rank of the tool is explicitly included.

The `PMIx_Spawn` API only supports the return of a single namespace resulting from the spawn request. In the case where the debugger job is co-spawned with the application, the spawn function shall return the namespace of the application and not the debugger job. Tools requiring access to the namespace of the debugger job must query the launcher for the spawned namespaces to find the one belonging to the debugger job.

## 1 17.4.3 Debugger Agents

2 Individual debuggers may, depending upon implementation, require varying degrees of control over  
3 each application process when it is started beyond those available via directives to **PMIx\_Spawn**.  
4 PMIx offers two mechanisms to help provide a means of meeting these needs.

5 The **PMIX\_FORKEXEC\_AGENT** attribute allows the debugger to specify an intermediate process  
6 (the Fork/Exec Agent (FEA)) for spawning the actual application process (see Fig. 17.7a), thereby  
7 interposing the debugger daemon between the application process and the launcher's daemon.  
8 Instead of spawning the application process, the launcher will spawn the FEA, which will connect  
9 back to the PMIx server as a tool to obtain the spawn description of the application process it is to  
10 spawn. The PMIx server in the launcher's daemon shall not register the fork/exec agent as a local  
11 client process, nor shall the launcher include the agent in any of the job-level values (e.g.,  
12 **PMIX\_RANK** within the job or **PMIX\_LOCAL\_RANK** on the node) provided to the application  
13 process. The launcher shall treat the collection of FEAs as a debugger job equivalent to the  
14 co-spawn use-case described in Section 17.4.2.

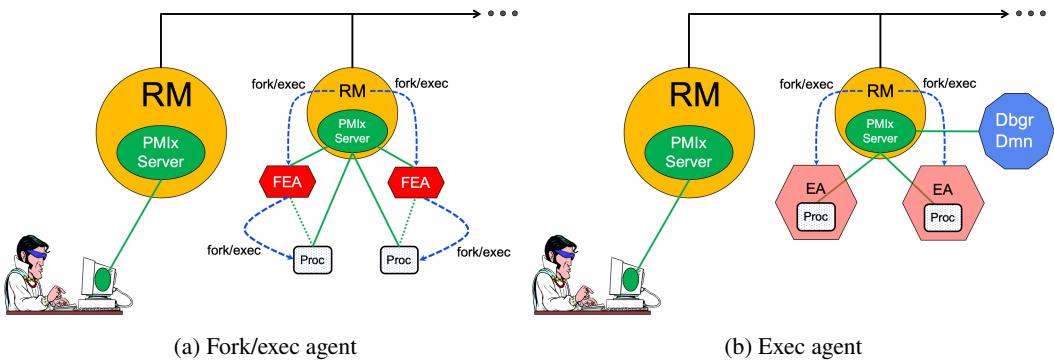


Figure 17.7.: Intermediary agents

15 In contrast, the **PMIX\_EXEC\_AGENT** attribute (Fig. 17.7b) allows the debugger to specify an agent  
16 that will perform some preparatory actions and then exec the eventual application process to replace  
17 itself. In this scenario, the exec agent is provided with the application process' command line as  
18 arguments on its command line (e.g., `./agent appargv[0] appargv[1]`) and does not  
19 connect back to the host's PMIx server. It is the responsibility of the exec agent to properly separate  
20 its own command line arguments (if any) from the application description.

## 17.4.4 Tracking the job lifecycle

There are a wide range of events a debugger can register to receive, but three are specifically defined for tracking a job's progress:

- **PMIX\_EVENT\_JOB\_START** indicates when the first process in the job has been spawned.
- **PMIX\_LAUNCH\_COMPLETE** indicates when the last process in the job has been spawned.
- **PMIX\_EVENT\_JOB\_END** indicates that all processes have terminated.

Each event is required to contain at least the namespace of the corresponding job and a **PMIX\_EVENT\_TIMESTAMP** indicating the time the event occurred. In addition, the **PMIX\_EVENT\_JOB\_END** event shall contain the returned status code (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job, plus the identity (**PMIX\_PROCID**) and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if applicable. Generation of these events by the launcher can be requested by including the **PMIX\_NOTIFY\_JOB\_EVENTS** attributes in the spawn request. Note that these events can be logged via the **PMIx\_Log** API by including the **PMIX\_LOG\_JOB\_EVENTS** attribute - this can be done either in conjunction with generated events, or in place of them.

Alternatively, if the debugger or tool solely wants to be alerted to job termination, then including the **PMIX\_NOTIFY\_COMPLETION** attribute in the spawn request would suffice. This attribute directs the launcher to provide just the **PMIX\_EVENT\_JOB\_END** event. Note that this event can be logged via the **PMIx\_Log** API by including the **PMIX\_LOG\_COMPLETION** attribute - this can be done either in conjunction with the generated event, or in place of it.

### Advice to users

The PMIx server is required to cache events in order to avoid race conditions - e.g., when a tool is trying to register for the **PMIX\_EVENT\_JOB\_END** event from a very short-lived job. Accordingly, registering for job-related events can result in receiving events relating to jobs other than the one of interest.

Users are therefore advised to specify the job whose events are of interest by including the **PMIX\_EVENT\_AFFECTED\_PROC** or **PMIX\_EVENT\_AFFECTED\_PROCS** attribute in the *info* array passed to the **PMIx\_Register\_event\_handler** API.

#### 17.4.4.1 Job lifecycle events

2       **PMIX\_EVENT\_JOB\_START**     The first process in the job has been spawned - includes  
3              **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_JOBID** and/or **PMIX\_NSPACE** of the job.  
4       **PMIX\_LAUNCH\_COMPLETE**     All processes in the job have been spawned - includes  
5              **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_JOBID** and/or **PMIX\_NSPACE** of the job.  
6       **PMIX\_EVENT\_JOB\_END**     All processes in the job have terminated - includes  
7              **PMIX\_EVENT\_TIMESTAMP** when the last process terminated as well as the **PMIX\_JOBID**  
8              and/or **PMIX\_NSPACE** of the job.  
9       **PMIX\_EVENT\_SESSION\_START**     The allocation has been instantiated and is ready for use -  
10          includes **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_SESSION\_ID** of the allocation.  
11          This event is issued after any system-controlled prologue has completed, but before any  
12          user-specified actions are taken.  
13       **PMIX\_EVENT\_SESSION\_END**     The allocation has terminated - includes  
14              **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_SESSION\_ID** of the allocation. This  
15          event is issued after any user-specified actions have completed, but before any  
16          system-controlled epilogue is performed.

17      The following events relate to processes within a job:

18       **PMIX\_EVENT\_PROC\_TERMINATED**     The specified process(es) terminated - normal or  
19          abnormal termination will be indicated by the **PMIX\_PROC\_TERM\_STATUS** in the *info*  
20          array of the notification. Note that a request for individual process events can generate a  
21          significant event volume from large-scale jobs.  
22       **PMIX\_ERR\_PROC\_TERM\_WO\_SYNC**     Process terminated without calling **PMIx\_Finalize**,  
23          or was a member of an assemblage formed via **PMIx\_Connect** and terminated or called  
24          **PMIx\_Finalize** without first calling **PMIx\_Disconnect** (or its non-blocking form)  
25          from that assemblage.

26      The following constants may be included via the **PMIX\_JOB\_TERM\_STATUS** attributed in the  
27          *info* array in the **PMIX\_EVENT\_JOB\_END** event notification to provide more detailed information  
28          regarding the reason for job abnormal termination:

29       **PMIX\_ERR\_JOB\_CANCELED**     The job was canceled by the host environment.  
30       **PMIX\_ERR\_JOB\_ABORTED**     One or more processes in the job called abort, causing the job to  
31          be terminated.  
32       **PMIX\_ERR\_JOB\_KILLED\_BY\_CMD**     The job was killed by user command.  
33       **PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG**     The job was aborted due to receipt of an error signal  
34          (e.g., SIGKILL).  
35       **PMIX\_ERR\_JOB\_TERM\_WO\_SYNC**     The job was terminated due to at least one process  
36          terminating without calling **PMIx\_Finalize**, or was a member of an assemblage formed  
37          via **PMIx\_Connect** and terminated or called **PMIx\_Finalize** without first calling  
38          **PMIx\_Disconnect** (or its non-blocking form) from that assemblage.  
39       **PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED**     The job was terminated due to one or more  
40          processes exceeding a specified sensor limit.

```
1 PMIX_ERR_JOB_NON_ZERO_TERM    The job was terminated due to one or more processes  
2      exiting with a non-zero status.  
3 PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT    The job was aborted due to receipt of a  
4      system event.
```

#### 5 17.4.4.2 Job lifecycle attributes

```
6 PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  
7      Status returned by job upon its termination. The status will be communicated as part of a  
8      PMIx event payload provided by the host environment upon termination of a job. Note that  
9      generation of the PMIX_EVENT_JOB_END event is optional and host environments may  
10     choose to provide it only upon request.  
11 PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  
12     State of the specified process as of the last report - may not be the actual current state based  
13     on update rate.  
14 PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  
15     Status returned by a process upon its termination. The status will be communicated as part  
16     of a PMIx event payload provided by the host environment upon termination of a process.  
17     Note that generation of the PMIX_EVENT_PROC_TERMINATED event is optional and host  
18     environments may choose to provide it only upon request.
```

#### 19 17.4.5 Debugger-related constants

```
20 The following constants are used in events used to coordinate applications and the debuggers  
21 attaching to them.
```

```
22 PMIX_DEBUG_WAITING_FOR_NOTIFY    All processes in the job to be debugged are paused  
23      waiting for a release at some point within the application. The application shall remain in a  
24      paused state awaiting release until receipt of the PMIX_DEBUGGER_RELEASE.  
25 PMIX_DEBUGGER_RELEASE    Release processes that are paused at the  
26      PMIX_DEBUG_WAIT_FOR_NOTIFY point in the target application.
```

#### 27 17.4.6 Debugger attributes

```
28 Attributes used to assist debuggers - these are values that can either be passed to the PMIx_Spawn  
29 APIs or accessed by a debugger itself using the PMIx_Get API with the  
30 PMIX_RANK_WILDCARD rank.
```

```
31 PMIX_DEBUG_STOP_ON_EXEC "pmix.dbg.exec" (bool)  
32      Included in either the pmix_info_t array in a pmix_app_t description (if the directive  
33      applies only to that application) or in the job_info array if it applies to all applications in the  
34      given spawn request. Indicates that the application is being spawned under a debugger, and  
35      that the local launch agent is to pause the resulting application processes on first instruction  
36      for debugger attach. The launcher (RM or IL) is to generate the  
37      PMIX_LAUNCH_COMPLETE event when all processes are stopped at the exec point.  
38 PMIX_DEBUG_STOP_IN_INIT "pmix.dbg.init" (bool)
```

1 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
 2 applies only to that application) or in the `job_info` array if it applies to all applications in the  
 3 given spawn request. Indicates that the specified application is being spawned under a  
 4 debugger. The PMIx client library in each resulting application process shall notify its PMIx  
 5 server that it is pausing and then pause during `PMIx_Init` of the spawned processes until  
 6 receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible  
 7 for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have  
 8 reached the pause point.  
 9 **`PMIX_DEBUG_WAIT_FOR_NOTIFY`** "pmix.dbg.notify" (bool)  
 10 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
 11 applies only to that application) or in the `job_info` array if it applies to all applications in the  
 12 given spawn request. Indicates that the specified application is being spawned under a  
 13 debugger. The resulting application processes are to notify their server (by generating the  
 14 `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some  
 15 application-determined location and pause at that point until receipt of the  
 16 `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating  
 17 the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they  
 18 are at the pause point.  
 19 **`PMIX_DEBUG_TARGET`** "pmix.dbg.tgt" (pmix\_proc\_t\*)  
 20 Identifier of process(es) to be debugged - a rank of `PMIX_RANK_WILDCARD` indicates that  
 21 all processes in the specified namespace are to be included.  
 22 **`PMIX_DEBUGGER_DAEMONS`** "pmix.debugger" (bool)  
 23 Included in the `pmix_info_t` array of a `pmix_app_t`, this attribute declares that the  
 24 application consists of debugger daemons and shall be governed accordingly. If used as the  
 25 sole `pmix_app_t` in a `PMIx_Spawn` request, then the `PMIX_DEBUG_TARGET` attribute  
 26 must also be provided (in either the `job_info` or in the `info` array of the `pmix_app_t`) to  
 27 identify the namespace to be debugged so that the launcher can determine where to place the  
 28 spawned daemons. If neither `PMIX_DEBUG_DAEMONS_PER_PROC` nor  
 29 `PMIX_DEBUG_DAEMONS_PER_NODE` is specified, then the launcher shall default to a  
 30 placement policy of one daemon per process in the target job.  
 31 **`PMIX_COSPAWN_APP`** "pmix.cospawn" (bool)  
 32 Designated application is to be spawned as a disconnected job - i.e., the launcher shall not  
 33 include the application in any of the job-level values (e.g., `PMIX_RANK` within the job)  
 34 provided to any other application process generated by the same spawn request. Typically  
 35 used to cospawn debugger daemons alongside an application.  
 36 **`PMIX_DEBUG_DAEMONS_PER_PROC`** "pmix.dbg.dpproc" (uint16\_t)  
 37 Number of debugger daemons to be spawned per application process. The launcher is to pass  
 38 the identifier of the namespace to be debugged by including the `PMIX_DEBUG_TARGET`  
 39 attribute in the daemon's job-level information. The debugger daemons spawned on a given  
 40 node are responsible for self-determining their specific target process(es) - e.g., by  
 41 referencing their own `PMIX_LOCAL_RANK` in the daemon debugger job versus the  
 42 corresponding `PMIX_LOCAL_RANK` of the target processes on the node.  
 43 **`PMIX_DEBUG_DAEMONS_PER_NODE`** "pmix.dbg.dpnd" (uint16\_t)

1 Number of debugger daemons to be spawned on each node where the target job is executing.  
 2 The launcher is to pass the identifier of the namespace to be debugged by including the  
 3 **PMIX\_DEBUG\_TARGET** attribute in the daemon's job-level information. The debugger  
 4 daemons spawned on a given node are responsible for self-determining their specific target  
 5 process(es) - e.g., by referencing their own **PMIX\_LOCAL\_RANK** in the daemon debugger  
 6 job versus the corresponding **PMIX\_LOCAL\_RANK** of the target processes on the node.  
 7 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.ptable" (char\*)  
 8 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
 9 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
 10 **PMIX\_NSPACE** indicating the namespace whose process table is being queried.  
 11 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lptable" (char\*)  
 12 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
 13 process in the specified namespace executing on the same node as the requester, ordered by  
 14 process job rank. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the namespace  
 15 whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME**  
 16 indicating the host whose local process table is being queried. By default, the query assumes  
 17 that the host upon which the request was made is to be used.

## 17.5 Tool-Specific APIs

19 PMIx-based tools automatically have access to all PMIx client functions. Tools designated as a  
 20 *launcher* or a *server* will also have access to all PMIx server functions. There are, however, an  
 21 additional set of functions (described in this section) that are specific to a PMIx tool. Access to  
 22 those functions require use of the tool initialization routine.

### 17.5.1 **PMIx\_tool\_init**

#### Summary

Initialize the PMIx library for operating as a tool, optionally connecting to a specified PMIx server.

#### Format

PMIx v2.0

C

```

27     pmix_status_t
28     PMIx_tool_init(pmix_proc_t *proc,
29                      pmix_info_t info[], size_t ninfo);

```

C

30 **INPUT proc**

31       **pmix\_proc\_t** structure (handle)

32 **IN info**

33       Array of **pmix\_info\_t** structures (array of handles)

34 **IN ninfo**

35       Number of elements in the *info* array (**size\_t**)

36 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

## Required Attributes

The following attributes are required to be supported by all PMIx libraries:

1           **PMIX\_TOOL\_NSPACE** "pmix.tool.nspace" (`char*`)  
2           Name of the namespace to use for this tool.  
3  
4           **PMIX\_TOOL\_RANK** "pmix.tool.rank" (`uint32_t`)  
5           Rank of this tool.  
6  
7           **PMIX\_TOOL\_DO\_NOT\_CONNECT** "pmix.tool.nocon" (`bool`)  
8           The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.  
9  
10          **PMIX\_TOOL\_ATTACHMENT\_FILE** "pmix.tool.attach" (`char*`)  
11          Pathname of file containing connection information to be used for attaching to a specific  
12          server.  
13  
14          **PMIX\_SERVER\_URI** "pmix.srvr.uri" (`char*`)  
15          URI of the PMIx server to be contacted.  
16  
17          **PMIX\_TCP\_URI** "pmix.tcp.uri" (`char*`)  
18          The URI of the PMIx server to connect to, or a file name containing it in the form of  
19          file:<name of file containing it>.  
20  
21          **PMIX\_SERVER\_PIDINFO** "pmix.srvr.pidinfo" (`pid_t`)  
22          PID of the target PMIx server for a tool.  
23  
24          **PMIX\_SERVER\_NSPACE** "pmix.srv.nspace" (`char*`)  
25          Name of the namespace to use for this PMIx server.  
26  
27          **PMIX\_CONNECT\_TO\_SYSTEM** "pmix.cnct.sys" (`bool`)  
28          The requester requires that a connection be made only to a local, system-level PMIx server.  
29  
30          **PMIX\_CONNECT\_SYSTEM\_FIRST** "pmix.cnct.sys.first" (`bool`)  
31          Preferentially, look for a system-level PMIx server first.

## Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

24  
25          **PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (`uint32_t`)  
26          Time in seconds between connection attempts to a PMIx server - the default value is  
27          implementation specific.  
28  
29          **PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (`uint32_t`)  
30          Maximum number of times to try to connect to PMIx server - the default value is  
31          implementation specific.  
32  
33          **PMIX\_SOCKET\_MODE** "pmix.sockmode" (`uint32_t`)

1       POSIX *mode\_t* (9 bits valid). If the library supports socket connections, this attribute may  
2       be supported for setting the socket mode.

3       **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (**char\***)

4       If provided, directs that the TCP URI be reported and indicates the desired method of  
5       reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket  
6       connections, this attribute may be supported for reporting the URI.

7       **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (**char\***)

8       Comma-delimited list of devices and/or CIDR notation to include when establishing the  
9       TCP connection. If the library supports TCP socket connections, this attribute may be  
10      supported for specifying the interfaces to be used.

11      **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (**char\***)

12      Comma-delimited list of devices and/or CIDR notation to exclude when establishing the  
13      TCP connection. If the library supports TCP socket connections, this attribute may be  
14      supported for specifying the interfaces that are *not* to be used.

15      **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (**int**)

16      The IPv4 port to be used.. If the library supports IPV4 connections, this attribute may be  
17      supported for specifying the port to be used.

18      **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (**int**)

19      The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be  
20      supported for specifying the port to be used.

21      **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (**bool**)

22      Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,  
23      this attribute may be supported for disabling it.

24      **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (**bool**)

25      Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,  
26      this attribute may be supported for disabling it.

27      **PMIX\_EVENT\_BASE** "pmix.evbase" (**struct event\_base \***)

28      Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.

---

<sup>1</sup><http://libevent.org/>

1           **Description**

2       Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided  
3       **pmix\_proc\_t** struct. The *info* array is used to pass user requests pertaining to the initialization  
4       and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives  
5       are desired.

6       If called with the **PMIX\_TOOL\_DO\_NOT\_CONNECT** attribute, the PMIx tool library will fully  
7       initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later  
8       point in time, if desired, by calling the **PMIx\_tool\_connect\_to\_server** function. If  
9       provided, the *proc* structure will be set to a zero-length namespace and a rank of  
10      **PMIX\_RANK\_UNDEF** unless the **PMIX\_TOOL\_NSPACE** and **PMIX\_TOOL\_RANK** attributes are  
11      included in the *info* array.

12      In all other cases, the PMIx tool library will automatically attempt to connect to a PMIx server  
13      according to the precedence chain described in Section 17.1. If successful, the function will return  
14      **PMIX\_SUCCESS** and will fill the process structure (if provided) with the assigned namespace and  
15      rank of the tool. The server to which the tool connects will be designated its *primary* server. Note  
16      that each connection attempt in the above precedence chain will retry (with delay between each  
17      retry) a number of times according to the values of the corresponding attributes.

18      Note that the PMIx tool library is referenced counted, and so multiple calls to **PMIx\_tool\_init**  
19      are allowed. If the tool is not connected to any server when this API is called, then the tool will  
20      attempt to connect to a server unless the **PMIX\_TOOL\_DO\_NOT\_CONNECT** is included in the call  
21      to API.

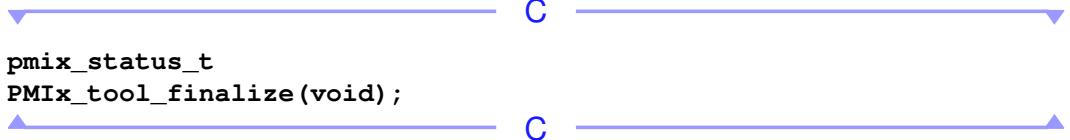
22     **17.5.2 PMIx\_tool\_finalize**

23       **Summary**

24       Finalize the PMIx tool library.

25       **Format**

PMIx v2.0

26         
27       **pmix\_status\_t**  
28       **PMIx\_tool\_finalize(void);**

29       Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

30       **Description**

31       Finalize the PMIx tool library, closing all existing connections to servers. An error code will be  
32       returned if, for some reason, a connection cannot be cleanly terminated — in such cases, the  
33       connection is dropped. Upon detecting loss of the connection, the PMIx server shall cleanup all  
      associated records of the tool.

1 **17.5.3 PMIx\_tool\_disconnect**

2 **Summary**

3 Disconnect the PMIx tool from the specified server connection while leaving the tool library  
4 initialized.

5 **Format**

6 *PMIx v4.0*

7 `pmix_status_t  
PMIx_tool_disconnect(pmix_proc_t *server);`

8 **IN server**  
9     `pmix_proc_t` structure (handle)

10 Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

11 **Description**

12 Close the current connection to the specified server, if one has been made, while leaving the PMIx  
13 library initialized. An error code will be returned if, for some reason, the connection cannot be  
14 cleanly terminated - in this case, the connection is dropped. In either case, the library will remain  
15 initialized. Upon detecting loss of the connection, the PMIx server shall cleanup all associated  
16 records of the tool.

17 Note that if the server being disconnected is the current *primary* server, then all operations  
18 requiring support from a server will return the `PMIX_ERR_UNREACH` error until the tool either  
19 designates an existing connection to be the *primary* server or, if no other connections exist, the tool  
20 establishes a connection to a PMIx server.

21 **17.5.4 PMIx\_tool\_attach\_to\_server**

22 **Summary**

23 Establish a connection to a PMIx server.

24 **Format**

25 *PMIx v4.0*

26     `pmix_status_t  
PMIx_tool_attach_to_server(pmix_proc_t *proc,  
                              pmix_proc_t *server,  
                              pmix_info_t info[],  
                             size_t ninfo);`

```

1 INOUT proc
2     Pointer to pmix_proc_t structure (handle)
3 INOUT server
4     Pointer to pmix_proc_t structure (handle)
5 IN info
6     Array of pmix_info_t structures (array of handles)
7 IN ninfo
8     Number of elements in the info array (size_t)
9 Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

### Required Attributes

The following attributes are required to be supported by all PMIx libraries:

```

11 PMIX_TOOL_ATTACHMENT_FILE "pmix.tool.attach" (char*)
12     Pathname of file containing connection information to be used for attaching to a specific
13     server.

14 PMIX_SERVER_URI "pmix.srvr.uri" (char*)
15     URI of the PMIx server to be contacted.

16 PMIX_TCP_URI "pmix.tcp.uri" (char*)
17     The URI of the PMIx server to connect to, or a file name containing it in the form of
18     file:<name of file containing it>.

19 PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
20     PID of the target PMIx server for a tool.

21 PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
22     Name of the namespace to use for this PMIx server.

23 PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)
24     The requester requires that a connection be made only to a local, system-level PMIx server.

25 PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool)
26     Preferentially, look for a system-level PMIx server first.

27 PMIX_PRIMARY_SERVER "pmix.pri.srvr" (bool)
28     The server to which the tool is connecting shall be designated the primary server once
29     connection has been accomplished.

```

1           **Description**

2     Establish a connection to a server. This function can be called at any time by a PMIx tool to create a  
3     new connection to a server. If a specific server is given and the tool is already attached to it, then  
4     the API shall return **PMIX\_SUCCESS** without taking any further action. In all other cases, the tool  
5     will attempt to discover a server using the method described in Section 17.1, ignoring all candidates  
6     to which it is already connected. The **PMIX\_ERR\_UNREACH** error shall be returned if no new  
7     connection is made.

8     The process identifier assigned to this tool is returned in the provided *proc* structure. Passing a  
9     value of **NULL** for the *proc* parameter is allowed if the user wishes solely to connect to a PMIx  
10    server and does not require return of the identifier at that time.

11    The process identifier of the server to which the tool attached is returned in the *server* structure.  
12    Passing a value of **NULL** for the *proc* parameter is allowed if the user wishes solely to connect to a  
13    PMIx server and does not require return of the identifier at that time.

14    Note that the **PMIX\_PRIMARY\_SERVER** attribute must be included in the *info* array if the server  
15    being connected to is to become the primary server, or a call to **PMIx\_tool\_set\_server** must  
16    be provided immediately after the call to this function.

17            **Advice to PMIx library implementers** 

18    When a tool connects to a server that is under a different namespace manager (e.g., host RM) from  
19    the prior server, the namespace in the identifier of the tool must remain unique in the new universe.  
20    If the namespace of the tool fails to meet this criteria in the new universe, then the new namespace  
   manager is required to return an error and the connection attempt must fail.

154 

21            **Advice to users** 

22    Some PMIx implementations may not support connecting to a server that is not under the same  
   namespace manager (e.g., host RM) as the server to which the tool is currently connected.

154 

23           **17.5.5 PMIx\_tool\_get\_servers**

24           **Summary**

25    Get an array containing the **pmix\_proc\_t** process identifiers of all servers to which the tool is  
26    currently connected.

1           **Format**

2        *pmix\_status\_t*  
3        *PMIx\_tool\_get\_servers*(*pmix\_proc\_t* \**servers*[], *size\_t* \**nservers*);

4           **OUT servers**

5           Address where the pointer to an array of *pmix\_proc\_t* structures shall be returned (handle)

6           **INOUT nservers**

7           Address where the number of elements in *servers* shall be returned (handle)

8           Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

9           **Description**

10          Return an array containing the *pmix\_proc\_t* process identifiers of all servers to which the tool is  
11          currently connected. The process identifier of the current primary server shall be the first entry in  
12          the array, with the remaining entries in order of attachment from earliest to most recent.

13 **17.5.6 PMIx\_tool\_set\_server**

14           **Summary**

15          Designate a server as the tool's *primary* server.

16           **Format**

17        *pmix\_status\_t*  
18        *PMIx\_tool\_set\_server*(*pmix\_proc\_t* \**server*);

19           **IN server**

20           *pmix\_proc\_t* structure (handle)

21           Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

22           **Description**

23          Designate the specified server to be the tool's *primary* server for all subsequent API calls.

24 **17.5.7 PMIx\_IOF\_pull**

25           **Summary**

26          Register to receive output forwarded from a set of remote processes.

1           **Format**

2        *PMIx v3.0*

C

```
3        pmix_status_t
4        PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,
5                     const pmix_info_t directives[], size_t ndirs,
6                     pmix_ifc_channel_t channel,
7                     pmix_ifc_cfunc_t cfunc,
8                     pmix_hdlr_reg_cfunc_t regcfunc,
9                     void *regcbdata);
```

C

9        **IN    procs**

10      Array of proc structures identifying desired source processes (array of handles)

11       **IN    nprocs**

12      Number of elements in the *procs* array (integer)

13       **IN    directives**

14      Array of [pmix\\_info\\_t](#) structures (array of handles)

15       **IN    ndirs**

16      Number of elements in the *directives* array (integer)

17       **IN    channel**

18      Bitmask of IO channels included in the request ([pmix\\_ifc\\_channel\\_t](#))

19       **IN    cfunc**

20      Callback function for delivering relevant output ([pmix\\_ifc\\_cfunc\\_t](#) function reference)

21       **IN    regcfunc**

22      Function to be called when registration is completed ([pmix\\_hdlr\\_reg\\_cfunc\\_t](#) function reference)

23       **IN    regcbdata**

24      Data to be passed to the *regcfunc* callback function (memory reference)

25        
26      Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant. In the event  
27      the function returns an error, the *regcfunc* will *not* be called.

28           **Required Attributes**

29      The following attributes are required for PMIx libraries that support IO forwarding:

30       **PMIX\_IOF\_CACHE\_SIZE "pmix.iof.cszie" (uint32\_t)**

31      The requested size of the PMIx server cache in bytes for each specified channel. By default,  
32      the server is allowed (but not required) to drop all bytes received beyond the max size.

33       **PMIX\_IOF\_DROP\_OLDEST "pmix.iof.old" (bool)**

34      In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
35      cache.

36       **PMIX\_IOF\_DROP\_NEWEST "pmix.iof.new" (bool)**

1 In an overflow situation, the PMIx server is to drop any new bytes received until room  
2 becomes available in the cache (default).

## Optional Attributes

3 The following attributes are optional for PMIx libraries that support IO forwarding:

4 **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)

5 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
6 specified number of bytes is collected to avoid being called every time a block of IO arrives.  
7 The PMIx tool library will execute the callback and reset the collection counter whenever the  
8 specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
9 the callback upon a call to deregister the respective channel.

10 **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)

11 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
12 size, this prevents IO from being held indefinitely while waiting for another payload to  
13 arrive.

14 **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (bool)

15 Requests that output be prefixed with the nspace,rank of the source and a string identifying  
16 the channel (**stdout**, **stderr**, etc.).

17 **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (bool)

18 Requests that output be marked with the time at which the data was received by the tool -  
19 note that this will differ from the time at which the data was collected from the source.

20 **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)

21 Requests that output be formatted in XML.

## Description

22 Register to receive output forwarded from a set of remote processes.

## Advice to users

24 Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated  
25 channels to be written to their corresponding **stdout/stderr** file descriptors. Use of  
26 **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but should be  
27 used carefully due to bandwidth and memory footprint considerations.

## 28 17.5.8 PMIx\_IOF\_deregister

### 29 Summary

30 Deregister from output forwarded from a set of remote processes.

1      **Format**

2      *PMIx v3.0*      C  
3      pmix\_status\_t  
4      PMIx\_IOF\_deregister(size\_t iofhdlr,  
5                        const pmix\_info\_t directives[], size\_t ndirs,  
                      pmix\_op\_cfunc\_t cfunc, void \*cbdata);

6      **IN iofhdlr**

7      Registration number returned from the `pmix_hdlr_reg_cfunc_t` callback from the  
8      call to `PMIx_IOF_pull(size_t)`

9      **IN directives**

10     Array of `pmix_info_t` structures (array of handles)

11     **IN ndirs**

12     Number of elements in the *directives* array (integer)

13     **IN cfunc**

14     Callback function to be called when deregistration has been completed. (function reference)

15     **IN cbdata**

16     Data to be passed to the *cfunc* callback function (memory reference)

17     Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cfunc* will *not* be called

25     **Description**

26     Deregister from output forwarded from a set of remote processes.

27     **Advice to PMIx library implementers**

28     Any currently buffered IO should be flushed upon receipt of a deregistration request. All received IO after receipt of the request shall be discarded.

29     **17.5.9 PMIx\_IOF\_push**

30     **Summary**

31     Push data collected locally (typically from `stdin` or a file) to `stdin` of the target recipients.

1           **Format**

2           `pmix\_status\_t  
3           PMIx\_IOF\_push(const pmix\_proc\_t targets[], size\_t ntargs,  
4                         pmix\_byte\_object\_t \*bo,  
5                         const pmix\_info\_t directives[], size\_t ndirs,  
6                         pmix\_op\_cbfunc\_t cbfunc, void \*cbdata);`

C

7           **IN    targets**

8           Array of proc structures identifying desired target processes (array of handles)

9           **IN    ntargs**

10          Number of elements in the *targets* array (integer)

11          **IN    bo**

12          Pointer to [pmix\\_byte\\_object\\_t](#) containing the payload to be delivered (handle)

13          **IN    directives**

14          Array of [pmix\\_info\\_t](#) structures (array of handles)

15          **IN    ndirs**

16          Number of elements in the *directives* array (integer)

17          **IN    directives**

18          Array of [pmix\\_info\\_t](#) structures (array of handles)

19          **IN    cbfunc**

20          Callback function to be called when operation has been completed. ([pmix\\_op\\_cbfunc\\_t](#)  
21                         function reference)

22          **IN    cbdata**

23          Data to be passed to the *cbfunc* callback function (memory reference)

24          Returns one of the following:

- [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
- [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called.
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called.

32          **Required Attributes**

33          The following attributes are required for PMIx libraries that support IO forwarding:

34          **PMIX\_IOF\_CACHE\_SIZE "pmix.iof.cszie" (uint32\_t)**

35          The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

```
1   PMIX_IOF_DROP_ODEST "pmix.iof.old" (bool)
2     In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the
3     cache.
4   PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)
5     In an overflow situation, the PMIx server is to drop any new bytes received until room
6     becomes available in the cache (default).
```

## Optional Attributes

The following attributes are optional for PMIx libraries that support IO forwarding:

```
8   PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)
9     Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the
10    specified number of bytes is collected to avoid being called every time a block of IO arrives.
11    The PMIx tool library will execute the callback and reset the collection counter whenever the
12    specified number of bytes becomes available. Any remaining buffered data will be flushed to
13    the callback upon a call to deregister the respective channel.
14   PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)
15     Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
16     size, this prevents IO from being held indefinitely while waiting for another payload to
17     arrive.
18   PMIX_IOF_PUSH_STDIN "pmix.iof.stdin" (bool)
19     Requests that the PMIx library collect the stdin of the requester and forward it to the
20     processes specified in the PMIX_IOF_push call. All collected data is sent to the same
21     targets until stdin is closed, or a subsequent call to PMIX_IOF_push is made that
22     includes the PMIX_IOF_COMPLETE attribute indicating that forwarding of stdin is to be
23     terminated.
```

1           **Description**  
2         Called either to:

- 3
  - 4           • push data collected by the caller themselves (typically from **stdin** or a file) to **stdin** of the  
5           target recipients;
  - 6           • request that the PMIx library automatically collect and push the **stdin** of the caller to the target  
7           recipients; or
  - 8           • indicate that automatic collection and transmittal of **stdin** is to stop

---

**Advice to users**

---

9         Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires  
10        the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets.  
11        Use of **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but  
          should be used carefully due to bandwidth and memory footprint considerations.

---

## APPENDIX A

# Python Bindings

---

1 While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of  
2 PMIx to that specific language. Support for other languages is captured in the Standard by  
3 describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This  
4 Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes.  
5 Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.

6 Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx  
7 “class” (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and  
8 therefore those have been included in the “client” class. Servers can utilize any of those functions  
9 plus a set focused on operations not commonly executed by an application process. Finally, tools  
10 can also act as servers but have their own initialization function.

## A.1 Design Considerations

Several issues arose during design of the Python bindings:

### A.1.1 Error Codes vs Python Exceptions

The C programming language reports errors through the return of the corresponding integer status codes. PMIx has defined a range of negative values for this purpose. However, Python has the option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and return PMIx status codes in lieu of raising exceptions as this method was considered more consistent for those working in both domains.

### A.1.2 Representation of Structured Data

PMIx utilizes a number of C-language structures to efficiently bundle related information. For example, the PMIx process identifier is represented as a struct containing a character array for the namespace and a 32-bit unsigned integer for the process rank. There are several options for translating such objects to Python – e.g., the PMIx process identifier could be represented as a two-element tuple (nspc, rank) or as a dictionary ‘nspc’: name, ‘rank’: 0. Exploration found no discernible benefit to either representation, nor was any clearly identifiable rationale developed that would lead a user to expect one versus the other for a given PMIx data type. Consistency in the translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion. Hence, the decision was made to express all complex datatypes as Python dictionaries.

## 1 A.2 Datatype Definitions

2 PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range  
3 integers (e.g., `uint32_t`), and structures. Each datatype is represented by a named unsigned 16-bit  
4 integer (`uint16_t`) constant. Users are advised to use the named PMIx constants for indicating  
5 datatypes instead of integer values to ensure compatibility with future PMIx versions.

6 With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 12 directly  
7 translate to Python. However, Python lacks the size-specific value definitions of C (e.g., `uint8_t`)  
8 and thus some care must be taken to protect against overflow/underflow situations when moving  
9 between the languages. Python bindings that accept values including PMIx datatypes shall  
10 therefore have the datatype and associated value checked for compatibility with their PMIx-defined  
11 equivalents, returning an error if:

- 12 • datatypes not defined by PMIx are encountered
- 13 • provided values fall outside the range of the C-equivalent definition - e.g., if a value identified as  
14 `PMIX_UINT8` lies outside the `uint8_t` range

15 Note that explicit labeling of PMIx data type, even when Python itself doesn't care, is often  
16 required for the Python bindings to know how to properly interpret and label the provided value  
17 when passing it to the PMIx library.

18 Table A.1 lists the correspondence between data types in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<b>bool</b>	PMIX_BOOL	boolean	
<b>byte</b>	PMIX_BYTE	A single element byte array (i.e., a byte array of length one)	
<b>char*</b>	PMIX_STRING	string	
<b>size_t</b>	PMIX_SIZE	integer	
<b>pid_t</b>	PMIX_PID	integer	value shall be limited to the <b>uint32_t</b> range
<b>int, int8_t, int16_t, int32_t, int64_t</b>	PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64	integer	value shall be limited to its corresponding range
<b>uint, uint8_t, uint16_t, uint32_t, uint64_t</b>	PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64	integer	value shall be limited to its corresponding range
<b>float, double</b>	PMIX_FLOAT, PMIX_DOUBLE	float	value shall be limited to its corresponding range
<b>struct timeval</b>	PMIX_TIMEVAL	{'sec': sec, 'usec': microsec}	each field is an integer value
<b>time_t</b>	PMIX_TIME	integer	limited to positive values
<b>pmix_data_type_t</b>	PMIX_DATA_TYPE	integer	value shall be limited to the <b>uint16_t</b> range
<b>pmix_status_t</b>	PMIX_STATUS	integer	
<b>pmix_key_t</b>	N/A	string	The string's length shall be limited to one less than the size of the <b>pmix_key_t</b> array (to reserve space for the terminating <b>NULL</b> )
<b>pmix_nspace_t</b>	N/A	string	The string's length shall be limited to one less than the size of the <b>pmix_nspace_t</b> array (to reserve space for the terminating <b>NULL</b> )

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<code>pmix_rank_t</code>	PMIX_PROC_RANK	integer	value shall be limited to the <code>uint32_t</code> range excepting the reserved values near <code>UINT32_MAX</code>
<code>pmix_proc_t</code>	PMIX_PROC	{'nspace': nspace, 'rank': rank}	<i>nspace</i> is a Python string and <i>rank</i> is an integer value. The <i>nspace</i> string's length shall be limited to one less than the size of the <code>pmix_nspace_t</code> array (to reserve space for the terminating <code>NULL</code> ), and the <i>rank</i> value shall conform to the constraints associated with <code>pmix_rank_t</code>
<code>pmix_byte_object_t</code>	PMIX_BYT_OBJECT	{'bytes': bytes, 'size': size}	<i>bytes</i> is a Python byte array and <i>size</i> is the integer number of bytes in that array.
<code>pmix_persistence_t</code>	PMIX_PERSISTENCE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_scope_t</code>	PMIX_SCOPE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_data_range_t</code>	PMIX_RANGE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_proc_state_t</code>	PMIX_PROC_STATE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_proc_info_t</code>	PMIX_PROC_INFO	{'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state}	<i>proc</i> is a Python <code>proc</code> dictionary; <i>hostname</i> and <i>executable</i> are Python strings; and <i>pid</i> , <i>exitcode</i> , and <i>state</i> are Python integers

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<code>pmix_data_array_t</code>	PMIX_DATA_ARRAY	{'type': type, 'array': array}	<i>type</i> is the PMIx type of object in the array and <i>array</i> is a Python <i>list</i> containing the individual array elements. Note that <i>array</i> can consist of <i>any</i> PMIx types, including (for example) a Python <code>info</code> object that itself contains an <code>array</code> value
<code>pmix_info_directives_t</code>	PMIX_INFO_DIRECTIVES	bitarray	32-bit array
<code>pmix_alloc_directive_t</code>	PMIX_ALLOC_DIRECTIVE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_iof_channel_t</code>	PMIX_IOF_CHANNEL	bitarray	16-bit array
<code>pmix_envar_t</code>	PMIX_ENVAR	{'envar': envar, 'value': value, 'separator': separator}	<i>envar</i> and <i>value</i> are Python strings, and <i>separator</i> a single-character Python string
<code>pmix_value_t</code>	PMIX_VALUE	{'value': value, 'val_type': type}	<i>type</i> is the PMIx datatype of <i>value</i> , and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype
<code>pmix_info_t</code>	PMIX_INFO	{'key': key, 'flags': flags, 'value': value, 'val_type': type}	<i>key</i> is a Python string <code>key</code> , <i>flags</i> is an <code>info directives</code> value, <i>type</i> is the PMIx datatype of <i>value</i> , and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype
<code>pmix_pdata_t</code>	PMIX_PDATA	{'proc': {'nspc': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type}	<i>proc</i> is a Python <code>proc</code> dictionary; <i>key</i> is a Python string <code>key</code> ; <i>type</i> is the PMIx datatype of <i>value</i> ; and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<code>pmix_app_t</code>	PMIX_APP	{'cmd': cmd, 'argv': [argv], 'env': [env], 'maxprocs': maxprocs, 'info': [info]}	<i>cmd</i> is a Python string; <i>argv</i> and <i>env</i> are Python lists containing Python strings; <i>maxprocs</i> is an integer; and <i>info</i> is a Python list of <code>info</code> values
<code>pmix_query_t</code>	PMIX_QUERY	{'keys': [keys], 'qualifiers': [info]}	<i>keys</i> is a Python list of Python strings, and <i>qualifiers</i> is a Python list of <code>info</code> values
<code>pmix_regattr_t</code>	PMIX_REGATTR	{'name': name, 'key': key, 'type': type, 'info': [info], 'description': [desc]}	<i>name</i> and <i>string</i> are Python strings; <i>type</i> is the PMIx datatype for the attribute's value; <i>info</i> is a Python list of <code>info</code> values; and <i>description</i> is a list of Python strings describing the attribute
<code>pmix_job_state_t</code>	PMIX_JOB_STATE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_link_state_t</code>	PMIX_LINK_STATE	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_cpuset_t</code>	N/A	{'source': source, 'cpus': bitmap}	<i>source</i> is a string name of the library that created the cpuset; and <i>cpus</i> is a bitarray containing the cpuset
<code>pmix_locality_t</code>	N/A	bitarray	16-bit array containing the relative locality of the specified local process
<code>pmix_fabric_t</code>	N/A	{'name': name, 'index': idx, 'info': [info]}	<i>name</i> is the string name assigned to the fabric; <i>index</i> is the integer ID assigned to the fabric; <i>info</i> is a list of <code>info</code> describing the fabric
<code>pmix_endpoint_t</code>	N/A	{'uuid': uuid, 'endpt': endpt}	<i>uuid</i> is the string system-unique identifier assigned to the device; <i>endpt</i> is a <code>byteobject</code> containing the endpoint information
<code>pmix_device_distance_t</code>	PMIX_DEVICE_DIST	{'uuid': uuid, 'mindist': mindist, 'maxdist': maxdist}	<i>uuid</i> is the string system-unique identifier assigned to the device; and <i>mindist</i> and <i>maxdist</i> are Python integers

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<code>pmix_topology_t</code>	N/A	{'name': name, 'index': idx, 'info': [info]}	<i>name</i> is the string name assigned to the fabric; <i>index</i> is the integer ID assigned to the fabric; <i>info</i> is a list of <code>info</code> describing the fabric
<code>pmix_coord_t</code>	PMIX_COORD	{'view': view, 'coord': [coords]}	<i>view</i> is the <code>pmix_coord_view_t</code> of the coordinate; and <i>coord</i> is a list of integer coordinates, one for each dimension of the fabric
<code>pmix_geometry_t</code>	PMIX_GEOMETRY	{'fabric': idx, 'uuid': uuid, 'coordinates': [coords]}	<i>fabric</i> is the Python integer index of the fabric; <i>uuid</i> is the string system-unique identifier assigned to the device; and <i>coordinates</i> is a list of <code>coord</code> containing the coordinates for the device across all views

## 1 A.2.1 Example

2 Converting a C-based program to its Python equivalent requires translation of the relevant  
3 datatypes as well as use of the appropriate API form. An example small program may help  
4 illustrate the changes. Consider the following C-based program snippet:

```
5 #include <pmix.h>
6 ...
7
8 pmix_info_t info[2];
9
10 PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "TEST", PMIX_STRING)
11 PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "PMIX", PMIX_STRING)
12
13 rc = PMIx_Init(&myproc, info, 2);
14
15 PMIX_INFO_DESTRUCT(&info[0]); // free the copied string
16 PMIX_INFO_DESTRUCT(&info[1]); // free the copied string
```

C

17 Moving to the Python version requires that the `pmix_info_t` be translated to the Python `info`  
18 equivalent, and that the returned information be captured in the return parameters as opposed to a  
19 pointer parameter in the function call, as shown below:

Python

```
20 import pmix
21 ...
22
23 myclient = PMIxClient()
24 info = [{'key':PMIX_PROGRAMMING_MODEL,
25           'value':'TEST', 'val_type'key':PMIX_MODEL_LIBRARY_NAME,
27           'value':'PMIX', 'val_type
```

Python

29 Note the use of the `PMIX_STRING` identifier to ensure the Python bindings interpret the provided  
30 string value as a PMIX "string" and not an array of bytes.

## 1 A.3 Callback Function Definitions

### 2 A.3.1 IOF Delivery Function

#### 3 Summary

4 Callback function for delivering forwarded IO to a process

#### 5 Format

PMIx v4.0

Python

```
6     def iofcfunc(iofhdlr:integer, channel:bitarray,
7                     source:dict, payload:dict, info:list)
```

Python

#### 8 IN iofhdlr

9 Registration number of the handler being invoked (integer)

#### 10 IN channel

11 Python `channel` 16-bit bitarray identifying the channel the data arrived on (bitarray)

#### 12 IN source

13 Python `proc` identifying the namespace/rank of the process that generated the data (dict)

#### 14 IN payload

15 Python `byteobject` containing the data (dict)

#### 16 IN info

17 List of Python `info` provided by the source containing metadata about the payload. This  
18 could include `PMIX_IOF_COMPLETE` (list)

19 Returns: nothing

20 See `pmix_iwf_cfunc_t` for details

## 21 A.3.2 Event Handler

#### 22 Summary

23 Callback function for event handlers

#### 24 Format

PMIx v4.0

## Python

```
1 def evhandler(evhdlr:integer, status:integer,
2                 source:dict, info:list, results:list)
3     Python
4
5     IN  iofhdlr
6         Registration number of the handler being invoked (integer)
7     IN  status
8         Status associated with the operation (integer)
9     IN  source
10        Python proc identifying the namespace/rank of the process that generated the event (dict)
11     IN  info
12        List of Python info provided by the source containing metadata about the event (list)
13     IN  results
14        List of Python info containing the aggregated results of all prior evhandlers (list)
15
16 Returns:
17
18     • rc - Status returned by the event handler's operation (integer)
19
20     • results - List of Python info containing results from this event handler's operation on the event
21         (list)
22
23 See pmix\_notification\_fn\_t for details
```

## A.3.3 Server Module Functions

The following definitions represent functions that may be provided to the PMIx server library at time of initialization for servicing of client requests. Module functions that are not provided default to returning "not supported" to the caller.

### A.3.3.1 Client Connected

#### Summary

Notify the host server that a client connected to this server.

#### Format

*PMIx v4.0*

Python

```
1 def clientconnected2(proc:dict is not None, info:list)
```

Python

2 IN proc

3 Python **proc** identifying the namespace/rank of the process that connected (dict)

4 IN info

5 list of Python **info** containing information about the process (list)

6 Returns:

7 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the connection should be rejected (integer)

8 See [\*\*pmix\\_server\\_client\\_connected2\\_fn\\_t\*\*](#) for details

### 9 A.3.3.2 Client Finalized

#### 10 Summary

11 Notify the host environment that a client called **PMIx\_Finalize**.

#### 12 Format

13 *PMIx v4.0*

Python

```
14 def clientfinalized(proc:dict is not None):
```

Python

15 IN proc

16 Python **proc** identifying the namespace/rank of the process that finalized (dict)

17 Returns: nothing

18 See [\*\*pmix\\_server\\_client\\_finalized\\_fn\\_t\*\*](#) for details

### 19 A.3.3.3 Client Aborted

#### 20 Summary

Notify the host environment that a local client called **PMIx\_Abort**.

```
1      Format  
2      PMIx v4.0          Python  
3      def clientaborted(args:dict is not None)  
4      Python  
5      IN   args  
6      Python dictionary containing:  
7          • 'caller': Python proc identifying the namespace/rank of the process calling abort (dict)  
8          • 'status': PMIx status to be returned on exit (integer)  
9          • 'msg': Optional string message to be printed (string)  
10         • 'targets': Optional list of Python proc identifying the namespace/rank of the processes to  
11           be aborted (list)  
12  
13     Returns:  
14         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
15     See pmix\_server\_abort\_fn\_t for details
```

### A.3.3.4 Fence

```
14    Summary  
15    At least one client called either PMIx_Fence or PMIx_Fence_nb  
16    Format  
17    PMIx v4.0          Python  
18    def fence(args:dict is not None)  
19    Python  
20  
21    IN   args  
22    Python dictionary containing:  
23        • 'procs': List of Python proc identifying the namespace/rank of the participating processes  
24          (list)  
25        • 'directives': Optional list of Python info containing directives controlling the operation  
26          (list)  
27        • 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)  
28  
29    Returns:  
30        • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
31        • data - Python bytearray containing the aggregated data from all participants (bytearray)  
32  
33    See pmix\_server\_fencenb\_fn\_t for details
```

### 1 A.3.3.5 Direct Modex

#### 2 Summary

3 Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
4 hosts the specified proc to obtain and return a direct modex blob for that proc.

#### 5 Format

6 *PMIx v4.0*

Python

```
def dmodex(args:dict is not None)
```

Python

#### 7 IN args

8 Python dictionary containing:

- 9 • 'proc': Python **proc** of process whose data is being requested (dict)
- 10 • 'directives': Optional list of Python **info** containing directives controlling the operation  
11 (list)

12 Returns:

- 13 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 14 • *data* - Python bytearray containing the data for the specified process (bytearray)

15 See [pmix\\_server\\_dmodex\\_req\\_fn\\_t](#) for details

### 16 A.3.3.6 Publish

#### 17 Summary

18 Publish data per the PMIx API specification.

#### 19 Format

20 *PMIx v4.0*

Python

```
def publish(args:dict is not None)
```

Python

#### 21 IN args

22 Python dictionary containing:

- 23 • 'proc': Python **proc** dictionary of process publishing the data (dict)
- 24 • 'directives': List of Python **info** containing data and directives (list)

25 Returns:

- 26 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 27 See [pmix\\_server\\_publish\\_fn\\_t](#) for details

### 1 A.3.3.7 Lookup

#### 2 Summary

3 Lookup published data.

#### 4 Format

5 *PMIx v4.0*

Python

```
def lookup(args:dict is not None)
```

Python

#### 6 IN args

7 Python dictionary containing:

- 8 • 'proc': Python **proc** of process seeking the data (dict)
- 9 • 'keys': List of Python strings (list)
- 10 • 'directives': Optional list of Python **info** containing directives (list)

11 Returns:

- 12 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 13 • *pdata* - List of **pdata** containing the returned results (list)

14 See [pmix\\_server\\_lookup\\_fn\\_t](#) for details

### 15 A.3.3.8 Unpublish

#### 16 Summary

17 Delete data from the data store.

#### 18 Format

19 *PMIx v4.0*

Python

```
def unpublish(args:dict is not None)
```

Python

#### 20 IN args

21 Python dictionary containing:

- 22 • 'proc': Python **proc** of process unpublishing data (dict)
- 23 • 'keys': List of Python strings (list)
- 24 • 'directives': Optional list of Python **info** containing directives (list)

25 Returns:

- 26 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 27 See [pmix\\_server\\_unpublish\\_fn\\_t](#) for details

1 **A.3.3.9 Spawn**

2      **Summary**

3      Spawn a set of applications/processes as per the [PMIx\\_Spawn](#) API.

4      **Format**

5      *PMIx v4.0*

```
def spawn(args:dict is not None)
```

Python

6      **IN args**

7      Python dictionary containing:

- 'proc': Python [proc](#) of process making the request (dict)
- 'jobinfo': Optional list of Python [info](#) job-level directives and information (list)
- 'apps': List of Python [app](#) describing applications to be spawned (list)

11     Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- *nspace* - Python string containing namespace of the spawned job (str)

14     See [pmix\\_server\\_spawn\\_fn\\_t](#) for details

15 **A.3.3.10 Connect**

16      **Summary**

17      Record the specified processes as *connected*.

18      **Format**

19      *PMIx v4.0*

```
def connect(args:dict is not None)
```

Python

20      **IN args**

21      Python dictionary containing:

- 'procs': List of Python [proc](#) identifying the namespace/rank of the participating processes (list)
- 'directives': Optional list of Python [info](#) containing directives controlling the operation (list)

26     Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- See [pmix\\_server\\_connect\\_fn\\_t](#) for details

### 1 A.3.3.11 Disconnect

#### 2 Summary

3 Disconnect a previously connected set of processes.

#### 4 Format

5 *PMIx v4.0*

```
def disconnect(args:dict is not None)
```

Python

Python

#### 6 IN args

7 Python dictionary containing:

- 8 • 'procs': List of Python **proc** identifying the namespace/rank of the participating processes  
9 (list)
- 10 • 'directives': Optional list of Python **info** containing directives controlling the operation  
11 (list)

12 Returns:

- 13 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

14 See [pmix\\_server\\_disconnect\\_fn\\_t](#) for details

### 15 A.3.3.12 Register Events

#### 16 Summary

17 Register to receive notifications for the specified events.

#### 18 Format

19 *PMIx v4.0*

```
def register_events(args:dict is not None)
```

Python

Python

#### 20 IN args

21 Python dictionary containing:

- 22 • 'codes': List of Python integers (list)
- 23 • 'directives': Optional list of Python **info** containing directives controlling the operation  
24 (list)

25 Returns:

- 26 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

27 See [pmix\\_server\\_register\\_events\\_fn\\_t](#) for details

### 1 A.3.3.13 Deregister Events

#### 2 Summary

3 Deregister to receive notifications for the specified events.

#### 4 Format

5 *PMIx v4.0*

Python

```
6     def deregister_events(args:dict is not None)
```

Python

#### 7 IN args

8 Python dictionary containing:

- 9 • 'codes': List of Python integers (list)

10 Returns:

- 11 • *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

12 See [pmix\\_server\\_deregister\\_events\\_fn\\_t](#) for details

### 13 A.3.3.14 Notify Event

#### 14 Summary

15 Notify the specified range of processes of an event.

#### 16 Format

17 *PMIx v4.0*

Python

```
18     def notify_event(args:dict is not None)
```

Python

#### 19 IN args

20 Python dictionary containing:

- 21 • 'code': Python integer [pmix\\_status\\_t](#) (integer)
- 22 • 'source': Python [proc](#) of process that generated the event (dict)
- 23 • 'range': Python [range](#) in which the event is to be reported (integer)
- 24 • 'directives': Optional list of Python [info](#) directives (list)

25 Returns:

- 26 • *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

27 See [pmix\\_server\\_notify\\_event\\_fn\\_t](#) for details

### 28 A.3.3.15 Query

#### Summary

Query information from the resource manager.

```
1      Format  
2      PMIx v4.0          Python  
3      def query(args:dict is not None)  
4      Python  
5      IN   args  
6          Python dictionary containing:  
7              • 'source': Python proc of requesting process (dict)  
8              • 'queries': List of Python query directives (list)  
9      Returns:  
10         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
11         • info - List of Python info containing the returned results (list)  
12     See pmix\_server\_query\_fn\_t for details
```

### A.3.3.16 Tool Connected

```
12    Summary  
13    Register that a tool has connected to the server.  
14    Format  
15    PMIx v4.0          Python  
16    def tool_connected(args:dict is not None)  
17    Python  
18    IN   args  
19        Python dictionary containing:  
20            • 'directives': Optional list of Python info info on the connecting tool (list)  
21    Returns:  
22        • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
23        • proc - Python proc containing the assigned namespace:rank for the tool (dict)  
24    See pmix\_server\_tool\_connection\_fn\_t for details
```

### A.3.3.17 Log

```
24    Summary  
25    Log data on behalf of a client.
```

```
1      Format  
2      PMIx v4.0  Python  
3      def log(args:dict is not None)  
4      Python  
5      IN  args  
6          Python dictionary containing:  
7              • 'source': Python proc of requesting process (dict)  
8              • 'data': Optional list of Python info containing data to be logged (list)  
9              • 'directives': Optional list of Python info containing directives (list)  
10         Returns:  
11             • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
12         See pmix\_server\_log\_fn\_t for details.
```

### A.3.3.18 Allocate Resources

```
12     Summary  
13     Request allocation operations on behalf of a client.  
14     Format  
15     PMIx v4.0  Python  
16     def allocate(args:dict is not None)  
17     Python  
18     IN  args  
19         Python dictionary containing:  
20             • 'source': Python proc of requesting process (dict)  
21             • 'action': Python allocdir specifying requested action (integer)  
22             • 'directives': Optional list of Python info containing directives (list)  
23         Returns:  
24             • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
25             • refarginfo - List of Python info containing results of requested operation (list)  
26         See pmix\_server\_alloc\_fn\_t for details.
```

### A.3.3.19 Job Control

```
26     Summary  
27     Execute a job control action on behalf of a client.
```

```
1      Format  
2      PMIx v4.0          Python  
3      def job_control(args:dict is not None)  
4      Python  
5      IN   args  
6      Python dictionary containing:  
7          • 'source': Python proc of requesting process (dict)  
8          • 'targets': List of Python proc specifying target processes (list)  
9          • 'directives': Optional list of Python info containing directives (list)  
10     Returns:  
11        • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
12     See pmix\_server\_job\_control\_fn\_t for details.
```

### A.3.3.20 Monitor

```
12    Summary  
13    Request that a client be monitored for activity.  
14    Format  
15    PMIx v4.0          Python  
16    def monitor(args:dict is not None)  
17    Python  
18    IN   args  
19    Python dictionary containing:  
20        • 'source': Python proc of requesting process (dict)  
21        • 'monitor': Python info attribute indicating the type of monitor being requested (dict)  
22        • 'error': Status code to be used when generating an event notification (integer) alerting that  
23          the monitor has been triggered.  
24        • 'directives': Optional list of Python info containing directives (list)  
25    Returns:  
26        • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
27    See pmix\_server\_monitor\_fn\_t for details.
```

### A.3.3.21 Get Credential

```
27    Summary  
28    Request a credential from the host environment.
```

```
1      Format  
2      PMIx v4.0          Python  
3      def get_credential(args:dict is not None)  
4      Python  
5      IN   args  
6      Python dictionary containing:  
7          • 'source': Python proc of requesting process (dict)  
8          • 'directives': Optional list of Python info containing directives (list)  
9      Returns:  
10     • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
11     • cred - Python byteobject containing returned credential (dict)  
12     • info - List of Python info containing any additional info about the credential (list)  
13     See pmix\_server\_get\_cred\_fn\_t for details.
```

### A.3.3.22 Validate Credential

```
13    Summary  
14    Request validation of a credential  
15    Format  
16    PMIx v4.0          Python  
17    def validate_credential(args:dict is not None)  
18    Python  
19    IN   args  
20    Python dictionary containing:  
21        • 'source': Python proc of requesting process (dict)  
22        • 'credential': Python byteobject containing credential (dict)  
23        • 'directives': Optional list of Python info containing directives (list)  
24    Returns:  
25    • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
26    • info - List of Python info containing any additional info from the credential (list)  
27    See pmix\_server\_validate\_cred\_fn\_t for details.
```

1 **A.3.3.23 IO Forward**

2      **Summary**

3      Request the specified IO channels be forwarded from the given array of processes.

4      **Format**

5      *PMIx v4.0*

6      def iof\_pull(args:dict is not None)

Python

Python

7      **IN args**

8      Python dictionary containing:

- 'sources': List of Python **proc** of processes whose IO is being requested (list)
- 'channels': Bitmask of Python **channel** identifying IO channels to be forwarded (integer)
- 'directives': Optional list of Python **info** containing directives (list)

9      Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

10     See [pmix\\_server\\_iоф\\_fn\\_t](#) for details.

11     **A.3.3.24 IO Push**

12      **Summary**

13      Pass standard input data to the host environment for transmission to specified recipients.

14      **Format**

15      *PMIx v4.0*

16      def iof\_push(args:dict is not None)

Python

Python

17      **IN args**

18      Python dictionary containing:

- 'source': Python **proc** of process whose input is being forwarded (dict)
- 'payload': Python **byteobject** containing input bytes (dict)
- 'targets': List of **proc** of processes that are to receive the payload (list)
- 'directives': Optional list of Python **info** containing directives (list)

20      Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

21     See [pmix\\_server\\_stdin\\_fn\\_t](#) for details.

### 1 A.3.3.25 Group Operations

#### 2 Summary

3 Request group operations (construct, destruct, etc.) on behalf of a set of processes.

#### 4 Format

5 *PMIx v4.0*

```
def group(args:dict is not None)
```

Python

Python

#### 6 IN args

7 Python dictionary containing:

- 8 • 'op': Operation host is to perform on the specified group (integer)
- 9 • 'group': String identifier of target group (str)
- 10 • 'procs': List of Python **proc** of participating processes (dict)
- 11 • 'directives': Optional list of Python **info** containing directives (list)

12 Returns:

- 13 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 14 • *refarginfo* - List of Python **info** containing results of requested operation (list)

15 See [pmix\\_server\\_grp\\_fn\\_t](#) for details.

### 16 A.3.3.26 Fabric Operations

#### 17 Summary

18 Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

#### 19 Format

20 *PMIx v4.0*

```
def fabric(args:dict is not None)
```

Python

Python

#### 21 IN args

22 Python dictionary containing:

- 23 • 'source': Python **proc** of requesting process (dict)
- 24 • 'index': Identifier of the fabric being operated upon (integer)
- 25 • 'op': Operation host is to perform on the specified fabric (integer)
- 26 • 'directives': Optional list of Python **info** containing directives (list)

27 Returns:

- 1     • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)  
2  
3     • *refarginfo* - List of Python **info** containing results of requested operation (list)  
See **pmix\_server\_fabric\_fn\_t** for details.

## 4     **A.4 PMIxClient**

5     The client Python class is by far the richest in terms of APIs as it houses all the APIs that an  
6     application might utilize. Due to the datatype translation requirements of the C-Python interface,  
7     only the blocking form of each API is supported – providing a Python callback function directly to  
8     the C interface underlying the bindings was not a supportable option.

### 9     **A.4.1 Client.init**

#### 10    **Summary**

11    Initialize the PMIx client library after obtaining a new PMIxClient object.

#### 12    **Format**

13    *PMIx v4.0*

14    *rc, proc = myclient.init(info:list)*

15    *IN info*

16    List of Python **info** dictionaries (list)

17    Returns:

- 18     • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
19     • *proc* - a Python **proc** dictionary (dict)

20    See **PMIx\_Init** for description of all relevant attributes and behaviors.

### 21    **A.4.2 Client.initialized**

#### 22    **Format**

23    *PMIx v4.0*

24    *rc = myclient.initialized()*

25    Returns:

- 26     • *rc* - a value of **1** (true) will be returned if the PMIx library has been initialized, and **0** (false)  
otherwise (integer)

27    See **PMIx\_Initialized** for description of all relevant attributes and behaviors.

1 **A.4.3 Client.get\_version**

2       **Format**

PMIx v4.0

3       `vers = myclient.get_version()`

Python

Python

4       Returns:

- 5       • `vers` - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)

6       See [PMIx\\_Get\\_version](#) for description of all relevant attributes and behaviors.

7 **A.4.4 Client.finalize**

8       **Summary**

9       Finalize the PMIx client library.

10      **Format**

PMIx v4.0

11      `rc = myclient.finalize(info:list)`

Python

Python

12      **IN info**

13       List of Python `info` dictionaries (list)

14       Returns:

- 15       • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

16       See [PMIx\\_Finalize](#) for description of all relevant attributes and behaviors.

17 **A.4.5 Client.abort**

18      **Summary**

19       Request that the provided list of processes be aborted.

1      **Format**

Python

2      `rc = myclient.abort(status:integer, msg:str, targets:list)`

Python

3      **IN status**

PMIx status to be returned on exit (integer)

4      **IN msg**

String message to be printed (string)

5      **IN targets**

List of Python `proc` dictionaries (list)

6      Returns:

- 7      • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

8      See [PMIx\\_Abort](#) for description of all relevant attributes and behaviors.

9      **A.4.6 Client.store\_internal**

10     **Summary**

11     Store some data locally for retrieval by other areas of the process

12     **Format**

Python

13     `rc = myclient.store_internal(proc:dict, key:str, value:dict)`

Python

14     **IN proc**

Python `proc` dictionary of the process being referenced (dict)

15     **IN key**

String key of the data (string)

16     **IN value**

Python `value` dictionary (dict)

17     Returns:

- 18     • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

19     See [PMIx\\_Store\\_internal](#) for details.

20     **A.4.7 Client.put**

21     **Summary**

22     Push a key/value pair into the client's namespace.

1           **Format**  
2        *PMIx v4.0*      Python  
3        *rc = myclient.put(scope:integer, key:str, value:dict)*  
4            Python  
5        IN **scope**  
6           Scope of the data being posted (integer)  
7        IN **key**  
8           String key of the data (string)  
9        IN **value**  
10          Python **value** dictionary (dict)  
11        Returns:  
12          

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

  
13        See **PMIx\_Put** for description of all relevant attributes and behaviors.

## 12 A.4.8 Client.commit

13           **Summary**  
14        Push all previously **PMIxClient.put** values to the local PMIx server.  
15           **Format**  
16        *PMIx v4.0*      Python  
17        *rc = myclient.commit()*  
18            Python  
19        Returns:  
20          

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

  
21        See **PMIx\_Commit** for description of all relevant attributes and behaviors.

## 20 A.4.9 Client.fence

21           **Summary**  
22        Execute a blocking barrier across the processes identified in the specified list.

```
1      Format  
PMIx v4.0   Python  
2      rc = myclient.fence(peers:list, directives:list)  
          Python  
3      IN  peers  
4          List of Python proc dictionaries (list)  
5      IN  directives  
6          List of Python info dictionaries (list)  
7      Returns:  
8          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9          See PMIx\_Fence for description of all relevant attributes and behaviors.
```

## 10 A.4.10 Client.get

```
11     Summary  
12     Retrieve a key/value pair.
```

```
13     Format  
PMIx v4.0   Python  
14    rc, val = myclient.get(proc:dict, key:str, directives:list)  
          Python  
15    IN  proc  
16        Python proc whose data is being requested (dict)  
17    IN  key  
18        Python string key of the data to be returned (str)  
19    IN  directives  
20        List of Python info dictionaries (list)  
21    Returns:  
22        • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
23        • val - Python value containing the returned data (dict)  
24    See PMIx\_Get for description of all relevant attributes and behaviors.
```

## 25 A.4.11 Client.publish

```
26     Summary  
27     Publish data for later access via PMIx\_Lookup.
```

```
1     Format  
2     rc = myclient.publish(directives:list)  
3     IN  directives  
4         List of Python info dictionaries containing data to be published and directives (list)  
5     Returns:  
6         • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
7     See PMIx\_Publish for description of all relevant attributes and behaviors.
```

## 8 A.4.12 Client.lookup

```
9     Summary  
10    Lookup information published by this or another process with PMIx\_Publish.  
11     Format  
12     rc,info = myclient.lookup(pdata:list, directives:list)  
13     IN  pdata  
14         List of Python pdata dictionaries identifying data to be retrieved (list)  
15     IN  directives  
16         List of Python info dictionaries (list)  
17     Returns:  
18         • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
19         • info - Python list of info containing the returned data (list)  
20     See PMIx\_Lookup for description of all relevant attributes and behaviors.
```

## 21 A.4.13 Client.unpublish

```
22     Summary  
23     Delete data published by this process with PMIx\_Publish.
```

```
1      Format  
2      rc = myclient.unpublish(keys:list, directives:list)  
3      IN  keys  
4          List of Python string keys identifying data to be deleted (list)  
5      IN  directives  
6          List of Python info dictionaries (list)  
7      Returns:  
8          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9      See PMIx_Unpublish for description of all relevant attributes and behaviors.
```

## 10 A.4.14 Client.spawn

```
11     Summary  
12     Spawns a new job.
```

```
13     Format  
14     rc, nspace = myclient.spawn(jobinfo:list, apps:list)  
15     IN  jobinfo  
16         List of Python info dictionaries (list)  
17     IN  apps  
18         List of Python app dictionaries (list)  
19     Returns:  
20         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
21         • nspace - Python nspace of the new job (dict)  
22     See PMIx_Spawn for description of all relevant attributes and behaviors.
```

## 23 A.4.15 Client.connect

```
24     Summary  
25     Connect namespaces.
```

1           **Format**  
2     `rc = myclient.connect(peers:list, directives:list)`  
3     IN **peers**  
4       List of Python **proc** dictionaries (list)  
5     IN **directives**  
6       List of Python **info** dictionaries (list)  
7     Returns:  
8       • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
9     See [PMIx\\_Connect](#) for description of all relevant attributes and behaviors.

## 10 A.4.16 Client.disconnect

11           **Summary**  
12     Disconnect namespaces.  
13           **Format**  
14     `rc = myclient.disconnect(peers:list, directives:list)`  
15     IN **peers**  
16       List of Python **proc** dictionaries (list)  
17     IN **directives**  
18       List of Python **info** dictionaries (list)  
19     Returns:  
20       • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
21     See [PMIx\\_Disconnect](#) for description of all relevant attributes and behaviors.

## 22 A.4.17 Client.resolve\_peers

23           **Summary**  
24     Return list of processes within the specified **nspc** on the given node.

1                   **Format**

2     `rc, procs = myclient.resolve_peers(node:str, nspace:str)`

3     **IN node**  
4       Name of node whose processes are being requested (str)  
5     **IN nspace**  
6       Python **nspace** whose processes are to be returned (str)  
7     Returns:  
8       • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
9       • *procs* - List of Python **proc** dictionaries (list)  
10      See [PMIx\\_Resolve\\_peers](#) for description of all relevant attributes and behaviors.

## 11 A.4.18 Client.resolve\_nodes

12                   **Summary**  
13      Return list of nodes hosting processes within the specified **nspace**.

14                   **Format**

15     `rc, nodes = myclient.resolve_nodes(nspace:str)`

16     **IN nspace**  
17       Python **nspace** (str)  
18     Returns:  
19       • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
20       • *nodes* - List of Python string node names (list)  
21      See [PMIx\\_Resolve\\_nodes](#) for description of all relevant attributes and behaviors.

## 22 A.4.19 Client.query

23                   **Summary**  
24      Query information about the system in general.

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myclient.query(queries:list)  
4      Python          Python  
5      IN   queries  
6      List of Python query dictionaries (list)  
7      Returns:  
8      • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9      • info - List of Python info containing results of the query (list)  
10     See PMIx_Query_info_nb for description of all relevant attributes and behaviors.
```

## 9 A.4.20 Client.log

```
10     Summary  
11     Log data to a central data service/store.  
12     Format  
13     PMIx v4.0          Python  
14     rc = myclient.log(data:list, directives:list)  
15     Python          Python  
16     IN   data  
17     List of Python info (list)  
18     IN   directives  
19     Optional list of Python info (list)  
20     Returns:  
21     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22     See PMIx_Log for description of all relevant attributes and behaviors.
```

## 21 A.4.21 Client.allocate

```
22     Summary  
23     Request an allocation operation from the host resource manager.
```

1           **Format**  
2        *rc, info = myclient.allocate(request:integer, directives:list)*  
3        IN **request**  
4           Python **allocodir** specifying requested operation (integer)  
5        IN **directives**  
6           List of Python **info** describing request (list)  
7        Returns:  
8           • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
9           • *info* - List of Python **info** containing results of the request (list)  
10      See [PMIx\\_Allocation\\_request\\_nb](#) for description of all relevant attributes and behaviors.

## 11 A.4.22 Client.job\_ctrl

12           **Summary**  
13      Request a job control action.  
14           **Format**  
15        *rc, info = myclient.job\_ctrl(targets:list, directives:list)*  
16        IN **targets**  
17           List of Python **proc** specifying targets of requested operation (integer)  
18        IN **directives**  
19           List of Python **info** describing operation to be performed (list)  
20        Returns:  
21           • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
22           • *info* - List of Python **info** containing results of the request (list)  
23      See [PMIx\\_Job\\_control\\_nb](#) for description of all relevant attributes and behaviors.

## 24 A.4.23 Client.monitor

25           **Summary**  
26      Request that something be monitored.

```
1      Format  
2      PMIx v4.0   Python  
3      rc,info = myclient.monitor(monitor:dict, error_code:integer, directives:list  
4      Python   Python  
5      IN  monitor  
6          Python info specifying specifying the type of monitor being requested (dict)  
7      IN  error_code  
8          Status code to be used when generating an event notification alerting that the monitor has  
9          been triggered (integer)  
10     IN  directives  
11         List of Python info describing request (list)  
12  
13     Returns:  
14         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
15         • info - List of Python info containing results of the request (list)  
16  
17     See PMIx_Process_monitor_nb for description of all relevant attributes and behaviors.
```

## 14 A.4.24 Client.get\_credential

```
15     Summary  
16     Request a credential from the PMIx server/SMS.  
17     Format  
18     PMIx v4.0   Python  
19     rc,cred = myclient.get_credential(directives:list)  
20     Python   Python  
21     IN  directives  
22         Optional list of Python info describing request (list)  
23  
24     Returns:  
25         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
26         • cred - Python byteobject containing returned credential (dict)  
27  
28     See PMIx_Get_credential for description of all relevant attributes and behaviors.
```

## 25 A.4.25 Client.validate\_credential

```
26     Summary  
27     Request validation of a credential by the PMIx server/SMS.
```

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myclient.validate_credential(cred:dict, directives:list)  
4      Python  
5      IN  cred  
6          Python byteobject containing credential (dict)  
7      IN  directives  
8          Optional list of Python info describing request (list)  
9      Returns:  
10     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
11     • info - List of Python info containing additional results of the request (list)  
12     See PMIx_Validate_credential for description of all relevant attributes and behaviors.
```

## 11 A.4.26 Client.group\_construct

```
12    Summary  
13    Construct a new group composed of the specified processes and identified with the provided group  
14    identifier.  
15    Format  
16    PMIx v4.0          Python  
17    rc,info = myclient.construct_group(grp:string,  
18    members:list, directives:list)  
19    Python  
20    IN  grp  
21        Python string identifier for the group (str)  
22    IN  members  
23        List of Python proc dictionaries identifying group members (list)  
24    IN  directives  
25        Optional list of Python info describing request (list)  
26    Returns:  
27    • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
28    • info - List of Python info containing results of the request (list)  
29    See PMIx_Group_construct for description of all relevant attributes and behaviors.
```

## 1 A.4.27 Client.group\_invite

### 2 Summary

3 Explicitly invite specified processes to join a group.

### 4 Format

5 *PMIx v4.0*

Python

```
6     rc,info = myclient.group_invite(grp:string,
7                                     members:list, directives:list)
```

Python

#### 7 IN grp

8 Python string identifier for the group (str)

#### 9 IN members

10 List of Python **proc** dictionaries identifying processes to be invited (list)

#### 11 IN directives

12 Optional list of Python **info** describing request (list)

13 Returns:

- 14 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 15 • *info* - List of Python **info** containing results of the request (list)

16 See **PMIx\_Group\_invite** for description of all relevant attributes and behaviors.

## 17 A.4.28 Client.group\_join

### 18 Summary

19 Respond to an invitation to join a group that is being asynchronously constructed.

### 20 Format

21 *PMIx v4.0*

Python

```
22     rc,info = myclient.group_join(grp:string,
23                                     leader:dict, opt:integer,
24                                     directives:list)
```

Python

#### 25 IN grp

26 Python string identifier for the group (str)

#### 27 IN leader

28 Python **proc** dictionary identifying process leading the group (dict)

#### 29 IN opt

30 One of the **pmix\_group\_opt\_t** values indicating decline/accept (integer)

#### 31 IN directives

Optional list of Python **info** describing request (list)

1        Returns:

2        • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

3        • *info* - List of Python **info** containing results of the request (list)

4        See **PMIx\_Group\_join** for description of all relevant attributes and behaviors.

## 5     A.4.29 Client.group\_leave

### 6        Summary

7        Leave a PMIx Group.

### 8        Format

PMIx v4.0

Python

9        `rc = myclient.group_leave(grp:string, directives:list)`

Python

10      IN **grp**

11        Python string identifier for the group (str)

12      IN **directives**

13        Optional list of Python **info** describing request (list)

14        Returns:

15        • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

16        See **PMIx\_Group\_leave** for description of all relevant attributes and behaviors.

## 17    A.4.30 Client.group\_destruct

### 18        Summary

19        Destruct a PMIx Group.

### 20        Format

PMIx v4.0

Python

21        `rc = myclient.group_destruct(grp:string, directives:list)`

Python

22      IN **grp**

23        Python string identifier for the group (str)

24      IN **directives**

25        Optional list of Python **info** describing request (list)

26        Returns:

27        • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

28        See **PMIx\_Group\_destruct** for description of all relevant attributes and behaviors.

## 1 A.4.31 Client.register\_event\_handler

### 2 Summary

3 Register an event handler to report events.

### 4 Format

5 *PMIx v4.0*

Python

```
6     rc,id = myclient.register_event_handler(codes:list,
7                                     directives:list, cbfunc)
```

Python

#### 7 IN codes

8 List of Python integer status codes that should be reported to this handler (list)

#### 9 IN directives

10 Optional list of Python `info` describing request (list)

#### 11 IN cbfunc

12 Python `evhandler` to be called when event is received (func)

13 Returns:

- 14 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- 15 • `id` - PMIx reference identifier for handler (integer)

16 See [PMIx\\_Register\\_event\\_handler](#) for description of all relevant attributes and behaviors.

## 17 A.4.32 Client.deregister\_event\_handler

### 18 Summary

19 Deregister an event handler.

### 20 Format

21 *PMIx v4.0*

Python

```
22     myclient.deregister_event_handler(id:integer)
```

Python

#### 23 IN id

24 PMIx reference identifier for handler (integer)

25 Returns: None

26 See [PMIx\\_Deregister\\_event\\_handler](#) for description of all relevant attributes and behaviors.

## 1 A.4.33 Client.notify\_event

### 2 Summary

3 Report an event for notification via any registered handler.

### 4 Format

5 PMIx v4.0

Python

```
6     rc = myclient.notify_event(status:integer, source:dict,
7                                   range:integer, directives:list)
```

Python

#### 7 IN status

8 PMIx status code indicating the event being reported (integer)

#### 9 IN source

10 Python **proc** of the process that generated the event (dict)

#### 11 IN range

12 Python **range** in which the event is to be reported (integer)

#### 13 IN directives

14 Optional list of Python **info** dictionaries describing the event (list)

15 Returns:

- 16 • **rc** - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

17 See [PMIx\\_Notify\\_event](#) for description of all relevant attributes and behaviors.

## 18 A.4.34 Client.fabric\_register

### 19 Summary

20 Register for access to fabric-related information, including communication cost matrix.

### 21 Format

22 PMIx v4.0

Python

```
23     rc, idx, fabricinfo = myclient.fabric_register(directives:list)
```

Python

#### 24 IN directives

25 Optional list of Python **info** containing directives (list)

26 Returns:

- 27 • **rc** - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 28 • **idx** - Index of the registered fabric (integer)
- 29 • **fabricinfo** - List of Python **info** containing fabric info (list)

See [PMIx\\_Fabric\\_register](#) for details.

1 **A.4.35 Client.fabric\_update**

2      **Summary**

3      Update fabric-related information, including communication cost matrix.

4      **Format**

5      *PMIx v4.0*

Python

6      `rc, fabricinfo = myclient.fabric_update(idx:integer)`

Python

7      **IN    idx**

8           Index of the registered fabric (list)

9      Returns:

- 10     • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

- 11     • `fabricinfo` - List of Python [info](#) containing updated fabric info (list)

12     See [PMIx\\_Fabric\\_update](#) for details.

13    **A.4.36 Client.fabric\_deregister**

14      **Summary**

15      Deregister fabric.

16      **Format**

17      *PMIx v4.0*

Python

18      `rc = myclient.fabric_deregister(idx:integer)`

Python

19      **IN    idx**

20           Index of the registered fabric (list)

21      Returns:

- 22     • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

23     See [PMIx\\_Fabric\\_deregister](#) for details.

24    **A.4.37 Client.load\_topology**

25      **Summary**

26      Load the local hardware topology into the PMIx library.

```
1      Format  
2      PMIx v4.0   Python  
3      rc = myclient.load_topology()  
4      Python  
5      Returns:  
6      • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
7      See PMIx\_Load\_topology for details - note that the topology loaded into the PMIx library may  
8      be utilized by PMIx and other libraries, but is not accessible by Python.
```

## 7 A.4.38 Client.get\_relative\_locality

```
8      Summary  
9      Get the relative locality of two local processes.  
10     Format  
11     PMIx v4.0   Python  
12     rc,locality = myclient.get_relative_locality(loc1:str, loc2:str)  
13     Python  
14     IN  loc1  
15       Locality string of a process (str)  
16     IN  loc2  
17       Locality string of a process (str)  
18     Returns:  
19     • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
20     • locality - locality bitarray containing the relative locality of the two processes (bitarray)  
21     See PMIx\_Get\_relative\_locality for details.
```

## 20 A.4.39 Client.error\_string

```
21     Summary  
22     Pretty-print string representation of pmix\_status\_t.
```

```
1      Format  
2      PMIx v4.0          Python  
3      rep = myclient.error_string(status:integer)  
4      Python  
5      IN   status  
6      PMIx status code (integer)
```

5 Returns:  
6 • *rep* - String representation of the provided status code (str)  
7 See [PMIx\\_Error\\_string](#) for further details.

## 8 A.4.40 Client.proc\_state\_string

```
9      Summary  
10     Pretty-print string representation of pmix\_proc\_state\_t.  
11      Format  
12      PMIx v4.0          Python  
13      rep = myclient.proc_state_string(state:integer)  
14      Python
```

```
13     IN   state  
14     PMIx process state code (integer)  
15 Returns:  
16 • rep - String representation of the provided process state (str)  
17 See PMIx\_Proc\_state\_string for further details.
```

## 18 A.4.41 Client.scope\_string

```
19      Summary  
20     Pretty-print string representation of pmix\_scope\_t.  
21      Format  
22      PMIx v4.0          Python  
23      rep = myclient.scope_string(scope:integer)  
24      Python
```

```
23     IN   scope  
24     PMIx scope value (integer)  
25 Returns:  
26 • rep - String representation of the provided scope (str)  
27 See PMIx\_Scope\_string for further details
```

1 **A.4.42 Client.persistence\_string**

2 **Summary**

3 Pretty-print string representation of [pmix\\_persistence\\_t](#).

4 **Format**

5 *PMIx v4.0*

Python

6    *rep = myclient.persistence\_string(persistence:integer)*

Python

7 **IN persistence**

8    PMIx persistence value (integer)

9 Returns:

- 10    • *rep* - String representation of the provided persistence (str)

See [PMIx\\_Persistence\\_string](#) for further details.

11 **A.4.43 Client.data\_range\_string**

12 **Summary**

13 Pretty-print string representation of [pmix\\_data\\_range\\_t](#).

14 **Format**

15 *PMIx v4.0*

Python

16    *rep = myclient.data\_range\_string(range:integer)*

Python

17 **IN range**

18    PMIx data range value (integer)

19 Returns:

- 20    • *rep* - String representation of the provided data range (str)

See [PMIx\\_Data\\_range\\_string](#) for further details.

21 **A.4.44 Client.info\_directives\_string**

22 **Summary**

23 Pretty-print string representation of [pmix\\_info\\_directives\\_t](#).

```
1      Format  
2      PMIx v4.0          Python  
3      rep = myclient.info_directives_string(directives:bitarray)  
4      Python  
5      IN  directives  
6      PMIx info directives value (bitarray)  
7      Returns:  
8      • rep - String representation of the provided info directives (str)  
9      See PMIx\_Info\_directives\_string for further details.
```

## A.4.45 Client.data\_type\_string

```
9      Summary  
10     Pretty-print string representation of pmix\_data\_type\_t.  
11     Format  
12     PMIx v4.0          Python  
13     rep = myclient.data_type_string(dtype:integer)  
14     Python
```

```
13     IN  dtype  
14     PMIx datatype value (integer)  
15     Returns:  
16     • rep - String representation of the provided datatype (str)  
17     See PMIx\_Data\_type\_string for further details.
```

## A.4.46 Client.alloc\_directive\_string

```
19     Summary  
20     Pretty-print string representation of pmix\_alloc\_directive\_t.  
21     Format  
22     PMIx v4.0          Python  
23     rep = myclient.alloc_directive_string(adir:integer)  
24     Python
```

```
23     IN  adir  
24     PMIx allocation directive value (integer)  
25     Returns:  
26     • rep - String representation of the provided allocation directive (str)  
27     See PMIx\_Alloc\_directive\_string for further details.
```

1 **A.4.47 Client.iof\_channel\_string**

2 **Summary**

3 Pretty-print string representation of [pmix\\_iof\\_channel\\_t](#).

4 **Format**

5 *PMIx v4.0*

Python

6    *rep = myclient.iof\_channel\_string(channel:bitarray)*

Python

7 **IN channel**

8    PMIx IOF [channel](#) value (bitarray)

9 Returns:

- 10    • *rep* - String representation of the provided IOF channel (str)

See [PMIx\\_IOF\\_channel\\_string](#) for further details.

11 **A.4.48 Client.job\_state\_string**

12 **Summary**

13 Pretty-print string representation of [pmix\\_job\\_state\\_t](#).

14 **Format**

15 *PMIx v4.0*

Python

16    *rep = myclient.job\_state\_string(state:integer)*

Python

17 **IN state**

18    PMIx job state value (integer)

19 Returns:

- 20    • *rep* - String representation of the provided job state (str)

See [PMIx\\_Job\\_state\\_string](#) for further details.

21 **A.4.49 Client.get\_attribute\_string**

22 **Summary**

23 Pretty-print string representation of a PMIx attribute.

```
1      Format  
2      PMIx v4.0          Python  
3      rep = myclient.get_attribute_string(attribute:str)  
4      Python  
5      IN   attribute  
6      PMIx attribute name (string)  
7      Returns:  
8      • rep - String representation of the provided attribute (str)  
9      See PMIx\_Get\_attribute\_string for further details.
```

## A.4.50 Client.get\_attribute\_name

### Summary

Pretty-print name of a PMIx attribute corresponding to the provided string.

### Format

```
PMIx v4.0          Python  
rep = myclient.get_attribute_name(attribute:str)  
Python
```

IN attributestring  
Attribute string (string)

Returns:

- rep - Attribute name corresponding to the provided string (str)

See [PMIx\\_Get\\_attribute\\_name](#) for further details.

## A.4.51 Client.link\_state\_string

### Summary

Pretty-print string representation of [pmix\\_link\\_state\\_t](#).

### Format

```
PMIx v4.0          Python  
rep = myclient.link_state_string(state:integer)  
Python
```

IN state  
PMIx link state value (integer)

Returns:

- rep - String representation of the provided link state (str)

See [PMIx\\_Link\\_state\\_string](#) for further details.

## 1 A.5 PMIxServer

2 The server Python class inherits the Python "client" class as its parent. Thus, it includes all client  
3 functions in addition to the ones defined in this section.

### 4 A.5.1 Server.init

#### 5 Summary

6 Initialize the PMIx server library after obtaining a new PMIxServer object.

#### 7 Format

8 *PMIx v4.0*

Python

```
9     rc = myserver.init(directives:list, map:dict)
```

Python

10 IN directives

11 List of Python [info](#) dictionaries (list)

12 IN map

13 Python dictionary key-function pairs that map [server module](#) callback functions to  
provided implementations (dict)

14 Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

15 See [PMIx\\_server\\_init](#) for description of all relevant attributes and behaviors.

### 17 A.5.2 Server.finalize

#### 18 Summary

19 Finalize the PMIx server library.

#### 20 Format

21 *PMIx v4.0*

Python

```
22     rc = myserver.finalize()
```

Python

23 Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

24 See [PMIx\\_server\\_finalize](#) for details.

## 1 A.5.3 Server.generate\_regex

### 2 Summary

3 Generate a regular expression representation of the input strings.

### 4 Format

5 *PMIx v4.0*

Python

```
6 rc, regex = myserver.generate_regex(input:list)
```

Python

#### 7 IN input

8 List of Python strings (e.g., node names) (list)

9 Returns:

- 10 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 11 • *regex* - Python **bytearray** containing regular expression representation of the input list  
**(bytearray)**

12 See [PMIx\\_generate\\_regex](#) for details.

## 13 A.5.4 Server.generate\_ppn

### 14 Summary

15 Generate a regular expression representation of the input strings.

### 16 Format

17 *PMIx v4.0*

Python

```
18 rc, regex = myserver.generate_ppn(input:list)
```

Python

#### 19 IN input

20 List of Python strings, each string consisting of a comma-delimited list of ranks on each node,  
with the strings being in the same order as the node names provided to "generate\_regex" (list)

21 Returns:

- 22 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 23 • *regex* - Python **bytearray** containing regular expression representation of the input list  
**(bytearray)**

25 See [PMIx\\_generate\\_ppn](#) for details.

1 **A.5.5 Server.register\_nspace**

2      **Summary**

3      Setup the data about a particular namespace.

4      **Format**

5      *PMIx v4.0*

Python

```
6      rc = myserver.register_nspace(nspace:str,
7                                         nlocalprocs:integer,
8                                         directives:list)
```

Python

8      **IN nspace**

9      Python string containing the namespace (str)

10     **IN nlocalprocs**

11     Number of local processes (integer)

12     **IN directives**

13     List of Python [info](#) dictionaries (list)

14    Returns:

- 15    • *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

16    See [PMIx\\_server\\_register\\_nspace](#) for description of all relevant attributes and behaviors.

17 **A.5.6 Server.deregister\_nspace**

18      **Summary**

19      Deregister a namespace.

20      **Format**

21      *PMIx v4.0*

Python

```
22      myserver.deregister_nspace(nspace:str)
```

Python

22      **IN nspace**

23      Python string containing the namespace (str)

24    Returns: None

25    See [PMIx\\_server\\_deregister\\_nspace](#) for details.

26 **A.5.7 Server.register\_resources**

27      **Summary**

28      Register non-namespace related information with the local PMIx library

```
1      Format  
2      PMIx v4.0          Python  
3      myserver.register_resources(directives:list)  
4      Python  
5      IN  directives  
6      List of Python info dictionaries (list)  
7      Returns: None  
8      See PMIx\_server\_register\_resources for details.
```

## 7 A.5.8 Server.deregister\_resources

```
8      Summary  
9      Remove non-namespace related information from the local PMIx library
```

```
10     Format  
11     PMIx v4.0          Python  
12     myserver.deregister_resources(directives:list)  
13     Python  
14     IN  directives  
15     List of Python info dictionaries (list)  
16     Returns: None  
17     See PMIx\_server\_deregister\_resources for details.
```

## 16 A.5.9 Server.register\_client

```
17     Summary  
18     Register a client process with the PMIx server library.
```

1       **Format**

2       `rc = myserver.register_client(proc:dict, uid:integer, gid:integer)`

Python

3       **IN proc**

4           Python `proc` dictionary identifying the client process (dict)

5       **IN uid**

6           Linux uid value for user executing client process (integer)

7       **IN gid**

8           Linux gid value for user executing client process (integer)

9       Returns:

- 10
  - `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

11       See [PMIx\\_server\\_register\\_client](#) for details.

12      **A.5.10 Server.deregister\_client**

13      **Summary**

14       Deregister a client process and purge all data relating to it.

15      **Format**

16       `myserver.deregister_client(proc:dict)`

Python

17       **IN proc**

18           Python `proc` dictionary identifying the client process (dict)

19       Returns: None

20       See [PMIx\\_server\\_deregister\\_client](#) for details.

21      **A.5.11 Server.setup\_fork**

22      **Summary**

23       Setup the environment of a child process that is to be forked by the host.

```
1      Format  
2      PMIx v4.0          Python  
3      rc = myserver.setup_fork(proc:dict, envin:dict)  
4      Python  
5      IN proc  
6          Python proc dictionary identifying the client process (dict)  
7      INOUT envin  
8          Python dictionary containing the environment to be passed to the client (dict)  
9      Returns:  
10     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
11     See PMIx\_server\_setup\_fork for details.
```

## 10 A.5.12 Server.dmodex\_request

```
11    Summary  
12    Function by which the host server can request modex data from the local PMIx server.  
13    Format  
14    PMIx v4.0          Python  
15    rc,data = myserver.dmodex_request(proc:dict)  
16    Python  
17    IN proc  
18        Python proc dictionary identifying the process whose data is requested (dict)  
19    Returns:  
20    • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
21    • data - Python byteobject containing the returned data (dict)  
22    See PMIx\_server\_dmodex\_request for details.
```

## 21 A.5.13 Server.setup\_application

```
22    Summary  
23    Function by which the resource manager can request application-specific setup data prior to launch  
24    of a job.
```

```
1      Format  
PMIx v4.0  
2      rc,info = myserver.setup_application(nspace:str, directives:list)  
Python  
3      IN  nspace  
4          Namespace whose setup information is being requested (str)  
5      IN  directives  
6          Python list of info directives  
7      Returns:  
8          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9          • info - Python list of info dictionaries containing the returned data (list)  
10     See PMIx\_server\_setup\_application for details.
```

## A.5.14 Server.register\_attributes

```
12     Summary  
13     Register host environment attribute support for a function.  
14     Format  
PMIx v4.0  
15     rc = myserver.register_attributes(function:str, attrs:list)  
Python  
16     IN  function  
17         Name of the function (str)  
18     IN  attrs  
19         Python list of regattr describing the supported attributes  
20     Returns:  
21         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22     See PMIx\_Register\_attributes for details.
```

## A.5.15 Server.setup\_local\_support

```
24     Summary  
25     Function by which the local PMIx server can perform any application-specific operations prior to  
26     spawning local clients of a given application.
```

```
1      Format  
2      PMIx v4.0          Python  
3      rc = myserver.setup_local_support(nspace:str, info:list)  
4      Python  
5      IN  nspace  
6      Namespace whose setup information is being requested (str)  
7      IN  info  
8      Python list of info containing the setup data (list)  
9      Returns:  
10     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
11     See PMIx\_server\_setup\_local\_support for details.
```

## A.5.16 Server.iof\_deliver

```
11    Summary  
12    Function by which the host environment can pass forwarded IO to the PMIx server library for  
13    distribution to its clients.
```

```
14    Format  
15    PMIx v4.0          Python  
16    rc = myserver.iof_deliver(source:dict, channel:integer,  
17                                data:dict, directives:list)  
18    Python  
19    IN  source  
20      Python proc dictionary identifying the process who generated the data (dict)  
21    IN  channel  
22      Python channel bitmask identifying IO channel of the provided data (integer)  
23    IN  data  
24      Python byteobject containing the data (dict)  
25    IN  directives  
26      Python list of info containing directives (list)  
27      Returns:  
28      • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
29      See PMIx\_server\_IOF\_deliver for details.
```

## A.5.17 Server.collect\_inventory

```
29    Summary  
30    Collect inventory of resources on a node.
```

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myserver.collect_inventory(directives:list)  
4      Python  
5      IN  directives  
6          Optional Python list of info containing directives (list)  
7      Returns:  
8          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9          • info - Python list of info containing the returned data (list)  
10         See PMIx\_server\_collect\_inventory for details.
```

## 9 A.5.18 Server.deliver\_inventory

```
10     Summary  
11     Pass collected inventory to the PMIx server library for storage.  
12     Format  
13     PMIx v4.0          Python  
14     rc = myserver.deliver_inventory(info:list, directives:list)  
15     Python  
16     IN  info  
17         - Python list of info dictionaries containing the inventory data (list)  
18     IN  directives  
19         Python list of info dictionaries containing directives (list)  
20     Returns:  
21         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22         See PMIx\_server\_deliver\_inventory for details.
```

## 21 A.5.19 Server.generate\_locality\_string

```
22     Summary  
23     Generate a PMIx locality string from a given cpuset.
```

```
1      Format  
2      PMIx v4.0          Python  
3      rc,locality = myserver.generate_locality_string(cpuset:cpuset)  
4      IN cpuset  
5      - Python cpuset identifying the PUs (cpuset)  
6      Returns:  
7      • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
8      • locality - string representation of the locality corresponding to the provided cpuset (str)  
9      See PMIx_server_generate_locality_string for details.
```

## 9 A.5.20 Server.define\_process\_set

```
10     Summary  
11     Add members to a PMIx process set.  
12     Format  
13     PMIx v4.0          Python  
14     rc = myserver.define_process_set(members:list, name:str)  
15     IN members  
16     - List of Python proc dictionaries identifying the processes to be added to the process set  
17     (list)  
18     IN name  
19     - Name of the process set (str)  
20     Returns:  
21     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22     See PMIx_server_define_process_set for details.
```

## 22 A.5.21 Server.delete\_process\_set

```
23     Summary  
24     Delete members from a PMIx process set.
```

```
1      Format  
2      PMIx v4.0   Python  
3      rc = myserver.delete_process_set(members:list, name:str)  
4      Python  
5      IN members  
6          - List of Python proc dictionaries identifying the processes to be removed from the process  
7          set (list)  
8      IN name  
9          - Name of the process set (str)  
10     Returns:  
11     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
12     See PMIx\_server\_delete\_process\_set for details.
```

## 11 A.6 PMIxTool

12 The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and  
13 server functions in addition to the ones defined in this section.

### 14 A.6.1 Tool.init

15 **Summary**  
16 Initialize the PMIx tool library after obtaining a new PMIxTool object.

```
17      Format  
18      PMIx v4.0   Python  
19      rc,proc = mytool.init(info:list)  
20      Python  
21      IN info  
22          List of Python info directives (list)  
23      Returns:  
24          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
25          • proc - a Python proc (dict)  
26          See PMIx\_tool\_init for description of all relevant attributes and behaviors.
```

### 25 A.6.2 Tool.finalize

26 **Summary**  
27 Finalize the PMIx tool library, closing the connection to the server.

```
1      Format  
2      PMIx v4.0   Python  
3      rc = mytool.finalize()  
4      Python  
5      Returns:  
6      • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
7      See PMIx\_tool\_finalize for description of all relevant attributes and behaviors.
```

### 6 A.6.3 Tool.disconnect

#### 7 Summary

8 Disconnect the PMIx tool from the specified server connection while leaving the tool library  
9 initialized.

#### 10 Format

```
10     PMIx v4.0   Python  
11     rc = mytool.disconnect(server:dict)  
12     Python  
13     IN  server  
14     Process identifier of server from which the tool is to be disconnected (proc)  
15     Returns:  
16     • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
17     See PMIx\_tool\_disconnect for details.
```

### 17 A.6.4 Tool.attach\_to\_server

#### 18 Summary

19 Switch connection from the current PMIx server to another one, or initialize a connection to a  
20 specified server.

```
1      Format  
2      rc,proc,server = mytool.connect_to_server(info:list)  
3      IN   info  
4          List of Python info dictionaries (list)  
5      Returns:  
6          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
7          • proc - a Python proc containing the tool's identifier (dict)  
8          • server - a Python proc containing the identifier of the server to which the tool attached (dict)  
9      See PMIx_tool_attach_to_server for details.
```

## 10 A.6.5 Tool.get\_servers

### 11 Summary

12 Get a list containing the **proc** process identifiers of all servers to which the tool is currently  
13 connected.

### 14 Format

```
14      PMIx v4.0  
15      rc,servers = mytool.get_servers()  
16      Returns:  
17          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
18          • servers - a list of Python proc containing the identifiers of the servers to which the tool is  
19              currently attached (dict)  
20      See PMIx_tool_get_servers for details.
```

## 21 A.6.6 Tool.iof\_pull

### 22 Summary

23 Register to receive output forwarded from a remote process.

```
1      Format  
2      PMIx v4.0   Python  
3      rc,id = mytool.iof_pull(sources:list, channel:integer,  
4                                directives:list, cbfunc)  
5      Python  
6  
7      IN sources  
8          List of Python proc dictionaries of processes whose IO is being requested (list)  
9      IN channel  
10         Python channel bitmask identifying IO channels to be forwarded (integer)  
11      IN directives  
12          List of Python info dictionaries describing request (list)  
13      IN cbfunc  
14          Python iofcbfunc to receive IO payloads (func)  
15  
16     Returns:  
17         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
18         • id - PMIx reference identifier for request (integer)  
19  
20     See PMIx_IOF_pull for description of all relevant attributes and behaviors.
```

## 16 A.6.7 Tool.iof\_deregister

```
17      Summary  
18      Deregister from output forwarded from a remote process.  
19      Format  
20      PMIx v4.0   Python  
21      rc = mytool.iof_deregister(id:integer, directives:list)  
22      Python  
23  
24      IN id  
25          PMIx reference identifier returned by pull request (list)  
26      IN directives  
27          List of Python info dictionaries describing request (list)  
28  
29     Returns:  
30         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
31  
32     See PMIx_IOF_deregister for description of all relevant attributes and behaviors.
```

## 1 A.6.8 Tool.iof\_push

### 2 Summary

3 Push data collected locally (typically from stdin) to stdin of target recipients.

### 4 Format

5 PMIx v4.0

Python

```
6   rc = mytool.iof_push(targets:list, data:dict, directives:list)
```

Python

#### 6 IN sources

7 List of Python `proc` of target processes (list)

#### 8 IN data

9 Python `byteobject` containing data to be delivered (dict)

#### 10 IN directives

11 Optional list of Python `info` describing request (list)

12 Returns:

- 13 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

14 See `PMIx_IOF_push` for description of all relevant attributes and behaviors.

## 15 A.7 Example Usage

16 The following examples are provided to illustrate the use of the Python bindings.

### 17 A.7.1 Python Client

18 The following example contains a client program that illustrates a fairly common usage pattern.

19 The program instantiates and initializes the PMIxClient class, posts some data that is to be shared  
20 across all processes in the job, executes a “fence” that circulates the data, and then retrieves a value  
21 posted by one of its peers. Note that the example has been formatted to fit the document layout.

```
1  from pmix import *
2
3  def main():
4      # Instantiate a client object
5      myclient = PMIxClient()
6      print("Testing PMIx ", myclient.get_version())
7
8      # Initialize the PMIx client library, declaring the programming model
9      # as "TEST" and the library name as "PMIX", just for the example
10     info = ['key':PMIX_PROGRAMMING_MODEL,
11             'value':'TEST', 'val_type':PMIX_STRING,
12             'key':PMIX_MODEL_LIBRARY_NAME,
13             'value':'PMIX', 'val_type':PMIX_STRING]
14     rc,myname = myclient.init(info)
15     if PMIX_SUCCESS != rc:
16         print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
17         exit(1)
18
19     # try posting a value
20     rc = myclient.put(PMIX_GLOBAL, "mykey",
21                       'value':1, 'val_type':PMIX_INT32)
22     if PMIX_SUCCESS != rc:
23         print("PMIx_Put FAILED WITH ERROR", myclient.error_string(rc))
24         # cleanly finalize
25         myclient.finalize()
26         exit(1)
27
28     # commit it
29     rc = myclient.commit()
30     if PMIX_SUCCESS != rc:
31         print("PMIx_Commit FAILED WITH ERROR",
32               myclient.error_string(rc))
33         # cleanly finalize
34         myclient.finalize()
35         exit(1)
36
37     # execute fence across all processes in my job
38     procs = []
39     info = []
40     rc = myclient.fence(procs, info)
41     if PMIX_SUCCESS != rc:
42         print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
```

```

1      # cleanly finalize
2      myclient.finalize()
3      exit(1)
4
5      # Get a value from a peer
6      if 0 != myname['rank']:
7          info = []
8          rc, get_val = myclient.get('nspacE' :"testnspace", 'rank': 0,
9                                      "mykey", info)
10     if PMIX_SUCCESS != rc:
11         print("PMIx_Commit FAILED WITH ERROR",
12               myclient.error_string(rc))
13     # cleanly finalize
14     myclient.finalize()
15     exit(1)
16     print("Get value returned: ", get_val)
17
18     # test a fence that should return not_supported because
19     # we pass a required attribute that the server is known
20     # not to support
21     procs = []
22     info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
23             'value':10, 'val_type':PMIX_INT]
24     rc = myclient.fence(procs, info)
25     if PMIX_SUCCESS == rc:
26         print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
27         # cleanly finalize
28         myclient.finalize()
29         exit(1)
30
31     # Publish something
32     info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
33     rc = myclient.publish(info)
34     if PMIX_SUCCESS != rc:
35         print("PMIx_Publish FAILED WITH ERROR",
36               myclient.error_string(rc))
37     # cleanly finalize
38     myclient.finalize()
39     exit(1)
40
41     # finalize
42     info = []
43     myclient.finalize(info)

```

```
1     print("Client finalize complete")
2
3 # Python main program entry point
4 if __name__ == '__main__':
5     main()
```

Python

## A.7.2 Python Server

The following example contains a minimum-level server host program that instantiates and initializes the PMIxServer class. The program illustrates passing several server module functions to the bindings and includes code to setup and spawn a simple client application, waiting until the spawned client terminates before finalizing and exiting itself. Note that the example has been formatted to fit the document layout.

Python

```
12 from pmix import *
13 import signal, time
14 import os
15 import select
16 import subprocess
17
18 def clientconnected(proc:tuple is not None):
19     print("CLIENT CONNECTED", proc)
20     return PMIX_OPERATION_SUCCEEDED
21
22 def clientfinalized(proc:tuple is not None):
23     print("CLIENT FINALIZED", proc)
24     return PMIX_OPERATION_SUCCEEDED
25
26 def clientfence(procs:list, directives:list, data:bytearray):
27     # check directives
28     if directives is not None:
29         for d in directives:
30             # these are each an info dict
31             if "pmix" not in d['key']:
32                 # we do not support such directives - see if
33                 # it is required
34                 try:
35                     if d['flags'] & PMIX_INFO_REQD:
36                         # return an error
37                         return PMIX_ERR_NOT_SUPPORTED
38                 except:
```

```

1                         #it can be ignored
2                         pass
3             return PMIX_OPERATION_SUCCEEDED
4
5 def main():
6     try:
7         myserver = PMIxServer()
8     except:
9         print("FAILED TO CREATE SERVER")
10        exit(1)
11    print("Testing server version ", myserver.get_version())
12
13    args = ['key':PMIX_SERVER_SCHEDULER,
14            'value':'T', 'val_type':PMIX_BOOL]
15    map = 'clientconnected': clientconnected,
16            'clientfinalized': clientfinalized,
17            'fencenb': clientfence
18    my_result = myserver.init(args, map)
19
20    # get our environment as a base
21    env = os.environ.copy()
22
23    # register an nspace for the client app
24    (rc, regex) = myserver.generate_regex("test000,test001,test002")
25    (rc, ppn) = myserver.generate_ppn("0")
26    kvals = ['key':PMIX_NODE_MAP,
27              'value':regex, 'val_type':PMIX_STRING,
28              'key':PMIX_PROC_MAP,
29              'value':ppn, 'val_type':PMIX_STRING,
30              'key':PMIX_UNIV_SIZE,
31              'value':1, 'val_type':PMIX_UINT32,
32              'key':PMIX_JOB_SIZE,
33              'value':1, 'val_type':PMIX_UINT32]
34    rc = foo.register_nspace("testnspace", 1, kvals)
35    print("RegNspace ", rc)
36
37    # register a client
38    uid = os.getuid()
39    gid = os.getgid()
40    rc = myserver.register_client('nspace':"testnspace", 'rank':0,
41                                  uid, gid)
42    print("RegClient ", rc)
43    # setup the fork

```

```

1      rc = myserver.setup_fork('nspacename' :"testnspace", 'rank':0, env)
2      print("SetupFrk", rc)
3
4      # setup the client argv
5      args = ["../client.py"]
6      # open a subprocess with stdout and stderr
7      # as distinct pipes so we can capture their
8      # output as the process runs
9      p = subprocess.Popen(args, env=env,
10                          stdout=subprocess.PIPE, stderr=subprocess.PIPE)
11      # define storage to catch the output
12      stdout = []
13      stderr = []
14      # loop until the pipes close
15      while True:
16          reads = [p.stdout.fileno(), p.stderr.fileno()]
17          ret = select.select(reads, [], [])
18
19          stdout_done = True
20          stderr_done = True
21
22          for fd in ret[0]:
23              # if the data
24              if fd == p.stdout.fileno():
25                  read = p.stdout.readline()
26                  if read:
27                      read = read.decode('utf-8').rstrip()
28                      print('stdout: ' + read)
29                      stdout_done = False
30              elif fd == p.stderr.fileno():
31                  read = p.stderr.readline()
32                  if read:
33                      read = read.decode('utf-8').rstrip()
34                      print('stderr: ' + read)
35                      stderr_done = False
36
37          if stdout_done and stderr_done:
38              break
39      print("FINALIZING")
40      myserver.finalize()
41
42
43  if __name__ == '__main__':

```

1

`main()`

Python

## APPENDIX B

# Revision History

---

## 1 B.1 Version 1.0: June 12, 2015

2 The PMIx version 1.0 *ad hoc* standard was defined in a set of header files as part of the v1.0.0  
3 release of the OpenPMIx library prior to the creation of the formal PMIx 2.0 standard. Below are a  
4 summary listing of the interfaces defined in the 1.0 headers.

5 • Client APIs

- 6   – `PMIx_Init`, `PMIx_Initialized`, `PMIx_Abort`, `PMIx_Finalize`  
7   – `PMIx_Put`, `PMIx_Commit`,  
8   – `PMIx_Fence`, `PMIx_Fence_nb`  
9   – `PMIx_Get`, `PMIx_Get_nb`  
10   – `PMIx_Publish`, `PMIx_Publish_nb`  
11   – `PMIx_Lookup`, `PMIx_Lookup_nb`  
12   – `PMIx_Unpublish`, `PMIx_Unpublish_nb`  
13   – `PMIx_Spawn`, `PMIx_Spawn_nb`  
14   – `PMIx_Connect`, `PMIx_Connect_nb`  
15   – `PMIx_Disconnect`, `PMIx_Disconnect_nb`  
16   – `PMIx_Resolve_nodes`, `PMIx_Resolve_peers`

17 • Server APIs

- 18   – `PMIx_server_init`, `PMIx_server_finalize`  
19   – `PMIx_generate_regex`, `PMIx_generate_ppn`  
20   – `PMIx_server_register_nspace`, `PMIx_server_deregister_nspace`  
21   – `PMIx_server_register_client`, `PMIx_server_deregister_client`  
22   – `PMIx_server_setup_fork`, `PMIx_server_dmodex_request`

23 • Common APIs

- 24   – `PMIx_Get_version`, `PMIx_Store_internal`, `PMIx_Error_string`  
25   – `PMIx_Register_errhandler`, `PMIx_Deregister_errhandler`, `PMIx_Notify_error`

26 The `PMIx_Init` API was subsequently modified in the v1.1.0 release of that library.

## 1 B.2 Version 2.0: Sept. 2018

2 The following APIs were introduced in v2.0 of the PMIx Standard:

3 • Client APIs

- 4   – `PMIx_Query_info_nb`, `PMIx_Log_nb`  
5   – `PMIx_Allocation_request_nb`, `PMIx_Job_control_nb`,  
6    `PMIx_Process_monitor_nb`, `PMIx_Heartbeat`

7 • Server APIs

- 8   – `PMIx_server_setup_application`, `PMIx_server_setup_local_support`

9 • Tool APIs

- 10   – `PMIx_tool_init`, `PMIx_tool_finalize`

11 • Common APIs

- 12   – `PMIx_Register_event_handler`, `PMIx_Deregister_event_handler`  
13   – `PMIx_Notify_event`  
14   – `PMIx_Proc_state_string`, `PMIx_Scope_string`  
15   – `PMIx_Persistence_string`, `PMIx_Data_range_string`  
16   – `PMIx_Info_directives_string`, `PMIx_Data_type_string`  
17   – `PMIx_Alloc_directive_string`  
18   – `PMIx_Data_pack`, `PMIx_Data_unpack`, `PMIx_Data_copy`  
19   – `PMIx_Data_print`, `PMIx_Data_copy_payload`

### 20 B.2.1 Removed/Modified APIs

21 The `PMIx_Init` API was modified in v2.0 of the standard from its *ad hoc* v1.0 signature to  
22 include passing of a `pmix_info_t` array for flexibility and “future-proofing” of the API. In  
23 addition, the `PMIx_Notify_error`, `PMIx_Register_errhandler`, and `PMIx_Deregister_errhandler`  
24 APIs were replaced. This pre-dated official adoption of PMIx as a Standard.

### 25 B.2.2 Deprecated constants

26 The following constants were deprecated in v2.0:

27   `PMIX_MODEX`  
28   `PMIX_INFO_ARRAY`

## 1 B.2.3 Deprecated attributes

2 The following attributes were deprecated in v2.0:

3   **PMIX\_ERROR\_NAME** "pmix.errname" (**pmix\_status\_t**)  
4       Specific error to be notified  
5   **PMIX\_ERROR\_GROUP\_COMM** "pmix.errgroup.comm" (**bool**)  
6       Set true to get comm errors notification  
7   **PMIX\_ERROR\_GROUP\_ABORT** "pmix.errgroup.abort" (**bool**)  
8       Set true to get abort errors notification  
9   **PMIX\_ERROR\_GROUP\_MIGRATE** "pmix.errgroup.migrate" (**bool**)  
10      Set true to get migrate errors notification  
11   **PMIX\_ERROR\_GROUP\_RESOURCE** "pmix.errgroup.resource" (**bool**)  
12      Set true to get resource errors notification  
13   **PMIX\_ERROR\_GROUP\_SPAWN** "pmix.errgroup.spawn" (**bool**)  
14      Set true to get spawn errors notification  
15   **PMIX\_ERROR\_GROUP\_NODE** "pmix.errgroup.node" (**bool**)  
16      Set true to get node status notification  
17   **PMIX\_ERROR\_GROUP\_LOCAL** "pmix.errgroup.local" (**bool**)  
18      Set true to get local errors notification  
19   **PMIX\_ERROR\_GROUP\_GENERAL** "pmix.errgroup.gen" (**bool**)  
20      Set true to get notified of generic errors  
21   **PMIX\_ERROR\_HANDLER\_ID** "pmix.errhandler.id" (**int**)  
22      Errhandler reference id of notification being reported

## 23 B.3 Version 2.1: Dec. 2018

24 The v2.1 update includes clarifications and corrections from the v2.0 document, plus addition of  
25 examples:

- 26   • Clarify description of **PMIx\_Connect** and **PMIx\_Disconnect** APIs.
- 27   • Explain that values for the **PMIX\_COLLECTIVE\_ALGO** are environment-dependent
- 28   • Identify the namespace/rank values required for retrieving attribute-associated information using  
29       the **PMIx\_Get** API
- 30   • Provide definitions for *session, job, application*, and other terms used throughout the document
- 31   • Clarify definitions of **PMIX\_UNIV\_SIZE** versus **PMIX\_JOB\_SIZE**
- 32   • Clarify server module function return values
- 33   • Provide examples of the use of **PMIx\_Get** for retrieval of information
- 34   • Clarify the use of **PMIx\_Get** versus **PMIx\_Query\_info\_nb**
- 35   • Clarify return values for non-blocking APIs and emphasize that callback functions must not be  
36       invoked prior to return from the API
- 37   • Provide detailed example for construction of the **PMIx\_server\_register\_nspace** input  
38       information array

- 1     ● Define information levels (e.g., *session* vs *job*) and associated attributes for both storing and  
2        retrieving values  
3     ● Clarify roles of PMIx server library and host environment for collective operations  
4     ● Clarify definition of **PMIX\_UNIV\_SIZE**

## 5     **B.4 Version 2.2: Jan 2019**

6     The v2.2 update includes the following clarifications and corrections from the v2.1 document:

- 7     ● Direct modex upcall function (**pmix\_server\_dmodex\_req\_fn\_t**) cannot complete  
8        atomically as the API cannot return the requested information except via the provided callback  
9        function  
10    ● Add missing **pmix\_data\_array\_t** definition and support macros  
11    ● Add a rule divider between implementer and host environment required attributes for clarity  
12    ● Add **PMIX\_QUERY\_QUALIFIERS\_CREATE** macro to simplify creation of **pmix\_query\_t**  
13        qualifiers  
14    ● Add **PMIX\_APP\_INFO\_CREATE** macro to simplify creation of **pmix\_app\_t** directives  
15    ● Add flag and **PMIX\_INFO\_IS\_END** macro for marking and detecting the end of a  
16        **pmix\_info\_t** array  
17    ● Clarify the allowed hierarchical nesting of the **PMIX\_SESSION\_INFO\_ARRAY**,  
18        **PMIX\_JOB\_INFO\_ARRAY**, and associated attributes

## 19    **B.5 Version 3.0: Dec. 2018**

20    The following APIs were introduced in v3.0 of the PMIx Standard:

- 21    ● Client APIs  
22      – **PMIx\_Log**, **PMIx\_Job\_control**  
23      – **PMIx\_Allocation\_request**, **PMIx\_Process\_monitor**  
24      – **PMIx\_Get\_credential**, **PMIx\_Validate\_credential**  
25    ● Server APIs  
26      – **PMIx\_server\_IOF\_deliver**  
27      – **PMIx\_server\_collect\_inventory**, **PMIx\_server\_deliver\_inventory**  
28    ● Tool APIs  
29      – **PMIx\_IOF\_pull**, **PMIx\_IOF\_push**, **PMIx\_IOF\_deregister**  
30      – **PMIx\_tool\_connect\_to\_server**  
31    ● Common APIs  
32      – **PMIx\_IOF\_channel\_string**

1 The document added a chapter on security credentials, a new section for IO forwarding to the  
2 Process Management chapter, and a few blocking forms of previously-existing non-blocking APIs.  
3 Attributes supporting the new APIs were introduced, as well as additional attributes for a few  
4 existing functions.

## 5 **B.5.1 Removed constants**

6 The following constants were removed in v3.0:

7 **PMIX\_MODEX**  
8 **PMIX\_INFO\_ARRAY**

## 9 **B.5.2 Deprecated attributes**

10 The following attributes were deprecated in v3.0:

11 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)  
12 If **true**, indicates that the requested choice of algorithm is mandatory.

## 13 **B.5.3 Removed attributes**

14 The following attributes were removed in v3.0:

15 **PMIX\_ERROR\_NAME** "pmix.errname" (pmix\_status\_t)  
16 Specific error to be notified  
17 **PMIX\_ERROR\_GROUP\_COMM** "pmix.errgroup.comm" (bool)  
18 Set true to get comm errors notification  
19 **PMIX\_ERROR\_GROUP\_ABORT** "pmix.errgroup.abort" (bool)  
20 Set true to get abort errors notification  
21 **PMIX\_ERROR\_GROUP\_MIGRATE** "pmix.errgroup.migrate" (bool)  
22 Set true to get migrate errors notification  
23 **PMIX\_ERROR\_GROUP\_RESOURCE** "pmix.errgroup.resource" (bool)  
24 Set true to get resource errors notification  
25 **PMIX\_ERROR\_GROUP\_SPAWN** "pmix.errgroup.spawn" (bool)  
26 Set true to get spawn errors notification  
27 **PMIX\_ERROR\_GROUP\_NODE** "pmix.errgroup.node" (bool)  
28 Set true to get node status notification  
29 **PMIX\_ERROR\_GROUP\_LOCAL** "pmix.errgroup.local" (bool)  
30 Set true to get local errors notification  
31 **PMIX\_ERROR\_GROUP\_GENERAL** "pmix.errgroup.gen" (bool)  
32 Set true to get notified of generic errors  
33 **PMIX\_ERROR\_HANDLER\_ID** "pmix.errhandler.id" (int)  
34 Errhandler reference id of notification being reported

## 1 B.6 Version 3.1: Jan. 2019

2 The v3.1 update includes clarifications and corrections from the v3.0 document:

- 3 • Direct modex upcall function (`pmix_server_dmodex_req_fn_t`) cannot complete  
4 atomically as the API cannot return the requested information except via the provided callback  
5 function
- 6 • Fix typo in name of `PMIX_FWD_STDDIAG` attribute
- 7 • Correctly identify the information retrieval and storage attributes as “new” to v3 of the standard
- 8 • Add missing `pmix_data_array_t` definition and support macros
- 9 • Add a rule divider between implementer and host environment required attributes for clarity
- 10 • Add `PMIX_QUERY_QUALIFIERS_CREATE` macro to simplify creation of `pmix_query_t`  
11 qualifiers
- 12 • Add `PMIX_APP_INFO_CREATE` macro to simplify creation of `pmix_app_t` directives
- 13 • Add new attributes to specify the level of information being requested where ambiguity may exist  
(see 6.1)
- 14 • Add new attributes to assemble information by its level for storage where ambiguity may exist  
(see 16.2.3.1)
- 15 • Add flag and `PMIX_INFO_IS_END` macro for marking and detecting the end of a  
`pmix_info_t` array
- 16 • Clarify that `PMIX_NUM_SLOTS` is duplicative of (a) `PMIX_UNIV_SIZE` when used at the  
`session` level and (b) `PMIX_MAX_PROCS` when used at the `job` and `application` levels, but leave  
it in for backward compatibility.
- 17 • Clarify difference between `PMIX_JOB_SIZE` and `PMIX_MAX_PROCS`
- 18 • Clarify that `PMIx_server_setup_application` must be called per-`job` instead of  
per-`application` as the name implies. Unfortunately, this is a historical artifact. Note that both  
`PMIX_NODE_MAP` and `PMIX_PROC_MAP` must be included as input in the `info` array provided  
to that function. Further descriptive explanation of the “instant on” procedure will be provided in  
the next version of the PMIx Standard.
- 19 • Clarify how the PMIx server expects data passed to the host by  
`pmix_server_fencenb_fn_t` should be aggregated across nodes, and provide a code  
snippet example

## 31 B.7 Version 3.2: Sept. 2020

32 The v3.2 update includes clarifications and corrections from the v3.1 document:

- 33 • Correct an error in the `PMIx_Allocation_request` function signature, and clarify the  
34 allocation ID attributes
- 35 • Rename the `PMIX_ALLOC_ID` attribute to `PMIX_ALLOC_REQ_ID` to clarify that this is a  
36 string the user provides as a means to identify their request to query status

- Add a new **PMIX\_ALLOC\_ID** attribute that contains the identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to **PMIx\_Spawn**
- Update the **PMIx\_generate\_regex** and **PMIx\_generate\_ppn** descriptions to clarify that the output from these generator functions may not be a NULL-terminated string, but instead could be a byte array of arbitrary binary content.
- Add a new **PMIX\_REGEX** constant that represents a regular expression data type.

## 8 B.8 Version 4.0: Sept 2020

9 NOTE: The PMIx Standard document has undergone significant reorganization in an effort to  
 10 become more user-friendly. Highlights include:

- Moving all added, deprecated, and removed items to this revision log section to make them more visible
- Co-locating constants and attribute definitions with the primary API that uses them - citations and hyperlinks are retained elsewhere
- Splitting the Key-Value Management chapter into separate chapters on the use of reserved keys, non-reserved keys, and non-process-related key-value data exchange
- Creating a new chapter on synchronization and data access methods
- Removing references to specific implementations of PMIx and to implementation-specific features and/or behaviors

20 In addition to the reorganization, the following changes were introduced in v4.0 of the PMIx  
 21 Standard:

- Clarified that the **PMIx\_Fence\_nb** operation can immediately return **PMIX\_OPERATION\_SUCCEEDED** in lieu of passing the request to a PMIx server if only the calling process is involved in the operation
- Added the **PMIx\_Register\_attributes** API by which a host environment can register the attributes it supports for each server-to-host operation
- Added the ability to query supported attributes from the PMIx tool, client and server libraries, as well as the host environment via the new **pmix\_regattr\_t** structure. Both human-readable and machine-parsable output is supported. New attributes to support this operation include:
  - **PMIX\_CLIENT\_ATTRIBUTES**, **PMIX\_SERVER\_ATTRIBUTES**, **PMIX\_TOOL\_ATTRIBUTES**, and **PMIX\_HOST\_ATTRIBUTES** to identify which library supports the attribute; and
  - **PMIX\_MAX\_VALUE**, **PMIX\_MIN\_VALUE**, and **PMIX\_ENUM\_VALUE** to provide machine-parsable description of accepted values
- Add **PMIX\_APP\_WILDCARD** to reference all applications within a given job
- Fix signature of blocking APIs **PMIx\_Allocation\_request**, **PMIx\_Job\_control**, **PMIx\_Process\_monitor**, **PMIx\_Get\_credential**, and **PMIx\_Validate\_credential** to allow return of results

- Update description to provide an option for blocking behavior of the `PMIx_Register_event_handler`, `PMIx_Deregister_event_handler`, `PMIx_Notify_event`, `PMIx_IOF_pull`, `PMIx_IOF_deregister`, and `PMIx_IOF_push` APIs. The need for blocking forms of these functions was not initially anticipated but has emerged over time. For these functions, the return value is sufficient to provide the caller with information otherwise returned via callback. Thus, use of a **NULL** value as the callback function parameter was deemed a minimal disruption method for providing the desired capability
- Added a chapter on fabric support that includes new APIs, datatypes, and attributes
- Added a chapter on process sets and groups that includes new APIs and attributes
- Added APIs and a new datatype to support generation and parsing of PMIx locality strings
- Added a new chapter on tools that provides deeper explanation on their operation and collecting all tool-relevant definitions into one location. Also introduced two new APIs and removed restriction that limited tools to being connected to only one server at a time.
- Extended behavior of `PMIx_server_init` to scalably expose the topology description to the local clients. This includes creating any required shared memory backing stores and/or XML representations, plus ensuring that all necessary key-value pairs for clients to access the description are included in the job-level information provided to each client.

The above changes included introduction of the following APIs and data types:

- Client APIs

- `PMIx_Group_construct`, `PMIx_Group_construct_nb`
- `PMIx_Group_destruct`, `PMIx_Group_destruct_nb`
- `PMIx_Group_invite`, `PMIx_Group_invite_nb`
- `PMIx_Group_join`, `PMIx_Group_join_nb`
- `PMIx_Group_leave`, `PMIx_Group_leave_nb`
- `PMIx_Get_relative_locality`, `PMIx_Load_topology`
- `PMIx_Get_cpuset`
- `PMIx_Link_state_string`, `PMIx_Job_state_string`
- `PMIx_Fabric_register`, `PMIx_Fabric_register_nb`
- `PMIx_Fabric_update`, `PMIx_Fabric_update_nb`
- `PMIx_Fabric_deregister`, `PMIx_Fabric_deregister_nb`
- `PMIx_Fabric_update_distances`, `PMIx_Fabric_update_distances_nb`

- Server APIs

- `PMIx_server_generate_locality_string`
- `PMIx_Register_attributes`
- `PMIx_server_define_process_set`, `PMIx_server_delete_process_set`
- `pmix_server_grp_fn_t`, `pmix_server_fabric_fn_t`
- `pmix_server_client_connected2_fn_t`
- `PMIx_server_generate_cpuset_string`
- `PMIx_server_register_resources`, `PMIx_server_deregister_resources`

- 1     • Tool APIs
- 2        – `PMIx_tool_disconnect`
- 3        – `PMIx_tool_set_server`
- 4        – `PMIx_tool_attach_to_server`
- 5        – `PMIx_tool_get_servers`
- 6     • Data types
- 7        – `pmix_regattr_t`
- 8        – `pmix_cpuset_t`
- 9        – `pmix_topology_t`
- 10       – `pmix_locality_t`
- 11       – `pmix_group_opt_t`
- 12       – `pmix_group_operation_t`
- 13       – `pmix_fabric_t`
- 14       – `pmix_device_distance_t`
- 15       – `pmix_coord_t`
- 16       – `pmix_coord_view_t`
- 17       – `pmix_geometry_t`
- 18       – `pmix_link_state_t`
- 19       – `pmix_job_state_t`

## 20    B.8.1 Added Constants

### 21    Data type constants

22    `PMIX_COORD` `PMIX_REGATTR` `PMIX_REGEX` `PMIX_JOB_STATE` `PMIX_LINK_STATE`  
23    `PMIX_PROC_CPUSET` `PMIX_GEOMETRY` `PMIX_DEVICE_DIST` `PMIX_ENDPOINT`  
24    `PMIX_DATA_TYPE_MAX`

### 25    Query constants

26    `PMIX_ERR_GET_MALLOC_REQD`

### 27    Server constants

28    `PMIX_ERR_REPEAT_ATTR_REGISTRATION`

### 29    Job-Mgmt constants

30    `PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES`

### 31    Publish constants

32    `PMIX_ERR_DUPLICATE_KEY`

```
1      Tool constants
2      PMIX_LAUNCH_DIRECTIVE
3      PMIX_LAUNCHER_READY
4      PMIX_ERR_IOF_FAILURE
5      PMIX_ERR_IOF_COMPLETE
6      PMIX_EVENT_JOB_START
7      PMIX_LAUNCH_COMPLETE
8      PMIX_EVENT_JOB_END
9      PMIX_EVENT_SESSION_START
10     PMIX_EVENT_SESSION_END
11     PMIX_ERR_PROC_TERM_WO_SYNC
12     PMIX_ERR_JOB_CANCELED
13     PMIX_ERR_JOB_ABORTED
14     PMIX_ERR_JOB_KILLED_BY_CMD
15     PMIX_ERR_JOB_ABORTED_BY_SIG
16     PMIX_ERR_JOB_TERM_WO_SYNC
17     PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED
18     PMIX_ERR_JOB_NON_ZERO_TERM
19     PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT
20     PMIX_DEBUG_WAITING_FOR_NOTIFY
21     PMIX_DEBUGGER_RELEASE
22
```

```
23     Fabric constants
24     PMIX_FABRIC_UPDATE_PENDING
25     PMIX_FABRIC_UPDATED
26     PMIX_FABRIC_UPDATE_ENDPOINTS
27     PMIX_COORD_VIEW_UNDEF
28     PMIX_COORD_LOGICAL_VIEW
29     PMIX_COORD_PHYSICAL_VIEW
30     PMIX_LINK_STATE_UNKNOWN
31     PMIX_LINK_DOWN
32     PMIX_LINK_UP
33     PMIX_FABRIC_REQUEST_INFO
34     PMIX_FABRIC_UPDATE_INFO
35
```

```
36     Sets-Groups constants
37     PMIX_PROCESS_SET_DEFINE
38     PMIX_PROCESS_SET_DELETE
39     PMIX_GROUP_INVITED
40     PMIX_GROUP_LEFT
41     PMIX_GROUP_MEMBER_FAILED
42     PMIX_GROUP_INVITE_ACCEPTED
```

```
1 PMIX_GROUP_INVITE_DECLINED  
2 PMIX_GROUP_INVITE_FAILED  
3 PMIX_GROUP_MEMBERSHIP_UPDATE  
4 PMIX_GROUP_CONSTRUCT_ABORT  
5 PMIX_GROUP_CONSTRUCT_COMPLETE  
6 PMIX_GROUP_LEADER_FAILED  
7 PMIX_GROUP_LEADER_SELECTED  
8 PMIX_GROUP_CONTEXT_ID_ASSIGNED  
9
```

## 10 Process-Mgmt constants

```
11 PMIX_ERR_JOB_ALLOC_FAILED  
12 PMIX_ERR_JOB_APP_NOT_EXECUTABLE  
13 PMIX_ERR_JOB_NO_EXE_SPECIFIED  
14 PMIX_ERR_JOB_FAILED_TO_MAP  
15 PMIX_ERR_JOB_FAILED_TO_LAUNCH  
16 PMIX_LOCALITY_UNKNOWN  
17 PMIX_LOCALITY_NONLOCAL  
18 PMIX_LOCALITY_SHARE_HWTHREAD  
19 PMIX_LOCALITY_SHARE_CORE  
20 PMIX_LOCALITY_SHARE_L1CACHE  
21 PMIX_LOCALITY_SHARE_L2CACHE  
22 PMIX_LOCALITY_SHARE_L3CACHE  
23 PMIX_LOCALITY_SHARE_PACKAGE  
24 PMIX_LOCALITY_SHARE_NUMA  
25 PMIX_LOCALITY_SHARE_NODE  
26
```

## 27 Events

```
28 PMIX_EVENT_SYS_BASE  
29 PMIX_EVENT_NODE_DOWN  
30 PMIX_EVENT_NODE_OFFLINE  
31 PMIX_EVENT_SYS_OTHER  
32
```

## 33 B.8.2 Added Attributes

### 34 Sync-Access attributes

```
35 PMIX_COLLECT_GENERATED_JOB_INFO "pmix.collect.gen" (bool)
```

36 Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx  
37 servers. Some job-level information (e.g., distance between processes and fabric devices) is  
38 best determined on a distributed basis as it primarily pertains to local processes. Should  
39 remote processes need to access the information, it can either be obtained collectively using  
40 the **PMIx\_Fence** operation with this directive, or can be retrieved one peer at a time using  
41 **PMIx\_Get** without first having performed the job-wide collection.

```

1   PMIX_ALL_CLONES_PARTICIPATE "pmix.clone.part" (bool)
2     All clones of the calling process must participate in the collective operation.

3   PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)
4     Request that any pointers in the returned value point directly to values in the key-value store.
5     The user must not release any returned data pointers. Note that a return status of
6     PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported - in
7     which case, the returned data contains allocated memory that the user must release.

8   PMIX_GET_REFRESH_CACHE "pmix.get.refresh" (bool)
9     When retrieving data for a remote process, refresh the existing local data cache for the
10    process in case new values have been put and committed by the process since the last refresh.
11

12  PMIX_QUERY_RESULTS "pmix.qry.res" (pmix_data_array_t)
13    Contains an array of query results for a given pmix_query_t passed to the
14    PMIx_Query_info APIs. If qualifiers were included in the query, then the first element
15    of the array shall be the PMIX_QUERY_QUALIFIERS attribute containing those qualifiers.
16    Each of the remaining elements of the array is a pmix_info_t containing the query key
17    and the corresponding value returned by the query. This attribute is solely for reporting
18    purposes and cannot be used in PMIx_Get or other query operations.

19  PMIX_QUERY_QUALIFIERS "pmix.qry.quals" (pmix_data_array_t)
20    Contains an array of qualifiers that were included in the query that produced the provided
21    results. This attribute is solely for reporting purposes and cannot be used in PMIx_Get or
22    other query operations.

23  PMIX_QUERY_SUPPORTED_KEYS "pmix.qry.keys" (char*)
24    Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS.

25  PMIX_QUERY_SUPPORTED_QUALIFIERS "pmix.qry.quals" (char*)
26    Return comma-delimited list of qualifiers supported by a query on the provided key, instead
27    of actually performing the query on the key. NO QUALIFIERS.

28  PMIX_QUERY_NAMESPACE_INFO "pmix.qry.nsinfo" (pmix_data_array_t*)
29    Return an array of active namespace information - each element will itself contain an array
30    including the namespace plus the command line of the application executing within it.
31    OPTIONAL QUALIFIERS: PMIX_NSPACE of specific namespace whose info is being
32    requested

33  PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
34    Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or
35    more of PMIX_CLIENT_FUNCTIONS, PMIX_SERVER_FUNCTIONS,
36    PMIX_TOOL_FUNCTIONS, and PMIX_HOST_FUNCTIONS.

37  PMIX_QUERY_AVAIL_SERVERS "pmix.qry.asrvrs" (pmix_data_array_t*)

```

```

1      Return an array of pmix_info_t, each element itself containing a
2      PMIX_SERVER_INFO_ARRAY entry holding all available data for a server on this node to
3      which the caller might be able to connect.
4      PMIX_SERVER_INFO_ARRAY "pmix.srv.arr" (pmix_data_array_t)
5          Array of pmix_info_t about a given server, starting with its PMIX_NSPACE and
6          including at least one of the rendezvous-required pieces of information.
7      PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (bool)
8          Request a list of functions supported by the PMIx client library.
9      PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
10         Request attributes supported by the PMIx client library.
11     PMIX_SERVER_FUNCTIONS "pmix.srvr.fns" (bool)
12         Request a list of functions supported by the PMIx server library.
13     PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
14         Request attributes supported by the PMIx server library.
15     PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (bool)
16         Request a list of functions supported by the host environment.
17     PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
18         Request attributes supported by the host environment.
19     PMIX_TOOL_FUNCTIONS "pmix.tool.fns" (bool)
20         Request a list of functions supported by the PMIx tool library.
21     PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
22         Request attributes supported by the PMIx tool library functions.

23 Server attributes
24     PMIX_TOPOLOGY2 "pmix.topo2" (pmix_topology_t)
25         Provide a pointer to an implementation-specific description of the local node topology.
26     PMIX_SERVER_SHARE_TOPOLOGY "pmix.srvr.share" (bool)
27         The PMIx server is to share its copy of the local node topology (whether given to it or
28         self-discovered) with any clients.
29     PMIX_SERVER_SESSION_SUPPORT "pmix.srvr.sess" (bool)
30         The host RM wants to declare itself as being the local session server for PMIx connection
31         requests.
32     PMIX_SERVER_START_TIME "pmix.srvr.strtime" (char*)
33         Time when the server started - i.e., when the server created it's rendezvous file (given in
34         ctime string format).
35     PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
36         Server is supporting system scheduler and desires access to appropriate services.

```

```

1   PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
2     Provide an array of pmix_info_t containing job-realm information. The
3     PMIX_SESSION_ID attribute of the session containing the job is required to be included in
4     the array whenever the PMIx server library may host multiple sessions (e.g., when executing
5     with a host RM daemon). As information is registered one job (aka namespace) at a time via
6     the PMIx_server_register_nspace API, there is no requirement that the array
7     contain either the PMIX_NSPACE or PMIX_JOBID attributes when used in that context
8     (though either or both of them may be included). At least one of the job identifiers must be
9     provided in all other contexts where the job being referenced is ambiguous.

10  PMIX_APP_INFO_ARRAY "pmix.app.arr" (pmix_data_array_t)
11    Provide an array of pmix_info_t containing application-realm information. The
12    PMIX_NSPACE or PMIX_JOBID attributes of the job containing the application, plus its
13    PMIX_APPNUM attribute, must to be included in the array when the array is not included as
14    part of a call to PMIx_server_register_nspace - i.e., when the job containing the
15    application is ambiguous. The job identification is otherwise optional.

16  PMIX_PROC_INFO_ARRAY "pmix pdata" (pmix_data_array_t)
17    Provide an array of pmix_info_t containing process-realm information. The
18    PMIX_RANK and PMIX_NSPACE attributes, or the PMIX_PROCID attribute, are required
19    to be included in the array when the array is not included as part of a call to
20    PMIx_server_register_nspace - i.e., when the job containing the process is
21    ambiguous. All three may be included if desired. When the array is included in some
22    broader structure that identifies the job, then only the PMIX_RANK or the PMIX_PROCID
23    attribute must be included (the others are optional).

24  PMIX_NODE_INFO_ARRAY "pmix.node.arr" (pmix_data_array_t)
25    Provide an array of pmix_info_t containing node-realm information. At a minimum,
26    either the PMIX_NODEID or PMIX_HOSTNAME attribute is required to be included in the
27    array, though both may be included.

28  PMIX_MAX_VALUE "pmix.descr.maxval" (varies)
29    Used in pmix_regattr_t to describe the maximum valid value for the associated
30    attribute.

31  PMIX_MIN_VALUE "pmix.descr.minval" (varies)
32    Used in pmix_regattr_t to describe the minimum valid value for the associated
33    attribute.

34  PMIX_ENUM_VALUE "pmix.descr.enum" (char*)
35    Used in pmix_regattr_t to describe accepted values for the associated attribute.
36    Numerical values shall be presented in a form convertible to the attribute's declared data
37    type. Named values (i.e., values defined by constant names via a typical C-language enum
38    declaration) must be provided as their numerical equivalent.

```

1           **Job-Mgmt attributes**

2       **PMIX\_ALLOC\_ID** "pmix.alloc.id" (**char\***)

3           A string identifier (provided by the host environment) for the resulting allocation which can  
4           later be used to reference the allocated resources in, for example, a call to **PMIx\_Spawn**.

5       **PMIX\_ALLOC\_QUEUE** "pmix.alloc.queue" (**char\***)

6           Name of the WLM queue to which the allocation request is to be directed, or the queue being  
7           referenced in a query.

8           **Publish attributes**

9       **PMIX\_ACCESS\_PERMISSIONS** "pmix.aperms" (**pmix\_data\_array\_t**)

10          Define access permissions for the published data. The value shall contain an array of  
11          **pmix\_info\_t** structs containing the specified permissions.

12       **PMIX\_ACCESS\_USERIDS** "pmix.auids" (**pmix\_data\_array\_t**)

13          Array of effective UIDs that are allowed to access the published data.

14       **PMIX\_ACCESS\_GRPIDS** "pmix.agids" (**pmix\_data\_array\_t**)

15          Array of effective GIDs that are allowed to access the published data.

16           **Reserved keys**

17       **PMIX\_NUM\_ALLOCATED\_NODES** "pmix.num.anodes" (**uint32\_t**)

18          Number of nodes in the specified realm regardless of whether or not they currently host  
19          processes.

20       **PMIX\_NUM\_NODES** "pmix.num.nodes" (**uint32\_t**)

21          Number of nodes currently hosting processes in the specified realm.

22       **PMIX\_CMD\_LINE** "pmix.cmd.line" (**char\***)

23          Command line used to execute the specified job (e.g., "mpirun -n 2 –map-by foo ./myapp : -n  
24          4 ./myapp2").

25       **PMIX\_APP\_ARGV** "pmix.app.argv" (**char\***)

26          Consolidated argv passed to the spawn command for the given application (e.g., "./myapp  
27          arg1 arg2 arg3").

28       **PMIX\_PACKAGE\_RANK** "pmix.pkgrank" (**uint16\_t**)

29          Rank of the specified process on the *package* where this process resides - refers to the  
30          numerical location (starting from zero) of the process on its package when counting only  
31          those processes from the same job that share the package, ordered by their overall rank  
32          within that job. Note that processes that are not bound to PUs within a single specific  
33          package cannot have a package rank.

34       **PMIX\_REINCARNATION** "pmix.reinc" (**uint32\_t**)

35          Number of times this process has been re-instantiated - i.e, a value of zero indicates that the  
36          process has never been restarted.

37       **PMIX\_HOSTNAME\_ALIASES** "pmix.alias" (**char\***)

1 Comma-delimited list of names by which the target node is known.

2 **PMIX\_HOSTNAME\_KEEP\_FQDN** "pmix.fqdn" (bool)  
3 FQDNs are being retained by the PMIx library.

4 **Tool attributes**

5 **PMIX\_TOOL\_CONNECT\_OPTIONAL** "pmix.tool.conopt" (bool)  
6 The tool shall connect to a server if available, but otherwise continue to operate  
7 unconnected.

8 **PMIX\_TOOL\_ATTACHMENT\_FILE** "pmix.tool.attach" (char\*)  
9 Pathname of file containing connection information to be used for attaching to a specific  
10 server.

11 **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** "pmix.tool.lncrnd" (char\*)  
12 Pathname of file where the launcher is to store its connection information so that the  
13 spawning tool can connect to it.

14 **PMIX\_PRIMARY\_SERVER** "pmix.pri.srvr" (bool)  
15 The server to which the tool is connecting shall be designated the *primary* server once  
16 connection has been accomplished.

17 **PMIX\_NOHUP** "pmix.nohup" (bool)  
18 Any processes started on behalf of the calling tool (or the specified namespace, if such  
19 specification is included in the list of attributes) should continue after the tool disconnects  
20 from its server.

21 **PMIX\_LAUNCHER\_DAEMON** "pmix.lnch.dmn" (char\*)  
22 Path to executable that is to be used as the backend daemon for the launcher. This replaces  
23 the launcher's own daemon with the specified executable. Note that the user is therefore  
24 responsible for ensuring compatibility of the specified executable and the host launcher.

25 **PMIX\_FORKEXEC\_AGENT** "pmix.frkex.agnt" (char\*)  
26 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
27 application processes. The fork/exec agent shall connect back (as a PMIx tool) to the  
28 launcher's daemon to receive its spawn instructions, and is responsible for starting the actual  
29 application process it replaced. See Section 17.4.3 for details.

30 **PMIX\_EXEC\_AGENT** "pmix.exec.agnt" (char\*)  
31 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
32 application processes. The launcher's daemon shall pass the full command line of the  
33 application on the command line of the exec agent, which shall not connect back to the  
34 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
35 process in its own place. See Section 17.4.3 for details.

36 **PMIX\_IOF\_PUSH\_STDIN** "pmix.iof.stdin" (bool)

```

1 Requests that the PMIx library collect the stdin of the requester and forward it to the
2 processes specified in the PMIX_IOF_PUSH call. All collected data is sent to the same
3 targets until stdin is closed, or a subsequent call to PMIX_IOF_PUSH is made that
4 includes the PMIX_IOF_COMPLETE attribute indicating that forwarding of stdin is to be
5 terminated.

6 PMIX_IOF_COPY "pmix.iof.cpy" (bool)
7 Requests that the host environment deliver a copy of the specified output stream(s) to the
8 tool, letting the stream(s) continue to also be delivered to the default location. This allows the
9 tool to tap into the output stream(s) without redirecting it from its current final destination.

10 PMIX_IOF_REDIRECT "pmix.iof.redir" (bool)
11 Requests that the host environment intercept the specified output stream(s) and deliver it to
12 the requesting tool instead of its current final destination. This might be used, for example,
13 during a debugging procedure to avoid injection of debugger-related output into the
14 application's results file. The original output stream(s) destination is restored upon
15 termination of the tool.

16 PMIX_DEBUG_TARGET "pmix.dbg.tgt" (pmix_proc_t*)
17 Identifier of process(es) to be debugged - a rank of PMIX_RANK_WILDCARD indicates that
18 all processes in the specified namespace are to be included.

19 PMIX_DEBUG_DAEMONS_PER_PROC "pmix.dbg.dpproc" (uint16_t)
20 Number of debugger daemons to be spawned per application process. The launcher is to pass
21 the identifier of the namespace to be debugged by including the PMIX_DEBUG_TARGET
22 attribute in the daemon's job-level information. The debugger daemons spawned on a given
23 node are responsible for self-determining their specific target process(es) - e.g., by
24 referencing their own PMIX_LOCAL_RANK in the daemon debugger job versus the
25 corresponding PMIX_LOCAL_RANK of the target processes on the node.

26 PMIX_DEBUG_DAEMONS_PER_NODE "pmix.dbg.dpnd" (uint16_t)
27 Number of debugger daemons to be spawned on each node where the target job is executing.
28 The launcher is to pass the identifier of the namespace to be debugged by including the
29 PMIX_DEBUG_TARGET attribute in the daemon's job-level information. The debugger
30 daemons spawned on a given node are responsible for self-determining their specific target
31 process(es) - e.g., by referencing their own PMIX_LOCAL_RANK in the daemon debugger
32 job versus the corresponding PMIX_LOCAL_RANK of the target processes on the node.

33 Fabric attributes
34 PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
35 Server is supporting system scheduler and desires access to appropriate services.

36 PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer)
37 Pointer to a two-dimensional square array of point-to-point relative communication costs
38 expressed as uint16_t values.

39 PMIX_FABRIC_GROUPS "pmix.fab.grp" (string)

```

```

1 A string delineating the group membership of nodes in the overall system, where each fabric
2 group consists of the group number followed by a colon and a comma-delimited list of nodes
3 in that group, with the groups delimited by semi-colons (e.g.,
4 0:node000,node002,node004,node006;1:node001,node003,
5 node005,node007)

6 PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string)
7 Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

8 PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)
9 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

10 PMIX_FABRIC_INDEX "pmix.fab.idx" (size_t)
11 The index of the fabric as returned in pmix_fabric_t.

12 PMIX_FABRIC_NUM_DEVICES "pmix.fab.nverts" (size_t)
13 Total number of fabric devices in the overall system - corresponds to the number of rows or
14 columns in the cost matrix.

15 PMIX_FABRIC_COORDINATES "pmix.fab.coords" (pmix_data_array_t)
16 Array of pmix_geometry_t fabric coordinates for devices on the specified node. The
17 array will contain the coordinates of all devices on the node, including values for all
18 supported coordinate views. The information for devices on the local node shall be provided
19 if the node is not specified in the request.

20 PMIX_FABRIC_DIMS "pmix.fab.dims" (uint32_t)
21 Number of dimensions in the specified fabric plane/view. If no plane is specified in a
22 request, then the dimensions of all planes in the overall system will be returned as a
23 pmix_data_array_t containing an array of uint32_t values. Default is to provide
24 dimensions in logical view.

25 PMIX_FABRIC_ENDPT "pmix.fab.endpt" (pmix_data_array_t)
26 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given
27 process (e.g., in the case where multiple devices are associated with a package to which the
28 process is bound), the returned values will be provided in a pmix_data_array_t of
29 pmix_endpoint_t elements.

30 PMIX_FABRIC_DEVICE_DIST "pmix.fab.endptdist" (pmix_data_array_t)
31 Relative distance from the location of the calling process (either as sampled at the time of a
32 PMIx_Fabric_update_distances request, or based on the initial binding location
33 set at time of start of execution) to each local fabric device, returned as an array of
34 pmix_device_distance_t elements in no particular order.

35 PMIX_FABRIC_SHAPE "pmix.fab.shape" (pmix_data_array_t*)
36 The size of each dimension in the specified fabric plane/view, returned in a
37 pmix_data_array_t containing an array of uint32_t values. The size is defined as
38 the number of elements present in that dimension - e.g., the number of devices in one
39 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of

```

1 each plane in the overall system will be returned in a `pmix_data_array_t` array where  
2 each element is itself a two-element array containing the `PMIX_FABRIC_PLANE` followed  
3 by that plane's fabric shape. Default is to provide the shape in *logical* view.

4 **PMIX\_FABRIC\_SHAPE\_STRING** "pmix.fab.shapestr" (`string`)

5 Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the  
6 shape of each plane in the overall system will be returned in a `pmix_data_array_t` array  
7 where each element is itself a two-element array containing the `PMIX_FABRIC_PLANE`  
8 followed by that plane's fabric shape string. Default is to provide the shape in *logical* view.

9 **PMIX\_SWITCH\_PEERS** "pmix.speers" (`pmix_data_array_t`)

10 Peer ranks that share the same switch as the process specified in the call to `PMIx_Get`.

11 Returns a `pmix_data_array_t` array of `pmix_info_t` results, each element  
12 containing the `PMIX_SWITCH_PEERS` key with a three-element `pmix_data_array_t`  
13 array of `pmix_info_t` containing the `PMIX_FABRIC_DEVICE_ID` of the local fabric  
14 device, the `PMIX_FABRIC_SWITCH` identifying the switch to which it is connected, and a  
15 comma-delimited string of peer ranks sharing the switch to which that device is connected.

16 **PMIX\_FABRIC\_PLANE** "pmix.fab.plane" (`string`)

17 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
18 for information, specifies the plane whose information is to be returned. When used directly  
19 as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
20 planes in the overall system.

21 **PMIX\_FABRIC\_SWITCH** "pmix.fab.switch" (`string`)

22 ID string of a fabric switch. When used as a modifier in a request for information, specifies  
23 the switch whose information is to be returned. When used directly as a key in a request,  
24 returns a `pmix_data_array_t` of string identifiers for all fabric switches in the overall  
25 system.

26 **PMIX\_FABRIC\_DEVICE** "pmix.fabdev" (`pmix_data_array_t`)

27 An array of `pmix_info_t` describing a particular fabric device using one or more of the  
28 attributes defined below. The first element in the array shall be the  
29 `PMIX_FABRIC_DEVICE_ID` of the device.

30 **PMIX\_FABRIC\_DEVICE\_ID** "pmix.fabdev.id" (`string`)

31 System-wide UUID of a particular fabric device.

32 **PMIX\_FABRIC\_DEVICE\_INDEX** "pmix.fabdev.idx" (`uint32_t`)

33 Index of the device within an associated communication cost matrix.

34 **PMIX\_FABRIC\_DEVICE\_NAME** "pmix.fabdev.nm" (`string`)

35 The operating system name associated with the device. This may be a logical fabric interface  
36 name (e.g. "eth0" or "eno1") or an absolute filename.

37 **PMIX\_FABRIC\_DEVICE\_VENDOR** "pmix.fabdev.vndr" (`string`)

38 Indicates the name of the vendor that distributes the device.

```

1   PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
2     The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
3   PMIX_FABRIC_DEVICE_VENDORID "pmix.fabdev.vendid" (string)
4     This is a vendor-provided identifier for the device or product.
5   PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string)
6     The name of the driver associated with the device.
7   PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string)
8     The device's firmware version.
9   PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)
10    The primary link-level address associated with the device, such as a MAC address. If
11    multiple addresses are available, only one will be reported.
12  PMIX_FABRIC_DEVICE_COORDINATES "pmix.fab.coord" (pmix_geometry_t)
13    The pmix_geometry_t fabric coordinates for the device, including values for all
14    supported coordinate views.
15  PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
16    The maximum transfer unit of link level frames or packets, in bytes.
17  PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
18    The active link data rate, given in bits per second.
19  PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" (pmix_link_state_t)
20    The last available physical port state for the specified device. Possible values are
21    PMIX_LINK_STATE_UNKNOWN, PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate
22    if the port state is unknown or not applicable (unknown), inactive (down), or active (up).
23  PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
24    Specifies the type of fabric interface currently active on the device, such as Ethernet or
25    InfiniBand.
26  PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
27    A node-level unique identifier for a PCI device. Provided only if the device is located on a
28    PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
29    the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
30    zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
31    combination of node identifier (PMIX_HOSTNAME or PMIX_NODEID) and
32    PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the overall system.

```

### 33 **Sets-Groups attributes**

```

34  PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
35    Return the number of process sets defined in the specified range (defaults to
36    PMIX_RANGE_SESSION).
37  PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)

```

```

1      Return a pmix_data_array_t containing an array of strings of the process set names
2      defined in the specified range (defaults to PMIX_RANGE_SESSION).
3      PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t*)
4          Return an array of pmix_proc_t containing the members of the specified process set.
5      PMIX_PSET_NAME "pmix.pset.nm" (char*)
6          The name of the newly defined process set.
7      PMIX_PSET_MEMBERS "pmix.pset.mems" (pmix_data_array_t*)
8          An array of pmix_proc_t containing the members of the newly defined process set.
9      PMIX_PSET_NAMES "pmix.pset.nms" (pmix_data_array_t*)
10         Returns an array of char* string names of the process sets in which the given process is a
11         member.
12     PMIX_QUERY_NUM_GROUPS "pmix.qry.pgrpnum" (size_t)
13         Return the number of process groups defined in the specified range (defaults to session).
14         OPTIONAL QUALIFERS: PMIX_RANGE.
15     PMIX_QUERY_GROUP_NAMES "pmix.qry.pgrp" (pmix_data_array_t*)
16         Return a pmix_data_array_t containing an array of string names of the process groups
17         defined in the specified range (defaults to session). OPTIONAL QUALIFERS:
18         PMIX_RANGE.
19     PMIX_QUERY_GROUP_MEMBERSHIP
20     "pmix.qry.pgrpmems" (pmix_data_array_t*)
21         Return a pmix_data_array_t of pmix_proc_t containing the members of the
22         specified process group. REQUIRED QUALIFIERS: PMIX_GROUP_ID.
23     PMIX_GROUP_ID "pmix.grp.id" (char*)
24         User-provided group identifier - as the group identifier may be used in PMIx operations, the
25         user is required to ensure that the provided ID is unique within the scope of the host
26         environment (e.g., by including some user-specific or application-specific prefix or suffix to
27         the string).
28     PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)
29         This process is the leader of the group.
30     PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)
31         Participation is optional - do not return an error if any of the specified processes terminate
32         without having joined. The default is false.
33     PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)
34         Notify remaining members when another member terminates without first leaving the group.
35
36     PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)

```

```

1     Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective
2     operation.

3     PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)
4     Requests that the RM assign a new context identifier to the newly created group. The
5     identifier is an unsigned, size_t value that the RM guarantees to be unique across the range
6     specified in the request. Thus, the value serves as a means of identifying the group within
7     that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.

8     PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)
9     Group operation only involves local processes. PMIx implementations are required to
10    automatically scan an array of group members for local vs remote processes - if only local
11    processes are detected, the implementation need not execute a global collective for the
12    operation unless a context ID has been requested from the host environment. This can result
13    in significant time savings. This attribute can be used to optimize the operation by indicating
14    whether or not only local processes are represented, thus allowing the implementation to
15    bypass the scan.

16    PMIX_GROUP_CONTEXT_ID "pmix.grp.ctxid" (size_t)
17    Context identifier assigned to the group by the host RM.

18    PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)
19    Data collected during group construction to ensure communication between group members
20    is supported upon completion of the operation.

21    PMIX_GROUP_NAMES "pmix.pgrp.nm" (pmix_data_array_t*)
22    Returns an array of char* string names of the process groups in which the given process is
23    a member.

24 Process Mgmt attributes
25     PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
26     Direct output into files of form "<directory>/<jobid>/rank.<rank>/"
27     stdout [err] - can be assigned to the entire job (by including attribute in the job_info
28     array) or on a per-application basis in the info array for each pmix_app_t.

29     PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)
30     Include process stacktraces in timeout report from a job.

31     PMIX_TIMEOUT_REPORT_STATE "pmix.tim.state" (bool)
32     Report process states in timeout report from a job.

33     PMIX_NOTIFY_JOB_EVENTS "pmix.note.jev" (bool)
34     Requests that the launcher generate the PMIX_EVENT_JOB_START,
35     PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events. Each event is to
36     include at least the namespace of the corresponding job and a PMIX_EVENT_TIMESTAMP
37     indicating the time the event occurred. Note that the requester must register for these
38     individual events, or capture and process them by registering a default event handler instead
39     of individual handlers and then process the events based on the returned status code. Another

```

common method is to register one event handler for all job-related events, with a separate  
 handler for non-job events - see [PMIx\\_Register\\_event\\_handler](#) for details.  
  
 1           **PMIX\_NOTIFY\_PROC\_TERMINATION** "pmix.noteproc" (bool)  
 2           Requests that the launcher generate the **PMIX\_EVENT\_PROC\_TERMINATED** event  
 3           whenever a process either normally or abnormally terminates.  
  
 4           **PMIX\_NOTIFY\_PROC\_ABNORMAL\_TERMINATION** "pmix.noteabproc" (bool)  
 5           Requests that the launcher generate the **PMIX\_EVENT\_PROC\_TERMINATED** event only  
 6           when a process abnormally terminates.  
  
 7           **PMIX\_LOG\_PROC\_TERMINATION** "pmix.logproc" (bool)  
 8           Requests that the launcher log the **PMIX\_EVENT\_PROC\_TERMINATED** event whenever a  
 9           process either normally or abnormally terminates.  
  
 10          **PMIX\_LOG\_PROC\_ABNORMAL\_TERMINATION** "pmix.logabproc" (bool)  
 11          Requests that the launcher log the **PMIX\_EVENT\_PROC\_TERMINATED** event only when a  
 12           process abnormally terminates.  
  
 13          **PMIX\_LOG\_JOB\_EVENTS** "pmix.log.jev" (bool)  
 14          Requests that the launcher log the **PMIX\_EVENT\_JOB\_START**,  
 15           **PMIX\_LAUNCH\_COMPLETE**, and **PMIX\_EVENT\_JOB\_END** events using **PMIx\_Log**,  
 16           subject to the logging attributes of Section 12.4.3.  
  
 17          **PMIX\_LOG\_COMPLETION** "pmix.logcomp" (bool)  
 18          Requests that the launcher log the **PMIX\_EVENT\_JOB\_END** event for normal or abnormal  
 19           termination of the spawned job using **PMIx\_Log**, subject to the logging attributes of  
 20           Section 12.4.3. The event shall include the returned status code  
 21           (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job; the identity (**PMIX\_PROCID**)  
 22           and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if applicable; and a  
 23           **PMIX\_EVENT\_TIMESTAMP** indicating the time the termination occurred.  
  
 24          **PMIX\_FIRST\_ENVAR** "pmix.environ.first" (pmix\_envar\_t\*)  
 25          Ensure the given value appears first in the specified envar using the separator character,  
 26           creating the envar if it doesn't already exist  
  
 27          **Event attributes**  
 28          **PMIX\_EVENT\_TIMESTAMP** "pmix.evtstamp" (time\_t)  
 29           System time when the associated event occurred.

## 32       B.8.3   Added Environmental Variables

33          **Tool environmental variables**  
 34          **PMIX\_LAUNCHER\_PAUSE\_FOR\_TOOL**  
 35          **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE**  
 36

## 1 B.8.4 Deprecated APIs

2       `pmix_evhdlr_reg_cfunc_t` Renamed to `pmix_hdlr_reg_cfunc_t`  
3       The `pmix_server_client_connected_fn_t` server module entry point has been  
4       *deprecated* in favor of `pmix_server_client_connected2_fn_t`  
5       `PMIx_tool_connect_to_server` Replaced by `PMIx_tool_attach_to_server` to  
6       allow return of the process identifier of the server to which the tool has attached.

## 7 B.8.5 Deprecated constants

8       The following constants were deprecated in v4.0:

9       `PMIX_ERR_DEBUGGER_RELEASE`      Renamed to `PMIX_DEBUGGER_RELEASE`  
10      `PMIX_ERR_JOB_TERMINATED`      Renamed to `PMIX_EVENT_JOB_END`  
11      `PMIX_EXISTS`      Renamed to `PMIX_ERR_EXISTS`  
12      `PMIX_ERR_PROC_ABORTED`      Consolidated with `PMIX_EVENT_PROC_TERMINATED`  
13      `PMIX_ERR_PROC_ABORTING`      Consolidated with `PMIX_EVENT_PROC_TERMINATED`  
14      `PMIX_ERR_LOST_CONNECTION_TO_SERVER`      Consolidated into  
15              `PMIX_ERR_LOST_CONNECTION`  
16      `PMIX_ERR_LOST_PEER_CONNECTION`      Consolidated into  
17              `PMIX_ERR_LOST_CONNECTION`  
18      `PMIX_ERR_LOST_CONNECTION_TO_CLIENT`      Consolidated into  
19              `PMIX_ERR_LOST_CONNECTION`  
20      `PMIX_ERR_INVALID_TERMINATION`      Renamed to `PMIX_ERR_JOB_TERM_WO_SYNC`  
21      `PMIX_PROC_TERMINATED`      Renamed to `PMIX_EVENT_PROC_TERMINATED`  
22      `PMIX_ERR_NODE_DOWN`      Renamed to `PMIX_EVENT_NODE_DOWN`  
23      `PMIX_ERR_NODE_OFFLINE`      Renamed to `PMIX_EVENT_NODE_OFFLINE`  
24      `PMIX_ERR_SYS_OTHER`      Renamed to `PMIX_EVENT_SYS_OTHER`  
25      `PMIX_CONNECT_REQUESTED`      Connection has been requested by a PMIx-based tool -  
26        deprecated as not required.  
27      `PMIX_PROC_HAS_CONNECTED`      A tool or client has connected to the PMIx server -  
28        deprecated in favor of the new `pmix_server_client_connected2_fn_t` server  
29        module API

## 30 B.8.6 Removed constants

31       The following constants were removed from the PMIx Standard in v4.0 as they are internal to a  
32        particular PMIx implementation.

33      `PMIX_ERR_HANDSHAKE_FAILED`      Connection handshake failed  
34      `PMIX_ERR_READY_FOR_HANDSHAKE`      Ready for handshake  
35      `PMIX_ERR_IN_ERRNO`      Error defined in `errno`  
36      `PMIX_ERR_INVALID_VAL_LENGTH`      Invalid value length

```

1   PMIX_ERR_INVALID_LENGTH      Invalid argument length
2   PMIX_ERR_INVALID_NUM_ARGS    Invalid number of arguments
3   PMIX_ERR_INVALID_ARGS       Invalid arguments
4   PMIX_ERR_INVALID_NUM_PARSED Invalid number parsed
5   PMIX_ERR_INVALID_KEYVALP    Invalid key/value pair
6   PMIX_ERR_INVALID_SIZE       Invalid size
7   PMIX_ERR_PROC_REQUESTED_ABORT Process is already requested to abort
8   PMIX_ERR_SERVER_FAILED_REQUEST Failed to connect to the server
9   PMIX_ERR_PROC_ENTRY_NOT_FOUND Process not found
10  PMIX_ERR_INVALID_ARG        Invalid argument
11  PMIX_ERR_INVALID_KEY        Invalid key
12  PMIX_ERR_INVALID_KEY_LENGTH Invalid key length
13  PMIX_ERR_INVALID_VAL       Invalid value
14  PMIX_ERR_INVALID_NAMESPACE  Invalid namespace
15  PMIX_ERR_SERVER_NOT_AVAIL  Server is not available
16  PMIX_ERR_SILENT            Silent error
17  PMIX_ERR_PACK_MISMATCH     Pack mismatch
18  PMIX_ERR_DATA_VALUE_NOT_FOUND Data value not found
19  PMIX_ERR_NOT_IMPLEMENTED   Not implemented
20  PMIX_GDS_ACTION_COMPLETE   The Global Data Storage (GDS) action has completed
21  PMIX_NOTIFY_ALLOC_COMPLETE Notify that a requested allocation operation is complete
22      - the result of the request will be included in the info array

```

## B.8.7 Deprecated attributes

The following attributes were deprecated in v4.0:

```

25  PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
26      Renamed to PMIX_TOPOLOGY2.
27  PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
28      Renamed to PMIX_DEBUG_TARGET)
29  PMIX_RECONNECT_SERVER "pmix.tool.recon" (bool)
30      Renamed to the PMIx_tool_connect_to_server API
31  PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)
32      Renamed to PMIX_ALLOC_FABRIC
33  PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
34      Renamed to PMIX_ALLOC_FABRIC_ID
35  PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
36      Renamed to PMIX_ALLOC_FABRIC_QOS
37  PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)
38      Renamed to PMIX_ALLOC_FABRIC_TYPE
39  PMIX_ALLOC_NETWORK_PLANE "pmix.alloc.netplane" (char*)
40      Renamed to PMIX_ALLOC_FABRIC_PLANE
41  PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)

```

```

1           Renamed to PMIX_ALLOC_FABRIC_ENDPTS
2 PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
3           Renamed to PMIX_ALLOC_FABRIC_ENDPTS_NODE
4 PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
5           Renamed to PMIX_ALLOC_FABRIC_SEC_KEY
6 PMIX_PROC_DATA "pmix_pdata" (pmix_data_array_t)
7           Renamed to PMIX_PROC_INFO_ARRAY
8 PMIX_LOCALITY "pmix.loc" (pmix_locality_t)
9           Relative locality of the specified process to the requester, expressed as a bitmask as per the
10          description in the pmix_locality_t section. This value is unique to the requesting
11          process and thus cannot be communicated by the server as part of the job-level information.
12          Its use has been replaced by the PMIx_Get_relative_locality function.

```

## 13 B.8.8 Removed attributes

14 The following attributes were removed from the PMIx Standard in v4.0 as they are internal to a  
15 particular PMIx implementation. Users are referred to the **PMIx\_Load\_topology** API for  
16 obtaining the local topology description.

```

17 PMIX_LOCAL_TOPO "pmix.ltopo" (char*)
18           XML representation of local node topology.
19 PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*)
20           XML-based description of topology
21 PMIX_TOPOLOGY_FILE "pmix.topo.file" (char*)
22           Full path to file containing XML topology description
23 PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
24           Topology signature string.
25 PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
26           Address of the HWLOC shared memory segment.
27 PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
28           Size of the HWLOC shared memory segment.
29 PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
30           Path to the HWLOC shared memory file.
31 PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
32           XML representation of local topology using HWLOC's v1.x format.
33 PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
34           XML representation of local topology using HWLOC's v2.x format.
35 PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)
36           Share the HWLOC topology via shared memory
37 PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)
38           Kind of VM "hole" HWLOC should use for shared memory
39 PMIX_DSTPATH "pmix.dstpath" (char*)
40           Path to shared memory data storage (dstore) files. Deprecated from Standard as being
41           implementation specific.

```

```
1 PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)
2     Comma-delimited list of algorithms to use for the collective operation. PMIx does not
3     impose any requirements on a host environment's collective algorithms. Thus, the
4     acceptable values for this attribute will be environment-dependent - users are encouraged to
5     check their host environment for supported values.
6 PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)
7     If true, indicates that the requested choice of algorithm is mandatory. This attribute was
8     deprecated in v3.0
9 PMIX_PROC_BLOB "pmix.pblob" (pmix_byte_object_t)
10    Packed blob of process data.
11 PMIX_MAP_BLOB "pmix.mblob" (pmix_byte_object_t)
12    Packed blob of process location.
13 PMIX_MAPPER "pmix.mapper" (char*)
14    Mapping mechanism to use for placing spawned processes - when accessed using
15    PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping
16    mechanism used for the provided namespace.
17 PMIX_NON_PMI "pmix.nonpmi" (bool)
18    Spawns processes will not call PMIx_Init.
19 PMIX_PROC_URI "pmix.puri" (char*)
20    URI containing contact information for the specified process.
21 PMIX_ARCH "pmix.arch" (uint32_t)
22    Architecture flag.
```

## APPENDIX C

# Acknowledgements

---

This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.

## C.1 Version 4.0

The following list includes some of the active participants in the PMIx v4 standardization process.

- ...

The following institutions supported this effort through time and travel support for the people listed above.

- ...

## C.2 Version 3.0

The following list includes some of the active participants in the PMIx v3 standardization process.

- Ralph H. Castain, Andrew Friedley, Brandon Yates
- Joshua Hursey
- Aurelien Bouteiller and George Bosilca
- Dirk Schubert
- Kevin Harms

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- The Exascale Computing Project, an initiative of the US Department of Energy

- 1       • National Science Foundation  
2       • Argonne National Laboratory  
3       • Allinea (ARM)

4 **C.3 Version 2.0**

5       The following list includes some of the active participants in the PMIx v2 standardization process.

- 6       • Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm  
7       and Terry Wilmarth  
8       • Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi  
9       • Aurelien Bouteiller and George Bosilca  
10      • Artem Polyakov, Igor Ivanov and Boris Karasev  
11      • Gilles Gouaillardet  
12      • Michael A Raymond and Jim Stoffel  
13      • Dirk Schubert  
14      • Moe Jette  
15      • Takahiro Kawashima and Shinji Sumimoto  
16      • Howard Pritchard  
17      • David Beer  
18      • Brice Goglin  
19      • Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt  
20      • Adam Moody and Martin Schulz  
21      • Ryan Grant and Stephen Olivier  
22      • Michael Karo

23      The following institutions supported this effort through time and travel support for the people listed  
24      above.

- 25      • Intel Corporation  
26      • IBM, Inc.  
27      • University of Tennessee, Knoxville  
28      • The Exascale Computing Project, an initiative of the US Department of Energy  
29      • National Science Foundation

- 1     ● Mellanox, Inc.
- 2     ● Research Organization for Information Science and Technology
- 3     ● HPE Co.
- 4     ● Allinea (ARM)
- 5     ● SchedMD, Inc.
- 6     ● Fujitsu Limited
- 7     ● Los Alamos National Laboratory
- 8     ● Adaptive Solutions, Inc.
- 9     ● INRIA
- 10    ● Oak Ridge National Laboratory
- 11    ● Lawrence Livermore National Laboratory
- 12    ● Sandia National Laboratory
- 13    ● Altair

## 14   **C.4 Version 1.0**

15   The following list includes some of the active participants in the PMIx v1 standardization process.

- 16   ● Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- 17   ● Joshua Hursey and David Solt
- 18   ● Aurelien Bouteiller and George Bosilca
- 19   ● Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- 20   ● Gilles Gouaillardet
- 21   ● Gary Brown
- 22   ● Moe Jette

23   The following institutions supported this effort through time and travel support for the people listed  
24   above.

- 25   ● Intel Corporation
- 26   ● IBM, Inc.
- 27   ● University of Tennessee, Knoxville
- 28   ● Mellanox, Inc.

- 1     ● Research Organization for Information Science and Technology
- 2     ● Adaptive Solutions, Inc.
- 3     ● SchedMD, Inc.

# Bibliography

---

- [1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMIx: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.
- [2] Balaji P. et al. PMI: A scalable parallel process-management interface for extreme-scale systems. In *Recent Advances in the Message Passing Interface*, EuroMPI '10, pages 31–41, Berlin, Heidelberg, 2010. Springer.

# Index

---

General terms and other items not induced in the other indices.

- application, [6](#), [88](#), [100](#), [292](#), [295](#), [297](#), [509](#), [512](#)
- attribute, [8](#)
- client, [7](#), [55](#)
- clients, [7](#)
- clone, [7](#)
- clones, [7](#), [63](#), [66](#), [172](#), [175](#), [176](#), [178](#), [194](#), [197](#), [518](#)
- data realm, [91](#), [257](#)
- data realms, [91](#)
- device, [8](#)
- devices, [8](#)
- Direct Modex, [243](#), [309](#)
- fabric, [8](#)
- fabric device, [8](#)
- fabric devices, [8](#)
- fabric plane, [8](#), [158](#), [163](#), [186](#), [189](#), [190](#), [262](#), [312](#)
- fabric planes, [8](#)
- fabrics, [8](#)
- host environment, [7](#)
- instant on, [8](#), [105](#), [242](#)
- job, [6](#), [7](#), [88](#), [95](#), [100](#), [284–286](#), [292–295](#), [297](#), [310](#), [313](#), [491](#), [509](#), [510](#), [512](#), [520](#)
- key, [8](#)
- namespace, [6](#)
- node, [7](#), [88](#), [100](#), [158](#), [163](#), [186](#), [189](#), [190](#), [292](#), [312](#)
- package, [7](#), [98](#), [521](#)
- peer, [7](#), [99](#), [289](#)
- peers, [7](#)
- process, [7](#), [88](#), [100](#), [158](#), [163](#), [186](#), [189](#), [190](#), [292](#), [312](#)
- processing unit, [8](#)

rank, [7](#), [298](#)  
realm, [91](#)  
realms, [91](#)  
resource manager, [7](#)  
RM, [7](#)  
  
scheduler, [7](#), [260](#)  
session, [6](#), [88](#), [100](#), [284](#), [292](#), [294](#), [509](#), [510](#), [512](#), [520](#)  
slot, [7](#)  
slots, [7](#)  
  
thread, [7](#)  
threads, [7](#)  
  
workflow, [7](#)  
workflows, [7](#), [381](#)

# Index of APIs

---

PMIx\_Abort, 23, [152](#), [153](#), [333](#), [334](#), [449](#), [464](#), [507](#)  
    PMIxClient.abort (Python), [463](#)

PMIx\_Alloc\_directive\_string, [53](#), [483](#), [508](#)  
    PMIxClient.alloc\_directive\_string (Python), [483](#)

PMIx\_Allocation\_request, [89](#), [183](#), [183](#), [189](#), [510](#), [512](#), [513](#)

PMIx\_Allocation\_request\_nb, [186](#), [189](#), [191](#), [472](#), [508](#)  
    PMIxClient.allocate (Python), [471](#)

PMIx\_Commit, [63](#), [65](#), [66](#), [106](#), [108](#), [108](#), [109](#), [309](#), [310](#), [336](#), [340](#), [465](#), [507](#)  
    PMIxClient.commit (Python), [465](#)

PMIx\_Connect, [171](#), [173](#), [175](#), [176](#), [178](#), [215–217](#), [422](#), [469](#), [507](#), [509](#)  
    PMIxClient.connect (Python), [468](#)

PMIx\_Connect\_nb, [173](#), [173](#), [507](#)

pmix\_connection\_cbfunc\_t, [362](#), [362](#)

pmix\_credential\_cbfunc\_t, [270](#), [379](#), [380](#)

PMIx\_Data\_copy, [149](#), [508](#)

PMIx\_Data\_copy\_payload, [150](#), [508](#)

PMIx\_Data\_pack, [145](#), [146](#), [282](#), [283](#), [508](#)

PMIx\_Data\_print, [149](#), [508](#)

PMIx\_Data\_range\_string, [52](#), [482](#), [508](#)  
    PMIxClient.data\_range\_string (Python), [482](#)

PMIx\_Data\_type\_string, [52](#), [483](#), [508](#)  
    PMIxClient.data\_type\_string (Python), [483](#)

PMIx\_Data\_unpack, [147](#), [508](#)

PMIx\_Deregister\_event\_handler, [137](#), [477](#), [508](#), [514](#)  
    PMIxClient.deregister\_event\_handler (Python), [477](#)

PMIx\_Disconnect, [175](#), [176–178](#), [217](#), [422](#), [469](#), [507](#), [509](#)  
    PMIxClient.disconnect (Python), [469](#)

PMIx\_Disconnect\_nb, [177](#), [178](#), [217](#), [507](#)

pmix\_dmodex\_response\_fn\_t, [308](#), [309](#)

PMIx\_Error\_string, [51](#), [481](#), [507](#)  
    PMIxClient.error\_string (Python), [480](#)

pmix\_event\_notification\_cbfunc\_fn\_t, [136](#), [142](#), [142](#)

pmix\_evhdlr\_reg\_cbfunc\_t (**Deprecated**), [530](#)

PMIx\_Fabric\_deregister, [264](#), [265](#), [479](#), [514](#)  
    PMIxClient.fabric\_deregister (Python), [479](#)

PMIx\_Fabric\_deregister\_nb, [265](#), [514](#)

PMIx\_Fabric\_register, [253](#), [261](#), [263](#), [478](#), [514](#)  
    PMIxClient.fabric\_register (Python), [478](#)

PMIx\_Fabric\_register\_nb, [262](#), [514](#)  
PMIx\_Fabric\_update, [262](#), [263](#), [264](#), [479](#), [514](#)  
    PMIxClient.fabric\_update (Python), [479](#)  
PMIx\_Fabric\_update\_distances, [260](#), [265](#), [267](#), [514](#), [524](#)  
PMIx\_Fabric\_update\_distances\_nb, [266](#), [514](#)  
PMIx\_Fabric\_update\_nb, [264](#), [514](#)  
PMIx\_Fence, [4](#), [62](#), [63](#), [64](#), [66](#), [105](#), [173](#), [176](#), [214](#), [223](#), [227](#), [243](#), [279](#), [309](#), [334](#), [337](#), [450](#), [466](#),  
    [507](#), [517](#)  
    PMIxClient.fence (Python), [465](#)  
PMIx\_Fence\_nb, [49](#), [64](#), [334](#), [337](#), [450](#), [507](#), [513](#)  
PMIx\_Finalize, [23](#), [58](#), [60](#), [61](#), [170](#), [332](#), [333](#), [422](#), [449](#), [463](#), [507](#)  
    PMIxClient.finalize (Python), [463](#)  
PMIx\_generate\_ppn, [282](#), [487](#), [507](#), [513](#)  
    PMIxServer.generate\_ppn (Python), [487](#)  
PMIx\_generate\_regex, [281](#), [283](#), [294](#), [487](#), [507](#), [513](#)  
    PMIxServer.generate\_regex (Python), [487](#)  
PMIx\_Get, [3](#), [8](#), [26](#), [58](#), [59](#), [63](#), [66](#), [67](#), [68](#)–[72](#), [75](#), [78](#), [80](#), [83](#), [88](#), [89](#), [91](#)–[95](#), [97](#)–[101](#), [105](#), [109](#), [155](#),  
    [156](#), [160](#)–[162](#), [164](#)–[166](#), [180](#)–[182](#), [189](#), [198](#), [204](#), [211](#), [213](#), [215](#), [219](#), [223](#), [243](#), [245](#), [257](#),  
    [260](#), [287](#), [291](#), [293](#), [324](#), [349](#), [351](#), [400](#), [423](#), [466](#), [507](#), [509](#), [517](#), [518](#), [525](#), [533](#)  
    PMIxClient.get (Python), [466](#)  
PMIx\_Get\_attribute\_name, [53](#), [485](#)  
    PMIxClient.get\_attribute\_name (Python), [485](#)  
PMIx\_Get\_attribute\_string, [53](#), [485](#)  
    PMIxClient.get\_attribute\_string (Python), [484](#)  
PMIx\_Get\_cpuset, [181](#), [325](#), [514](#)  
PMIx\_Get\_credential, [269](#), [271](#), [380](#), [473](#), [510](#), [513](#)  
    PMIxClient.get\_credential (Python), [473](#)  
PMIx\_Get\_credential\_nb, [270](#)  
PMIx\_Get\_nb, [49](#), [50](#), [69](#), [507](#)  
PMIx\_Get\_relative\_locality, [180](#), [181](#), [291](#), [324](#), [480](#), [514](#), [532](#)  
    PMIxClient.get\_relative\_locality (Python), [480](#)  
PMIx\_Get\_version, [10](#), [56](#), [463](#), [507](#)  
    PMIxClient.get\_version (Python), [463](#)  
PMIx\_Group\_construct, [214](#), [215](#), [220](#), [222](#), [223](#), [226](#), [474](#), [514](#)  
    PMIxClient.group\_construct (Python), [474](#)  
PMIx\_Group\_construct\_nb, [223](#), [226](#), [514](#)  
PMIx\_Group\_destruct, [217](#), [226](#), [227](#), [229](#), [239](#), [476](#), [514](#)  
    PMIxClient.group\_destruct (Python), [476](#)  
PMIx\_Group\_destruct\_nb, [227](#), [229](#), [514](#)  
PMIx\_Group\_invite, [216](#), [229](#), [231](#), [232](#), [234](#), [475](#), [514](#)  
    PMIxClient.group\_invite (Python), [475](#)  
PMIx\_Group\_invite\_nb, [232](#), [514](#)  
PMIx\_Group\_join, [216](#), [231](#), [232](#), [234](#), [235](#), [236](#)–[238](#), [476](#), [514](#)

PMIxClient.group\_join (Python), [475](#)  
    PMIx\_Group\_join\_nb, [234](#), [237](#), [238](#), [514](#)  
    PMIx\_Group\_leave, [217](#), [238](#), [239–241](#), [476](#), [514](#)  
        PMIxClient.group\_leave (Python), [476](#)  
    PMIx\_Group\_leave\_nb, [240](#), [514](#)  
pmix\_hdlr\_reg\_cbfunc\_t, [50](#), [131](#), [433](#), [435](#), [530](#)  
pmix\_info\_cbfunc\_t, [48](#), [50](#), [50](#), [79](#), [187](#), [195](#), [197](#), [202](#), [204](#), [224](#), [233](#), [237](#), [266](#), [322](#), [363](#), [371](#),  
         [373](#), [376](#), [390](#), [393](#)  
PMIx\_Info\_directives\_string, [52](#), [483](#), [508](#)  
    PMIxClient.info\_directives\_string (Python), [482](#)  
PMIx\_Init, [7](#), [55](#), [56](#), [58](#), [75](#), [80](#), [97](#), [331](#), [350](#), [404](#), [416](#), [424](#), [462](#), [508](#), [533](#)  
    PMIxClient.init (Python), [462](#)  
PMIx\_Initialized, [55](#), [462](#), [507](#)  
    PMIxClient.initialized (Python), [462](#)  
pmix\_iof\_cbfunc\_t, [387](#), [433](#), [447](#)  
PMIx\_IOF\_channel\_string, [53](#), [484](#), [510](#)  
    PMIxClient.iof\_channel\_string (Python), [484](#)  
PMIx\_IOF\_deregister, [434](#), [499](#), [510](#), [514](#)  
    PMIxTool.iof\_deregister (Python), [499](#)  
PMIx\_IOF\_pull, [350](#), [367](#), [401](#), [402](#), [406](#), [408](#), [410](#), [432](#), [435](#), [499](#), [510](#), [514](#)  
    PMIxTool.iof\_pull (Python), [498](#)  
PMIx\_IOF\_push, [350](#), [367](#), [401](#), [406](#), [408](#), [410](#), [412](#), [414](#), [435](#), [437](#), [500](#), [510](#), [514](#), [523](#)  
    PMIxTool.iof\_push (Python), [500](#)  
PMIx\_Job\_control, [183](#), [192](#), [193](#), [196–198](#), [375](#), [418](#), [510](#), [513](#)  
PMIx\_Job\_control\_nb, [73](#), [191](#), [194](#), [292](#), [472](#), [508](#)  
    PMIxClient.job\_ctrl (Python), [472](#)  
PMIx\_Job\_state\_string, [53](#), [484](#), [514](#)  
    PMIxClient.job\_state\_string (Python), [484](#)  
PMIx\_Link\_state\_string, [54](#), [485](#), [514](#)  
    PMIxClient.link\_state\_string (Python), [485](#)  
PMIx\_Load\_topology, [179](#), [480](#), [514](#), [532](#)  
    PMIxClient.load\_topology (Python), [479](#)  
PMIx\_Log, [167](#), [205](#), [208](#), [211](#), [404](#), [421](#), [471](#), [510](#), [529](#)  
    PMIxClient.log (Python), [471](#)  
PMIx\_Log\_nb, [208](#), [211](#), [508](#)  
PMIx\_Lookup, [111](#), [116](#), [118–120](#), [466](#), [467](#), [507](#)  
    PMIxClient.lookup (Python), [467](#)  
pmix\_lookup\_cbfunc\_t, [123](#), [123](#), [343](#)  
PMIx\_Lookup\_nb, [118](#), [123](#), [507](#)  
pmix\_modex\_cbfunc\_t, [48](#), [335](#), [338](#), [338](#), [339](#)  
pmix\_notification\_fn\_t, [131](#), [135](#), [135](#), [448](#)  
PMIx\_Notify\_event, [138](#), [361](#), [409](#), [478](#), [508](#), [514](#)  
    PMIxClient.notify\_event (Python), [478](#)

pmix\_op\_cbfunc\_t, [49](#), [49](#), [113](#), [126](#), [138](#), [139](#), [142](#), [174](#), [177](#), [208](#), [228](#), [240](#), [263–265](#), [284](#),  
[302–304](#), [306](#), [307](#), [314](#), [320](#), [321](#), [323](#), [330–332](#), [334](#), [340](#), [345](#), [352](#), [355](#), [357](#), [359](#), [360](#),  
[369](#), [374](#), [377](#), [385](#), [388](#), [436](#)

PMIx\_Persistence\_string, [52](#), [482](#), [508](#)  
    PMIxClient.persistence\_string (Python), [482](#)

PMIx\_Proc\_state\_string, [51](#), [481](#), [508](#)  
    PMIxClient.proc\_state\_string (Python), [481](#)

PMIx\_Process\_monitor, [183](#), [199](#), [204](#), [510](#), [513](#)

PMIx\_Process\_monitor\_nb, [202](#), [204](#), [473](#), [508](#)  
    PMIxClient.monitor (Python), [472](#)

PMIx\_Publish, [111](#), [113–116](#), [342](#), [467](#), [507](#)  
    PMIxClient.publish (Python), [466](#)

PMIx\_Publish\_nb, [113](#), [116](#), [507](#)

PMIx\_Put, [26](#), [62–66](#), [88](#), [91](#), [106](#), [106–109](#), [171](#), [223](#), [232](#), [309](#), [310](#), [336](#), [340](#), [465](#), [507](#)  
    PMIxClient.put (Python), [464](#)

PMIx\_Query\_info, [8](#), [74](#), [78](#), [82](#), [83](#), [86](#), [88](#), [213](#), [215](#), [257](#), [394](#), [398](#), [416](#), [417](#), [518](#)

PMIx\_Query\_info\_nb, [72](#), [73](#), [78](#), [78](#), [89](#), [170](#), [293](#), [315](#), [471](#), [508](#), [509](#)  
    PMIxClient.query (Python), [470](#)

PMIx\_Register\_attributes, [314](#), [492](#), [513](#), [514](#)  
    PMIxServer.register\_attributes (Python), [492](#)

PMIx\_Register\_event\_handler, [73](#), [130](#), [166](#), [403](#), [421](#), [477](#), [508](#), [514](#), [529](#)  
    PMIxClient.register\_event\_handler (Python), [477](#)

pmix\_release\_cbfunc\_t, [48](#), [48](#)

PMIx\_Resolve\_nodes, [73](#), [470](#), [507](#)  
    PMIxClient.resolve\_nodes (Python), [470](#)

PMIx\_Resolve\_peers, [73](#), [99](#), [289](#), [470](#), [507](#)  
    PMIxClient.resolve\_peers (Python), [469](#)

PMIx\_Scope\_string, [52](#), [481](#), [508](#)  
    PMIxClient.scope\_string (Python), [481](#)

pmix\_server\_abort\_fn\_t, [333](#), [450](#)

pmix\_server\_alloc\_fn\_t, [370](#), [457](#)

pmix\_server\_client\_connected2\_fn\_t, [330](#), [330–332](#), [449](#), [514](#), [530](#)

pmix\_server\_client\_connected\_fn\_t, [49](#), [268](#), [306](#), [530](#)

pmix\_server\_client\_connected\_fn\_t, [330](#)

pmix\_server\_client\_finalized\_fn\_t, [332](#), [333](#), [449](#)

PMIx\_server\_collect\_inventory, [322](#), [324](#), [494](#), [510](#)  
    PMIxServer.collect\_inventory (Python), [493](#)

pmix\_server\_connect\_fn\_t, [171](#), [352](#), [354](#), [356](#), [453](#)

PMIx\_server\_define\_process\_set, [213](#), [326](#), [495](#), [514](#)  
    PMIxServer.define\_process\_set (Python), [495](#)

PMIx\_server\_delete\_process\_set, [213](#), [327](#), [496](#), [514](#)  
    PMIxServer.delete\_process\_set (Python), [495](#)

PMIx\_server\_deliver\_inventory, [323](#), [494](#), [510](#)

PMIxServer.deliver\_inventory (Python), 494  
    PMIx\_server\_deregister\_client, 306, 490, 507  
        PMIxServer.deregister\_client (Python), 490  
        pmix\_server\_deregister\_events\_fn\_t, 358, 455  
    PMIx\_server\_deregister\_nspace, 302, 307, 488, 507  
        PMIxServer.deregister\_nspace (Python), 488  
    PMIx\_server\_deregister\_resources, 304, 489, 514  
        PMIxServer.deregister\_resources (Python), 489  
        pmix\_server\_disconnect\_fn\_t, 354, 356, 454  
        pmix\_server\_dmodex\_req\_fn\_t, 100, 109, 110, 338, 338, 451, 510, 512  
    PMIx\_server\_dmodex\_request, 308, 309, 310, 491, 507  
        PMIxServer.dmodex\_request (Python), 491  
    pmix\_server\_fabric\_fn\_t, 253, 260, 392, 462, 514  
    pmix\_server\_fencenb\_fn\_t, 334, 337, 338, 450, 512  
    PMIx\_server\_finalize, 279, 486, 507  
        PMIxServer.finalize (Python), 486  
    PMIx\_server\_generate\_cpuset\_string, 182, 324, 514  
    PMIx\_server\_generate\_locality\_string, 179, 180, 324, 495, 514  
        PMIxServer.generate\_locality\_string (Python), 494  
    pmix\_server\_get\_cred\_fn\_t, 379, 383, 459  
    pmix\_server\_grp\_fn\_t, 389, 461, 514  
    PMIx\_server\_init, 55, 257, 276, 280, 315, 328, 395, 396, 400, 486, 507, 514  
        PMIxServer.init (Python), 486  
    PMIx\_server\_IOF\_deliver, 321, 410, 493, 510  
        PMIxServer.iof\_deliver (Python), 493  
    pmix\_server\_iof\_fn\_t, 384, 460  
    pmix\_server\_job\_control\_fn\_t, 373, 458  
    pmix\_server\_listener\_fn\_t, 361  
    pmix\_server\_log\_fn\_t, 368, 457  
    pmix\_server\_lookup\_fn\_t, 342, 452  
    pmix\_server\_module\_t, 277, 279, 315, 316, 328, 328, 329  
    pmix\_server\_monitor\_fn\_t, 376, 458  
    pmix\_server\_notify\_event\_fn\_t, 137, 141, 360, 361, 455  
    pmix\_server\_publish\_fn\_t, 340, 451  
    pmix\_server\_query\_fn\_t, 363, 456  
    PMIx\_server\_register\_client, 268, 305, 306, 331, 333, 490, 507  
        PMIxServer.register\_client (Python), 489  
        pmix\_server\_register\_events\_fn\_t, 356, 454  
    PMIx\_server\_register\_nspace, 10, 49, 282, 283, 284, 285, 292–294, 297, 303, 304, 324, 325, 488,  
        507, 509, 520  
        PMIxServer.register\_nspace (Python), 488  
    PMIx\_server\_register\_resources, 286, 289, 290, 303, 489, 514  
        PMIxServer.register\_resources (Python), 488

PMIx\_server\_setup\_application, [310](#), [313](#), [314](#), [320](#), [324](#), [492](#), [508](#), [512](#)  
    PMIxServer.setup\_application (Python), [491](#)

PMIx\_server\_setup\_fork, [307](#), [491](#), [507](#)  
    PMIxServer.setup\_fork (Python), [490](#)

PMIx\_server\_setup\_local\_support, [319](#), [493](#), [508](#)  
    PMIxServer.setup\_local\_support (Python), [492](#)

pmix\_server\_spawn\_fn\_t, [170](#), [346](#), [405](#), [453](#)

pmix\_server\_stdin\_fn\_t, [388](#), [460](#)

pmix\_server\_tool\_connection\_fn\_t, [268](#), [365](#), [395](#), [456](#)

pmix\_server\_unpublish\_fn\_t, [344](#), [452](#)

pmix\_server\_validate\_cred\_fn\_t, [381](#), [459](#)

pmix\_setup\_application\_cbfunc\_t, [311](#), [313](#)

PMIx\_Spawn, [94](#), [97](#), [98](#), [153](#), [153](#), [158](#), [163](#), [164](#), [167](#), [186](#), [189](#), [291](#), [292](#), [308](#), [346](#), [348](#), [350](#),  
    [352](#), [372](#), [401](#), [404](#), [405](#), [408](#), [415](#), [416](#), [418](#)–[420](#), [423](#), [424](#), [453](#), [468](#), [507](#), [513](#), [521](#)

    PMIxClient.spawn (Python), [468](#)

pmix\_spawn\_cbfunc\_t, [159](#), [170](#), [170](#), [347](#), [408](#)

PMIx\_Spawn\_nb, [158](#), [167](#), [170](#), [507](#)

PMIx\_Store\_internal, [106](#), [107](#), [107](#), [464](#), [507](#)  
    PMIxClient.store\_internal (Python), [464](#)

PMIx\_tool\_attach\_to\_server, [429](#), [498](#), [515](#), [530](#)  
    PMIxTool.attach\_to\_server (Python), [497](#)

PMIx\_tool\_connect\_to\_server, [397](#), [400](#), [428](#), [510](#), [531](#)

PMIx\_tool\_connect\_to\_server (**Deprecated**), [530](#)

pmix\_tool\_connection\_cbfunc\_t, [366](#), [367](#), [368](#)

PMIx\_tool\_disconnect, [429](#), [497](#), [515](#)  
    PMIxTool.disconnect (Python), [497](#)

PMIx\_tool\_finalize, [428](#), [497](#), [508](#)  
    PMIxTool.finalize (Python), [496](#)

PMIx\_tool\_get\_servers, [431](#), [498](#), [515](#)  
    PMIxTool.get\_servers (Python), [498](#)

PMIx\_tool\_init, [55](#), [394](#), [397](#), [399](#)–[401](#), [410](#), [425](#), [428](#), [496](#), [508](#)  
    PMIxTool.init (Python), [496](#)

PMIx\_tool\_set\_server, [396](#), [431](#), [432](#), [515](#)

PMIx\_Unpublish, [124](#), [126](#), [127](#), [468](#), [507](#)  
    PMIxClient.unpublish (Python), [467](#)

PMIx\_Unpublish\_nb, [126](#), [507](#)

PMIx\_Validate\_credential, [272](#), [474](#), [510](#), [513](#)  
    PMIxClient.validate\_credential (Python), [473](#)

PMIx\_Validate\_credential\_nb, [273](#)

pmix\_validation\_cbfunc\_t, [274](#), [382](#), [383](#)

pmix\_value\_cbfunc\_t, [49](#), [49](#)

# Index of Support Macros

---

PMIX\_APP\_CONSTRUCT, [168](#)  
PMIX\_APP\_CREATE, [169](#)  
PMIX\_APP\_DESTRUCT, [168](#)  
PMIX\_APP\_FREE, [169](#)  
PMIX\_APP\_INFO\_CREATE, [169](#), [510](#), [512](#)  
PMIX\_APP\_RELEASE, [169](#)  
PMIX\_ARGV\_APPEND, [42](#)  
PMIX\_ARGV\_APPEND\_UNIQUE, [43](#)  
PMIX\_ARGV\_COPY, [45](#)  
PMIX\_ARGV\_COUNT, [45](#)  
PMIX\_ARGV\_FREE, [44](#)  
PMIX\_ARGV\_JOIN, [45](#)  
PMIX\_ARGV\_PREPEND, [43](#)  
PMIX\_ARGV\_SPLIT, [44](#)  
PMIX\_BYTE\_OBJECT\_CONSTRUCT, [39](#)  
PMIX\_BYTE\_OBJECT\_CREATE, [39](#)  
PMIX\_BYTE\_OBJECT\_DESTRUCT, [39](#)  
PMIX\_BYTE\_OBJECT\_FREE, [40](#)  
PMIX\_BYTE\_OBJECT\_LOAD, [40](#)  
PMIX\_CHECK\_KEY, [16](#)  
PMIX\_CHECK\_NSPACE, [18](#)  
PMIX\_CHECK\_PROCID, [21](#)  
PMIX\_COORD\_CONSTRUCT, [249](#)  
PMIX\_COORD\_CREATE, [250](#)  
PMIX\_COORD\_DESTRUCT, [250](#)  
PMIX\_COORD\_FREE, [250](#)  
PMIX\_CPUSET\_CONSTRUCT, [326](#)  
PMIX\_CPUSET\_CREATE, [326](#)  
PMIX\_DATA\_ARRAY\_CONSTRUCT, [41](#)  
PMIX\_DATA\_ARRAY\_CREATE, [41](#)  
PMIX\_DATA\_ARRAY\_DESTRUCT, [41](#)  
PMIX\_DATA\_ARRAY\_FREE, [42](#)  
PMIX\_DATA\_BUFFER\_CONSTRUCT, [144](#), [146](#), [148](#)  
PMIX\_DATA\_BUFFER\_CREATE, [144](#), [146](#), [148](#)  
PMIX\_DATA\_BUFFER\_DESTRUCT, [144](#)  
PMIX\_DATA\_BUFFER\_LOAD, [145](#)  
PMIX\_DATA\_BUFFER\_RELEASE, [144](#)  
PMIX\_DATA\_BUFFER\_UNLOAD, [145](#), [282](#), [283](#)

PMIX\_DEVICE\_DIST\_CONSTRUCT, [248](#)  
PMIX\_DEVICE\_DIST\_CREATE, [248](#)  
PMIX\_DEVICE\_DIST\_DESTRUCT, [248](#)  
PMIX\_DEVICE\_DIST\_FREE, [248](#)  
PMIX\_ENDPOINT\_CONSTRUCT, [246](#)  
PMIX\_ENDPOINT\_CREATE, [246](#)  
PMIX\_ENDPOINT\_DESTRUCT, [246](#)  
PMIX\_ENDPOINT\_FREE, [247](#)  
PMIX\_ENVAR\_CONSTRUCT, [37](#)  
PMIX\_ENVAR\_CREATE, [38](#)  
PMIX\_ENVAR\_DESTROY, [13](#), [37](#)  
PMIX\_ENVAR\_FREE, [38](#)  
PMIX\_ENVAR\_LOAD, [38](#)  
PMIX\_FABRIC\_CONSTRUCT, [256](#)  
PMIX\_GEOMETRY\_CONSTRUCT, [251](#)  
PMIX\_GEOMETRY\_CREATE, [252](#)  
PMIX\_GEOMETRY\_DESTRUCT, [251](#)  
PMIX\_GEOMETRY\_FREE, [252](#)  
PMIx\_Heartbeat, [204](#), [508](#)  
PMIX\_INFO\_CONSTRUCT, [31](#)  
PMIX\_INFO\_CREATE, [32](#), [35](#), [36](#)  
PMIX\_INFO\_DESTROY, [32](#)  
PMIX\_INFO\_FREE, [32](#)  
PMIX\_INFO\_IS\_END, [36](#), [510](#), [512](#)  
PMIX\_INFO\_IS\_OPTIONAL, [36](#)  
PMIX\_INFO\_IS\_REQUIRED, [34](#), [35](#), [36](#)  
PMIX\_INFO\_LOAD, [32](#)  
PMIX\_INFO\_OPTIONAL, [35](#)  
PMIX\_INFO\_REQUIRED, [34](#), [35](#)  
PMIX\_INFO\_TRUE, [34](#)  
PMIX\_INFO\_XFER, [33](#), [294](#)  
PMIX\_LOAD\_KEY, [17](#)  
PMIX\_LOAD\_NSPACE, [18](#)  
PMIX\_LOAD\_PROCID, [21](#), [21](#)  
PMIX\_MULTICLUSTER\_NSPACE\_CONSTRUCT, [22](#)  
PMIX\_MULTICLUSTER\_NSPACE\_PARSE, [22](#)  
PMIX\_PDATA\_CONSTRUCT, [120](#)  
PMIX\_PDATA\_CREATE, [121](#)  
PMIX\_PDATA\_DESTRUCT, [120](#)  
PMIX\_PDATA\_FREE, [121](#)  
PMIX\_PDATA\_LOAD, [121](#)  
PMIX\_PDATA\_RELEASE, [121](#)  
PMIX\_PDATA\_XFER, [122](#)

PMIX\_PROC\_CONSTRUCT, [19](#)  
PMIX\_PROC\_CREATE, [20](#)  
PMIX\_PROC\_DESTRUCT, [19](#)  
PMIX\_PROC\_FREE, [20](#), [73](#)  
PMIX\_PROC\_INFO\_CONSTRUCT, [24](#)  
PMIX\_PROC\_INFO\_CREATE, [25](#)  
PMIX\_PROC\_INFO\_DESTRUCT, [24](#)  
PMIX\_PROC\_INFO\_FREE, [25](#)  
PMIX\_PROC\_INFO\_RELEASE, [25](#)  
PMIX\_PROC\_LOAD, [20](#)  
PMIX\_PROC\_RELEASE, [20](#)  
PMIX\_QUERY\_CONSTRUCT, [86](#)  
PMIX\_QUERY\_CREATE, [87](#)  
PMIX\_QUERY\_DESTRUCT, [86](#)  
PMIX\_QUERY\_FREE, [87](#)  
PMIX\_QUERY\_QUALIFIERS\_CREATE, [87](#), [510](#), [512](#)  
PMIX\_QUERY\_RELEASE, [87](#)  
PMIX\_REGATTR\_CONSTRUCT, [317](#)  
PMIX\_REGATTR\_CREATE, [318](#)  
PMIX\_REGATTR\_DESTRUCT, [318](#)  
PMIX\_REGATTR\_FREE, [318](#)  
PMIX\_REGATTR\_LOAD, [319](#)  
PMIX\_REGATTR\_XFER, [319](#)  
PMIX\_SETENV, [46](#)  
PMIX\_SYSTEM\_EVENT, [134](#)  
PMIX\_TOPOLOGY\_CONSTRUCT, [181](#)  
PMIX\_VALUE\_CONSTRUCT, [28](#)  
PMIX\_VALUE\_CREATE, [28](#)  
PMIX\_VALUE\_DESTRUCT, [28](#)  
PMIX\_VALUE\_FREE, [29](#)  
PMIX\_VALUE\_GET\_NUMBER, [31](#)  
PMIX\_VALUE\_LOAD, [29](#)  
PMIX\_VALUE\_RELEASE, [28](#)  
PMIX\_VALUE\_UNLOAD, [30](#)  
PMIX\_VALUE\_XFER, [30](#)

# Index of Data Structures

---

pmix\_alloc\_directive\_t, 48, 53, 184, 187, 191, 191, 371, 443, 483  
pmix\_app\_t, 42, 43, 46, 154–156, 159, 161, 165, 167, 167–169, 347, 348, 350, 351, 401, 404, 405, 415, 416, 419, 423, 424, 444, 510, 512, 528  
pmix\_byte\_object\_t, 39, 39, 40, 47, 269, 270, 272, 274, 321, 381, 382, 388, 436, 442  
pmix\_coord\_t, 48, 249, 249–251, 445, 515  
pmix\_coord\_view\_t, 252, 445, 515  
pmix\_cpuset\_t, 48, 324, 325, 325, 326, 444, 515  
pmix\_data\_array\_t, 26, 40, 40–42, 47, 77, 78, 81, 82, 84, 89, 99, 185, 188, 190, 215, 218, 255, 257, 258, 260, 261, 285–290, 296, 297, 299, 311, 364, 373, 393, 417, 425, 443, 510, 512, 524, 525, 527  
pmix\_data\_buffer\_t, 143, 143–147, 151  
pmix\_data\_range\_t, 47, 52, 115, 115, 139, 360, 442, 482  
pmix\_data\_type\_t, 29, 31, 33, 41, 46, 46, 47, 52, 122, 146, 148–150, 319, 441, 483  
pmix\_device\_distance\_t, 48, 247, 247, 248, 260, 266, 444, 515, 524  
pmix\_endpoint\_t, 48, 245, 245–247, 260, 444, 524  
pmix\_envar\_t, 13, 37, 37, 38, 48, 443  
pmix\_fabric\_operation\_t, 253, 253, 392  
pmix\_fabric\_t, 245, 253, 253, 256, 258, 261–265, 393, 444, 515, 524  
pmix\_geometry\_t, 48, 244, 250, 250–252, 258, 259, 445, 515, 524, 526  
pmix\_group\_operation\_t, 390, 392, 392, 515  
pmix\_group\_opt\_t, 235, 237, 238, 238, 475, 515  
pmix\_info\_directives\_t, 34, 34, 35, 47, 52, 443, 482  
pmix\_info\_t, 4, 5, 8, 16, 17, 31, 31–36, 47, 50, 56, 58, 60, 74, 78, 83, 85–87, 89, 112, 114–117, 136, 139, 142, 169, 184, 185, 187, 188, 190–192, 194, 197, 199, 200, 204, 207, 210, 212, 220, 222, 224, 226, 228, 230, 233, 235, 237, 239, 240, 254, 255, 258, 260, 269, 270, 272, 274, 277, 279, 284–286, 288, 292–294, 296–299, 311, 317, 319, 321–323, 330, 350, 360, 366, 367, 370, 372, 373, 376, 377, 384, 385, 387, 393, 401, 404, 405, 415, 416, 423–425, 430, 433, 435, 436, 443, 446, 508, 510, 512, 518–521, 525  
pmix\_iof\_channel\_t, 48, 53, 321, 385, 387, 413, 413, 433, 443, 484  
pmix\_job\_state\_t, 26, 26, 48, 53, 444, 484, 515  
pmix\_key\_t, 8, 16, 16, 67, 106, 319, 441  
pmix\_link\_state\_t, 48, 54, 245, 253, 253, 256, 259, 444, 485, 515, 526  
pmix\_locality\_t, 180, 181, 181, 444, 515, 532  
pmix\_nspace\_t, 17, 17, 18, 21, 22, 170, 441, 442  
pmix\_pdata\_t, 116, 117, 120, 120–123, 443  
pmix\_persistence\_t, 47, 52, 115, 115, 442, 482  
pmix\_proc\_info\_t, 23, 23–25, 47, 75, 77, 80–82, 84, 364, 416, 417, 425, 442  
pmix\_proc\_state\_t, 22, 22, 47, 51, 442, 481

pmix\_proc\_t, 18, [19](#), 19–21, 47, 58, 62, 64, 65, 69, 85, 99, 122, 132, 133, 135, 136, 139, 140, 146, 147, 152, 215, 218–220, 224, 230, 233, 236, 289, 305, 307, 308, 319, 321, 326, 330–332, 334, 335, 339, 340, 343, 345, 347, 352, 355, 360, 363, 368, 369, 371, 374, 377, 379, 382, 385, 387, 388, 390–392, 425, 428–432, 442, 527  
pmix\_query\_t, 48, 76, 81, 83, [86](#), 86–88, 363, 365, 444, 510, 512, 518  
pmix\_rank\_t, [18](#), 18, 19, 21, 47, 442  
pmix\_regattr\_t, 48, 89, 315, [316](#), 316–319, 444, 513, 515, 520  
pmix\_scope\_t, 47, 52, [107](#), 107, 442, 481  
pmix\_status\_t, [14](#), 14, 30, 31, 42, 43, 46, 47, 50, 51, 131, 134, 136, 139, 142, 310, 314, 357, 359, 360, 368, 381, 384, 441, 455, 480  
pmix\_topology\_t, 179, [180](#), 180, 181, 445, 515  
pmix\_value\_t, 8, [26](#), 26–31, 47, 49, 50, 69, 106, 443

# Index of Constants

---

PMIX\_ALLOC\_DIRECTIVE, [48](#)  
PMIX\_ALLOC\_EXTEND, [191](#)  
PMIX\_ALLOC\_EXTERNAL, [191](#)  
PMIX\_ALLOC\_NEW, [191](#)  
PMIX\_ALLOC\_REAQUIRE, [191](#)  
PMIX\_ALLOC\_RELEASE, [191](#)  
PMIX\_APP, [47](#)  
PMIX\_APP\_WILDCARD, [13](#)  
PMIX\_BOOL, [47](#)  
PMIX\_BUFFER, [47](#)  
PMIX\_BYTE, [47](#)  
PMIX\_BYTE\_OBJECT, [47](#)  
PMIX\_COMMAND, [47](#)  
PMIX\_COMPRESSED\_STRING, [48](#)  
PMIX\_CONNECT\_REQUESTED, [530](#)  
PMIX\_COORD, [48](#)  
PMIX\_COORD\_LOGICAL\_VIEW, [252](#)  
PMIX\_COORD\_PHYSICAL\_VIEW, [252](#)  
PMIX\_COORD\_VIEW\_UNDEF, [252](#)  
PMIX\_DATA\_ARRAY, [47](#)  
PMIX\_DATA\_RANGE, [47](#)  
PMIX\_DATA\_TYPE, [47](#)  
PMIX\_DATA\_TYPE\_MAX, [48](#)  
PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY, [423](#)  
PMIX\_DEBUGGER\_RELEASE, [423](#)  
PMIX\_DEVICE\_DIST, [48](#)  
PMIX\_DOUBLE, [47](#)  
PMIX\_ENDPOINT, [48](#)  
PMIX\_ENVAR, [48](#)  
PMIX\_ERR\_BAD\_PARAM, [15](#)  
PMIX\_ERR\_COMM\_FAILURE, [15](#)  
PMIX\_ERR\_CONFLICTING\_CLEANUP\_DIRECTIVES, [197](#)  
PMIX\_ERR\_DATA\_VALUE\_NOT\_FOUND, [531](#)  
PMIX\_ERR\_DEBUGGER\_RELEASE, [530](#)  
PMIX\_ERR\_DUPLICATE\_KEY, [114](#)  
PMIX\_ERR\_EVENT\_REGISTRATION, [133](#)  
PMIX\_ERR\_EXISTS, [14](#)  
PMIX\_ERR\_GET\_MALLOC\_REQD, [71](#)

PMIX\_ERR\_HANDSHAKE\_FAILED, [530](#)  
PMIX\_ERR\_IN\_ERRNO, [530](#)  
PMIX\_ERR\_INIT, [15](#)  
PMIX\_ERR\_INVALID\_ARG, [531](#)  
PMIX\_ERR\_INVALID\_ARGS, [531](#)  
PMIX\_ERR\_INVALID\_CRED, [14](#)  
PMIX\_ERR\_INVALID\_KEY, [531](#)  
PMIX\_ERR\_INVALID\_KEY\_LENGTH, [531](#)  
PMIX\_ERR\_INVALID\_KEYVALP, [531](#)  
PMIX\_ERR\_INVALID\_LENGTH, [531](#)  
PMIX\_ERR\_INVALID\_NAMESPACE, [531](#)  
PMIX\_ERR\_INVALID\_NUM\_ARGS, [531](#)  
PMIX\_ERR\_INVALID\_NUM\_PARSED, [531](#)  
PMIX\_ERR\_INVALID\_OPERATION, [15](#)  
PMIX\_ERR\_INVALID\_SIZE, [531](#)  
PMIX\_ERR\_INVALID\_TERMINATION, [530](#)  
PMIX\_ERR\_INVALID\_VAL, [531](#)  
PMIX\_ERR\_INVALID\_VAL\_LENGTH, [530](#)  
PMIX\_ERR\_IOF\_COMPLETE, [413](#)  
PMIX\_ERR\_IOF\_FAILURE, [413](#)  
PMIX\_ERR\_JOB\_ABORTED, [422](#)  
PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG, [422](#)  
PMIX\_ERR\_JOB\_ABORTED\_BY\_SYS\_EVENT, [423](#)  
PMIX\_ERR\_JOB\_ALLOC\_FAILED, [163](#)  
PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE, [163](#)  
PMIX\_ERR\_JOB\_CANCELED, [422](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_LAUNCH, [163](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_MAP, [163](#)  
PMIX\_ERR\_JOB\_KILLED\_BY\_CMD, [422](#)  
PMIX\_ERR\_JOB\_NO\_EXE\_SPECIFIED, [163](#)  
PMIX\_ERR\_JOB\_NON\_ZERO\_TERM, [423](#)  
PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED, [422](#)  
PMIX\_ERR\_JOB\_TERM\_WO\_SYNC, [422](#)  
PMIX\_ERR\_JOB\_TERMINATED, [530](#)  
PMIX\_ERR\_LOST\_CONNECTION, [15](#)  
PMIX\_ERR\_LOST\_CONNECTION\_TO\_CLIENT, [530](#)  
PMIX\_ERR\_LOST\_CONNECTION\_TO\_SERVER, [530](#)  
PMIX\_ERR\_LOST\_PEER\_CONNECTION, [530](#)  
PMIX\_ERR\_NO\_PERMISSIONS, [14](#)  
PMIX\_ERR\_NODE\_DOWN, [530](#)  
PMIX\_ERR\_NODE\_OFFLINE, [530](#)  
PMIX\_ERR\_NOMEM, [15](#)  
PMIX\_ERR\_NOT\_FOUND, [15](#)

PMIX\_ERR\_NOT\_IMPLEMENTED, [531](#)  
PMIX\_ERR\_NOT\_SUPPORTED, [15](#)  
PMIX\_ERR\_OUT\_OF\_RESOURCE, [15](#)  
PMIX\_ERR\_PACK\_FAILURE, [14](#)  
PMIX\_ERR\_PACK\_MISMATCH, [531](#)  
PMIX\_ERR\_PARTIAL\_SUCCESS, [15](#)  
PMIX\_ERR\_PROC\_ABORTED, [530](#)  
PMIX\_ERR\_PROC\_ABORTING, [530](#)  
PMIX\_ERR\_PROC\_CHECKPOINT, [198](#)  
PMIX\_ERR\_PROC\_ENTRY\_NOT\_FOUND, [531](#)  
PMIX\_ERR\_PROC\_MIGRATE, [198](#)  
PMIX\_ERR\_PROC\_REQUESTED\_ABORT, [531](#)  
PMIX\_ERR\_PROC\_RESTART, [198](#)  
PMIX\_ERR\_PROC\_TERM\_WO\_SYNC, [422](#)  
PMIX\_ERR\_READY\_FOR\_HANDSHAKE, [530](#)  
PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION, [316](#)  
PMIX\_ERR\_RESOURCE\_BUSY, [15](#)  
PMIX\_ERR\_SERVER\_FAILED\_REQUEST, [531](#)  
PMIX\_ERR\_SERVER\_NOT\_AVAIL, [531](#)  
PMIX\_ERR\_SILENT, [531](#)  
PMIX\_ERR\_SYS\_OTHER, [530](#)  
PMIX\_ERR\_TIMEOUT, [14](#)  
PMIX\_ERR\_TYPE\_MISMATCH, [14](#)  
PMIX\_ERR\_UNKNOWN\_DATA\_TYPE, [14](#)  
PMIX\_ERR\_UNPACK\_FAILURE, [14](#)  
PMIX\_ERR\_UNPACK\_INADEQUATE\_SPACE, [14](#)  
PMIX\_ERR\_UNPACK\_READ\_PAST\_END\_OF\_BUFFER, [14](#)  
PMIX\_ERR\_UNREACH, [14](#)  
PMIX\_ERR\_WOULD\_BLOCK, [14](#)  
PMIX\_ERROR, [14](#)  
PMIX\_EVENT\_ACTION\_COMPLETE, [142](#)  
PMIX\_EVENT\_ACTION\_DEFERRED, [142](#)  
PMIX\_EVENT\_JOB\_END, [422](#)  
PMIX\_EVENT\_JOB\_START, [422](#)  
PMIX\_EVENT\_NO\_ACTION\_TAKEN, [142](#)  
PMIX\_EVENT\_NODE\_DOWN, [134](#)  
PMIX\_EVENT\_NODE\_OFFLINE, [134](#)  
PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN, [142](#)  
PMIX\_EVENT\_PROC\_TERMINATED, [422](#)  
PMIX\_EVENT\_SESSION\_END, [422](#)  
PMIX\_EVENT\_SESSION\_START, [422](#)  
PMIX\_EVENT\_SYS\_BASE, [134](#)  
PMIX\_EVENT\_SYS\_OTHER, [134](#)

PMIX\_EXISTS, [530](#)  
PMIX\_EXTERNAL\_ERR\_BASE, [15](#)  
PMIX\_FABRIC\_REQUEST\_INFO, [253](#)  
PMIX\_FABRIC\_UPDATE\_ENDPOINTS, [245](#)  
PMIX\_FABRIC\_UPDATE\_INFO, [253](#)  
PMIX\_FABRIC\_UPDATE\_PENDING, [245](#)  
PMIX\_FABRIC\_UPDATED, [245](#)  
PMIX\_FLOAT, [47](#)  
PMIX\_FWD\_ALL\_CHANNELS, [413](#)  
PMIX\_FWD\_NO\_CHANNELS, [413](#)  
PMIX\_FWD\_STDDIAG\_CHANNEL, [413](#)  
PMIX\_FWD\_STDERR\_CHANNEL, [413](#)  
PMIX\_FWD\_STDIN\_CHANNEL, [413](#)  
PMIX\_FWD\_STDOUT\_CHANNEL, [413](#)  
PMIX\_GDS\_ACTION\_COMPLETE, [531](#)  
PMIX\_GEOMETRY, [48](#)  
PMIX\_GLOBAL, [107](#)  
PMIX\_GROUP\_ACCEPT, [238](#)  
PMIX\_GROUP\_CONSTRUCT, [392](#)  
PMIX\_GROUP\_CONSTRUCT\_ABORT, [218](#)  
PMIX\_GROUP\_CONSTRUCT\_COMPLETE, [218](#)  
PMIX\_GROUP\_CONTEXT\_ID\_ASSIGNED, [218](#)  
PMIX\_GROUP\_DECLINE, [238](#)  
PMIX\_GROUP\_DESTRUCT, [392](#)  
PMIX\_GROUP\_INVITE\_ACCEPTED, [218](#)  
PMIX\_GROUP\_INVITE\_DECLINED, [218](#)  
PMIX\_GROUP\_INVITE\_FAILED, [218](#)  
PMIX\_GROUP\_INVITED, [217](#)  
PMIX\_GROUP\_LEADER FAILED, [218](#)  
PMIX\_GROUP\_LEADER\_SELECTED, [218](#)  
PMIX\_GROUP\_LEFT, [217](#)  
PMIX\_GROUP\_MEMBER\_FAILED, [217](#)  
PMIX\_GROUP\_MEMBERSHIP\_UPDATE, [218](#)  
PMIX\_INFO, [47](#)  
PMIX\_INFO\_ARRAY, [508](#), [511](#)  
PMIX\_INFO\_ARRAY\_END, [35](#)  
PMIX\_INFO\_DIR\_RESERVED, [35](#)  
PMIX\_INFO\_DIRECTIVES, [47](#)  
PMIX\_INFO\_REQD, [35](#)  
PMIX\_INT, [47](#)  
PMIX\_INT16, [47](#)  
PMIX\_INT32, [47](#)  
PMIX\_INT64, [47](#)

PMIX\_INT8, [47](#)  
PMIX\_INTERNAL, [107](#)  
PMIX\_JOF\_CHANNEL, [48](#)  
PMIX\_JCTRL\_CHECKPOINT, [198](#)  
PMIX\_JCTRL\_CHECKPOINT\_COMPLETE, [198](#)  
PMIX\_JCTRL\_PREEMPT\_ALERT, [198](#)  
PMIX\_JOB\_STATE, [48](#)  
PMIX\_JOB\_STATE\_AWAITING\_ALLOC, [26](#)  
PMIX\_JOB\_STATE\_CONNECTED, [26](#)  
PMIX\_JOB\_STATE\_LAUNCH\_UNDERWAY, [26](#)  
PMIX\_JOB\_STATE\_RUNNING, [26](#)  
PMIX\_JOB\_STATE\_SUSPENDED, [26](#)  
PMIX\_JOB\_STATE\_TERMINATED, [26](#)  
PMIX\_JOB\_STATE\_TERMINATED\_WITH\_ERROR, [26](#)  
PMIX\_JOB\_STATE\_UNDEF, [26](#)  
PMIX\_JOB\_STATE\_UNTERMINATED, [26](#)  
PMIX\_KVAL, [47](#)  
PMIX\_LAUNCH\_COMPLETE, [422](#)  
PMIX\_LAUNCH\_DIRECTIVE, [409](#)  
PMIX\_LAUNCHER\_READY, [409](#)  
PMIX\_LINK\_DOWN, [253](#)  
PMIX\_LINK\_STATE, [48](#)  
PMIX\_LINK\_STATE\_UNKNOWN, [253](#)  
PMIX\_LINK\_UP, [253](#)  
PMIX\_LOCAL, [107](#)  
PMIX\_LOCALITY\_NONLOCAL, [181](#)  
PMIX\_LOCALITY\_SHARE\_CORE, [181](#)  
PMIX\_LOCALITY\_SHARE\_HWTHREAD, [181](#)  
PMIX\_LOCALITY\_SHARE\_L1CACHE, [181](#)  
PMIX\_LOCALITY\_SHARE\_L2CACHE, [181](#)  
PMIX\_LOCALITY\_SHARE\_L3CACHE, [181](#)  
PMIX\_LOCALITY\_SHARE\_NODE, [181](#)  
PMIX\_LOCALITY\_SHARE\_NUMA, [181](#)  
PMIX\_LOCALITY\_SHARE\_PACKAGE, [181](#)  
PMIX\_LOCALITY\_UNKNOWN, [181](#)  
PMIX\_MAX\_KEYLEN, [13](#)  
PMIX\_MAX\_NSLEN, [13](#)  
PMIX\_MODEL\_DECLARED, [58](#)  
PMIX\_MODEL\_RESOURCES, [58](#)  
PMIX\_MODEX, [508](#), [511](#)  
PMIX\_MONITOR\_FILE\_ALERT, [204](#)  
PMIX\_MONITOR\_HEARTBEAT\_ALERT, [204](#)  
PMIX\_NOTIFY\_ALLOC\_COMPLETE, [531](#)

PMIX\_OPENMP\_PARALLEL\_ENTERED, [58](#)  
PMIX\_OPENMP\_PARALLEL\_EXITED, [58](#)  
PMIX\_OPERATION\_IN\_PROGRESS, [15](#)  
PMIX\_OPERATION\_SUCCEEDED, [15](#)  
PMIX\_PDATA, [47](#)  
PMIX\_PERSIST, [47](#)  
PMIX\_PERSIST\_APP, [115](#)  
PMIX\_PERSIST\_FIRST\_READ, [115](#)  
PMIX\_PERSIST\_INDEF, [115](#)  
PMIX\_PERSIST\_INVALID, [115](#)  
PMIX\_PERSIST\_PROC, [115](#)  
PMIX\_PERSIST\_SESSION, [115](#)  
PMIX\_PID, [47](#)  
PMIX\_POINTER, [47](#)  
PMIX\_PROC, [47](#)  
PMIX\_PROC\_CPUSET, [48](#)  
PMIX\_PROC\_HAS\_CONNECTED, [530](#)  
PMIX\_PROC\_INFO, [47](#)  
PMIX\_PROC\_RANK, [47](#)  
PMIX\_PROC\_STATE, [47](#)  
PMIX\_PROC\_STATE\_ABORTED, [23](#)  
PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG, [23](#)  
PMIX\_PROC\_STATE\_CALLED\_ABORT, [23](#)  
PMIX\_PROC\_STATE\_CANNOT\_RESTART, [23](#)  
PMIX\_PROC\_STATE\_COMM\_FAILED, [23](#)  
PMIX\_PROC\_STATE\_CONNECTED, [23](#)  
PMIX\_PROC\_STATE\_ERROR, [23](#)  
PMIX\_PROC\_STATE\_FAILED\_TO\_LAUNCH, [23](#)  
PMIX\_PROC\_STATE\_FAILED\_TO\_START, [23](#)  
PMIX\_PROC\_STATE\_HEARTBEAT\_FAILED, [23](#)  
PMIX\_PROC\_STATE\_KILLED\_BY\_CMD, [23](#)  
PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY, [23](#)  
PMIX\_PROC\_STATE\_MIGRATING, [23](#)  
PMIX\_PROC\_STATE\_PREPPED, [23](#)  
PMIX\_PROC\_STATE\_RESTART, [23](#)  
PMIX\_PROC\_STATE\_RUNNING, [23](#)  
PMIX\_PROC\_STATE\_SENSOR\_BOUND\_EXCEEDED, [23](#)  
PMIX\_PROC\_STATE\_TERM\_NON\_ZERO, [23](#)  
PMIX\_PROC\_STATE\_TERM\_WO\_SYNC, [23](#)  
PMIX\_PROC\_STATE\_TERMINATE, [23](#)  
PMIX\_PROC\_STATE\_TERMINATED, [23](#)  
PMIX\_PROC\_STATE\_UNDEF, [23](#)  
PMIX\_PROC\_STATE\_UNTERMINATED, [23](#)

PMIX\_PROC\_TERMINATED, [530](#)  
PMIX\_PROCESS\_SET\_DEFINE, [214](#)  
PMIX\_PROCESS\_SET\_DELETE, [214](#)  
PMIX\_QUERY, [48](#)  
PMIX\_QUERY\_PARTIAL\_SUCCESS, [83](#)  
PMIX\_RANGE\_CUSTOM, [115](#)  
PMIX\_RANGE\_GLOBAL, [115](#)  
PMIX\_RANGE\_INVALID, [115](#)  
PMIX\_RANGE\_LOCAL, [115](#)  
PMIX\_RANGE\_NAMESPACE, [115](#)  
PMIX\_RANGE\_PROC\_LOCAL, [115](#)  
PMIX\_RANGE\_RM, [115](#)  
PMIX\_RANGE\_SESSION, [115](#)  
PMIX\_RANGE\_UNDEF, [115](#)  
PMIX\_RANK\_INVALID, [19](#)  
PMIX\_RANK\_LOCAL\_NODE, [19](#)  
PMIX\_RANK\_LOCAL\_PEERS, [19](#)  
PMIX\_RANK\_UNDEF, [18](#)  
PMIX\_RANK\_VALID, [19](#)  
PMIX\_RANK\_WILDCARD, [18](#)  
PMIX\_REGATTR, [48](#)  
PMIX\_REGEX, [48](#)  
PMIX\_REMOTE, [107](#)  
PMIX\_SCOPE, [47](#)  
PMIX\_SCOPE\_UNDEF, [107](#)  
PMIX\_SIZE, [47](#)  
PMIX\_STATUS, [47](#)  
PMIX\_STRING, [47](#)  
PMIX\_SUCCESS, [14](#)  
PMIX\_TIME, [47](#)  
PMIX\_TIMEVAL, [47](#)  
PMIX\_UINT, [47](#)  
PMIX\_UINT16, [47](#)  
PMIX\_UINT32, [47](#)  
PMIX\_UINT64, [47](#)  
PMIX\_UINT8, [47](#)  
PMIX\_UNDEF, [47](#)  
PMIX\_VALUE, [47](#)

# Index of Environmental Variables

---

PMIX\_LAUNCHER\_PAUSE\_FOR\_TOOL, [399](#), [529](#)

PMIX\_LAUNCHER\_RENDEZVOUS\_FILE, [400](#), [529](#)

# Index of Attributes

---

PMIX\_ACCESS\_GRPIDS, [115](#), [521](#)  
PMIX\_ACCESS\_PERMISSIONS, [112](#), [114](#), [114](#), [521](#)  
PMIX\_ACCESS\_USERIDS, [115](#), [521](#)  
PMIX\_ADD\_ENVAR, [157](#), [162](#), [167](#)  
PMIX\_ADD\_HOST, [155](#), [160](#), [164](#), [349](#)  
PMIX\_ADD\_HOSTFILE, [155](#), [160](#), [164](#), [349](#)  
PMIX\_ALL\_CLONES\_PARTICIPATE, [63](#), [66](#), [66](#), [172](#), [175](#), [176](#), [178](#), [518](#)  
PMIX\_ALLOC\_BANDWIDTH, [157](#), [163](#), [185](#), [188](#), [190](#), [190](#), [311](#), [312](#), [373](#)  
PMIX\_ALLOC\_CPU\_LIST, [157](#), [162](#), [185](#), [188](#), [190](#), [372](#)  
PMIX\_ALLOC\_FABRIC, [185](#), [188](#), [190](#), [311](#), [372](#), [531](#)  
PMIX\_ALLOC\_FABRIC\_ENDPTS, [158](#), [163](#), [185](#), [186](#), [188](#), [189](#), [190](#), [190](#), [311](#), [312](#), [372](#), [532](#)  
PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE, [158](#), [163](#), [186](#), [189](#), [190](#), [312](#), [532](#)  
PMIX\_ALLOC\_FABRIC\_ID, [185](#), [188](#), [190](#), [190](#), [311](#), [372](#), [531](#)  
PMIX\_ALLOC\_FABRIC\_PLANE, [158](#), [163](#), [185](#), [188](#), [189](#), [190](#), [190](#), [311](#), [312](#), [373](#), [531](#)  
PMIX\_ALLOC\_FABRIC\_QOS, [157](#), [163](#), [185](#), [188](#), [189](#), [190](#), [190](#), [311](#), [312](#), [373](#), [531](#)  
PMIX\_ALLOC\_FABRIC\_SEC\_KEY, [185](#), [186](#), [188](#), [189](#), [190](#), [190](#), [311](#), [373](#), [532](#)  
PMIX\_ALLOC\_FABRIC\_TYPE, [158](#), [163](#), [185](#), [188](#), [189](#), [190](#), [190](#), [311](#), [312](#), [372](#), [373](#), [531](#)  
PMIX\_ALLOC\_ID, [186](#), [189](#), [372](#), [512](#), [513](#), [521](#)  
PMIX\_ALLOC\_MEM\_SIZE, [157](#), [162](#), [185](#), [188](#), [190](#), [372](#)  
PMIX\_ALLOC\_NETWORK (**Deprecated**), [531](#)  
PMIX\_ALLOC\_NETWORK\_ENDPTS (**Deprecated**), [531](#)  
PMIX\_ALLOC\_NETWORK\_ENDPTS\_NODE (**Deprecated**), [532](#)  
PMIX\_ALLOC\_NETWORK\_ID (**Deprecated**), [531](#)  
PMIX\_ALLOC\_NETWORK\_PLANE (**Deprecated**), [531](#)  
PMIX\_ALLOC\_NETWORK\_QOS (**Deprecated**), [531](#)  
PMIX\_ALLOC\_NETWORK\_SEC\_KEY (**Deprecated**), [532](#)  
PMIX\_ALLOC\_NETWORK\_TYPE (**Deprecated**), [531](#)  
PMIX\_ALLOC\_NODE\_LIST, [157](#), [162](#), [185](#), [188](#), [189](#), [372](#)  
PMIX\_ALLOC\_NUM\_CPU\_LIST, [157](#), [162](#), [185](#), [188](#), [189](#), [372](#)  
PMIX\_ALLOC\_NUM\_CPUS, [157](#), [162](#), [184](#), [188](#), [189](#), [372](#)  
PMIX\_ALLOC\_NUM\_NODES, [157](#), [162](#), [184](#), [187](#), [189](#), [372](#)  
PMIX\_ALLOC\_QUEUE, [77](#), [81](#), [83](#), [157](#), [162](#), [189](#), [364](#), [521](#)  
PMIX\_ALLOC\_REQ\_ID, [184](#), [187](#), [189](#), [512](#)  
PMIX\_ALLOC\_TIME, [157](#), [162](#), [184](#), [188](#), [190](#), [372](#)  
PMIX\_ALLOCATED\_NODELIST, [93](#), [286](#)  
PMIX\_ANL\_MAP, [93](#), [95](#), [287](#)  
PMIX\_APP\_ARGV, [96](#), [288](#), [521](#)  
PMIX\_APP\_INFO, [68](#), [70](#), [75](#), [80](#), [91](#), [95](#), [99](#), [288](#), [289](#)

PMIX\_APP\_INFO\_ARRAY, 285, 287, [292](#), 293, 297, 298, 520  
PMIX\_APP\_MAP\_REGEX, [96](#)  
PMIX\_APP\_MAP\_TYPE, [96](#)  
PMIX\_APP\_RANK, [97](#), 290  
PMIX\_APP\_SIZE, [96](#), 287, 297  
PMIX\_APPEND\_ENVAR, 157, 162, [167](#)  
PMIX\_APPLDR, [96](#), 288, 297  
PMIX\_APPNUM, 68, 70, 75, 80, 92, 95, [97](#), 99, 285, 287, 289, 290, 293, 297, 298, 520  
PMIX\_ARCH (**Deprecated**), [533](#)  
PMIX\_ATTR\_UNDEF, [5](#)  
PMIX\_AVAIL\_PHYS\_MEMORY, 85, [99](#), 289  
PMIX\_BINDTO, 155, 160, [164](#), 287, 349  
PMIX\_CLEANUP\_EMPTY, 193, 196, [199](#)  
PMIX\_CLEANUP\_IGNORE, 193, 196, [199](#)  
PMIX\_CLEANUP\_LEAVE\_TOPDIR, 193, 196, [199](#)  
PMIX\_CLEANUP\_RECURSIVE, 193, 196, [199](#)  
PMIX\_CLIENT\_ATTRIBUTES, 76, 81, [85](#), 89, 416, 513, 519  
PMIX\_CLIENT\_AVG\_MEMORY, [85](#)  
PMIX\_CLIENT\_FUNCTIONS, 76, 80, 84, [85](#), 89, 518, 519  
PMIX\_CLUSTER\_ID, [92](#), 286  
PMIX\_CMD\_LINE, [95](#), 521  
PMIX\_COLLECT\_DATA, 63, 65, [66](#), 105, 336  
PMIX\_COLLECT\_GENERATED\_JOB\_INFO, 63, 65, [66](#), 66, 243, 336, 517  
PMIX\_COLLECTIVE\_ALGO, [509](#)  
PMIX\_COLLECTIVE\_ALGO (**Deprecated**), [533](#)  
PMIX\_COLLECTIVE\_ALGO\_REQD, [533](#)  
PMIX\_COLLECTIVE\_ALGO\_REQD (**Deprecated**), [511](#)  
PMIX\_CONNECT\_MAX\_RETRIES, [400](#), 426  
PMIX\_CONNECT\_RETRY\_DELAY, [400](#), 426  
PMIX\_CONNECT\_SYSTEM\_FIRST, 398, [400](#), 426, 430  
PMIX\_CONNECT\_TO\_SYSTEM, 397, [400](#), 426, 430  
PMIX\_COSPAWN\_APP, [424](#)  
PMIX\_CPU\_LIST, 156, 161, [165](#), 351  
PMIX\_CPUS\_PER\_PROC, 156, 161, [165](#), 351  
PMIX\_CPUSET, [98](#), 182, 325  
PMIX\_CRED\_TYPE, [271](#), 380  
PMIX\_CREDENTIAL, [98](#), 366  
PMIX\_CRYPTO\_KEY, [271](#)  
PMIX\_DAEMON\_MEMORY, [85](#)  
PMIX\_DATA\_SCOPE, [67](#), [70](#), [72](#)  
PMIX\_DEBUG\_DAEMONS\_PER\_NODE, 350, 418, 419, [424](#), 424, 523  
PMIX\_DEBUG\_DAEMONS\_PER\_PROC, 350, 418, 419, [424](#), 424, 523  
PMIX\_DEBUG\_JOB (**Deprecated**), [531](#)

PMIX\_DEBUG\_STOP\_IN\_INIT, [404](#), [415](#), [416](#), [419](#), [423](#)  
PMIX\_DEBUG\_STOP\_ON\_EXEC, [404](#), [415](#), [416](#), [423](#)  
PMIX\_DEBUG\_TARGET, [350](#), [416](#), [418](#), [419](#), [424](#), [424](#), [425](#), [523](#), [531](#)  
PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY, [404](#), [416](#), [423](#), [424](#)  
PMIX\_DEBUGGER\_DAEMONS, [350](#), [418](#), [419](#), [424](#)  
PMIX\_DISPLAY\_MAP, [155](#), [160](#), [164](#), [349](#)  
PMIX\_DSTPATH (**Deprecated**), [532](#)  
PMIX\_EMBED\_BARRIER, [60](#), [61](#)  
PMIX\_ENUM\_VALUE, [317](#), [317](#), [513](#), [520](#)  
PMIX\_ERROR\_GROUP\_ABORT (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_COMM (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_GENERAL (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_LOCAL (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_MIGRATE (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_NODE (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_RESOURCE (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_GROUP\_SPAWN (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_HANDLER\_ID (**Deprecated**), [509](#), [511](#)  
PMIX\_ERROR\_NAME (**Deprecated**), [509](#), [511](#)  
PMIX\_EVENT\_ACTION\_TIMEOUT, [135](#), [141](#)  
PMIX\_EVENT\_AFFECTED\_PROC, [133](#), [135](#), [140](#), [421](#)  
PMIX\_EVENT\_AFFECTED\_PROCS, [133](#), [135](#), [140](#), [421](#)  
PMIX\_EVENT\_BASE, [57](#), [279](#), [281](#), [427](#)  
PMIX\_EVENT\_CUSTOM\_RANGE, [132](#), [135](#), [140](#)  
PMIX\_EVENT\_DO\_NOT\_CACHE, [135](#), [140](#)  
PMIX\_EVENT\_HDLR\_AFTER, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_APPEND, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_BEFORE, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_FIRST, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_LAST, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_NAME, [132](#), [134](#)  
PMIX\_EVENT\_HDLR\_PREPEND, [132](#), [134](#)  
PMIX\_EVENT\_NON\_DEFAULT, [135](#), [140](#)  
PMIX\_EVENT\_PROXY, [135](#), [140](#)  
PMIX\_EVENT\_RETURN\_OBJECT, [132](#), [135](#)  
PMIX\_EVENT\_SILENT\_TERMINATION, [167](#)  
PMIX\_EVENT\_TERMINATE\_JOB, [135](#), [140](#)  
PMIX\_EVENT\_TERMINATE\_NODE, [135](#), [140](#)  
PMIX\_EVENT\_TERMINATE\_PROC, [135](#), [140](#)  
PMIX\_EVENT\_TERMINATE\_SESSION, [135](#), [140](#)  
PMIX\_EVENT\_TEXT\_MESSAGE, [135](#), [140](#)

PMIX\_EVENT\_TIMESTAMP, [135](#), 166, 167, 403–405, 421, 422, 528, 529  
PMIX\_EXEC\_AGENT, [407](#), [409](#), 420, 522  
PMIX\_EXIT\_CODE, [97](#), 166, 167, 403–405, 421, 529  
PMIX\_FABRIC\_COORDINATES, [258](#), 524  
PMIX\_FABRIC\_COST\_MATRIX, [254](#), [257](#), 523  
PMIX\_FABRIC\_DEVICE, [243](#), [255](#), [258](#), 258, 525  
PMIX\_FABRIC\_DEVICE\_ADDRESS, [244](#), [256](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_BUS\_TYPE, [244](#), [256](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_COORDINATES, [244](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_DIST, [260](#), 267, 524  
PMIX\_FABRIC\_DEVICE\_DRIVER, [244](#), [255](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_FIRMWARE, [244](#), [255](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_ID, [243](#), [244](#), [255](#), [258](#), [259](#), 260, 305, 525  
PMIX\_FABRIC\_DEVICE\_INDEX, [244](#), [259](#), 393, 525  
PMIX\_FABRIC\_DEVICE\_MTU, [245](#), [256](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_NAME, [243](#), [244](#), [255](#), [259](#), 305, 525  
PMIX\_FABRIC\_DEVICE\_PCI\_DEVID, [244](#), [256](#), [259](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_SPEED, [245](#), [256](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_STATE, [245](#), [256](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_TYPE, [245](#), [256](#), [259](#), 526  
PMIX\_FABRIC\_DEVICE\_VENDOR, [244](#), [255](#), [259](#), 525  
PMIX\_FABRIC\_DEVICE\_VENDORID, [244](#), [259](#), 526  
PMIX\_FABRIC\_DEVICES, [243](#), [258](#)  
PMIX\_FABRIC\_DIMS, [254](#), [258](#), 524  
PMIX\_FABRIC\_ENDPT, [260](#), 524  
PMIX\_FABRIC\_GROUPS, [254](#), [257](#), 523  
PMIX\_FABRIC\_IDENTIFIER, [254](#), [257](#), [261](#), 393, 524  
PMIX\_FABRIC\_INDEX, [253](#), [257](#), [258](#), 524  
PMIX\_FABRIC\_NUM\_DEVICES, [254](#), [258](#), 524  
PMIX\_FABRIC\_PLANE, [255](#), [257](#), 258, 261, 262, 393, 525  
PMIX\_FABRIC\_SHAPE, [255](#), [258](#), 524  
PMIX\_FABRIC\_SHAPE\_STRING, [255](#), [258](#), 525  
PMIX\_FABRIC\_SWITCH, [257](#), [260](#), 525  
PMIX\_FABRIC\_VENDOR, [254](#), [257](#), [261](#), 393, 524  
PMIX\_FABRIC\_VIEW, [258](#)  
PMIX\_FIRST\_ENVAR, [157](#), [162](#), [167](#), 529  
PMIX\_FORKEXEC\_AGENT, [407](#), [409](#), 420, 522  
PMIX\_FWD\_STDDIAG, [402](#), [406](#), [408](#), 512  
PMIX\_FWD\_STDERR, [350](#), [367](#), [401](#), [406](#), [408](#), 410  
PMIX\_FWD\_STDIN, [350](#), [367](#), [401](#), [405](#), [408](#), 410  
PMIX\_FWD\_STDOUT, [350](#), [367](#), [401](#), [406](#), [408](#), 410  
PMIX\_GET\_REFRESH\_CACHE, [71](#), [72](#), 109, 518  
PMIX\_GET\_STATIC\_VALUES, [67](#), [68](#), [70](#), [71](#), [72](#), 518

PMIX\_GLOBAL\_RANK, [97](#), [290](#)  
PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID, [219](#), [221](#), [225](#), [231](#), [234](#), [391](#), [392](#), [528](#)  
PMIX\_GROUP\_CONTEXT\_ID, [219](#), [391](#), [528](#)  
PMIX\_GROUP\_ENDPT\_DATA, [219](#), [391](#), [528](#)  
PMIX\_GROUP\_FT\_COLLECTIVE, [219](#), [221](#), [225](#), [231](#), [234](#), [527](#)  
PMIX\_GROUP\_ID, [218](#), [218](#), [391](#), [527](#)  
PMIX\_GROUP\_LEADER, [219](#), [221](#), [222](#), [225](#), [232](#), [236](#), [527](#)  
PMIX\_GROUP\_LOCAL\_ONLY, [219](#), [221](#), [225](#), [391](#), [528](#)  
PMIX\_GROUP\_MEMBERSHIP, [219](#), [222](#), [391](#)  
PMIX\_GROUP NAMES, [219](#), [528](#)  
PMIX\_GROUP\_NOTIFY\_TERMINATION, [219](#), [221](#), [222](#), [225](#), [227](#), [231](#), [234](#), [527](#)  
PMIX\_GROUP\_OPTIONAL, [219](#), [221](#), [222](#), [225](#), [230](#), [234](#), [391](#), [527](#)  
PMIX\_GRPID, [76](#), [81](#), [112](#), [113](#), [117](#), [119](#), [125](#), [126](#), [184](#), [187](#), [192](#), [195](#), [200](#), [203](#), [206](#), [209](#), [269](#),  
[271](#), [273](#), [274](#), [341](#)–[344](#), [346](#), [348](#), [358](#), [364](#), [366](#), [368](#), [369](#), [372](#), [375](#), [378](#), [380](#), [383](#), [384](#),  
[386](#), [389](#)  
PMIX\_HOST, [154](#), [159](#), [164](#), [348](#)  
PMIX\_HOST\_ATTRIBUTES, [76](#), [81](#), [85](#), [89](#), [416](#), [513](#), [519](#)  
PMIX\_HOST\_FUNCTIONS, [76](#), [80](#), [84](#), [85](#), [89](#), [518](#), [519](#)  
PMIX\_HOSTFILE, [154](#), [160](#), [164](#), [348](#)  
PMIX\_HOSTNAME, [68](#), [70](#), [75](#), [77](#), [80](#), [82](#), [84](#), [85](#), [92](#), [95](#), [98](#), [98](#), [99](#), [243](#), [244](#), [255](#), [256](#), [259](#),  
[285](#), [288](#)–[291](#), [293](#), [305](#), [364](#), [417](#), [425](#), [520](#), [526](#)  
PMIX\_HOSTNAME\_ALIASES, [98](#), [289](#), [521](#)  
PMIX\_HOSTNAME\_KEEP\_FQDN, [98](#), [287](#), [522](#)  
PMIX\_HWLOC\_HOLE\_KIND (Deprecated), [532](#)  
PMIX\_HWLOC\_SHARE\_TOPO (Deprecated), [532](#)  
PMIX\_HWLOC\_SHMEM\_ADDR (Deprecated), [532](#)  
PMIX\_HWLOC\_SHMEM\_FILE (Deprecated), [532](#)  
PMIX\_HWLOC\_SHMEM\_SIZE (Deprecated), [532](#)  
PMIX\_HWLOC\_XML\_V1 (Deprecated), [532](#)  
PMIX\_HWLOC\_XML\_V2 (Deprecated), [532](#)  
PMIX\_IMMEDIATE, [67](#), [70](#), [72](#), [100](#), [109](#)  
PMIX\_INDEX\_ARGV, [156](#), [161](#), [165](#), [351](#)  
PMIX\_IOF\_BUFFERING\_SIZE, [386](#), [402](#), [407](#), [414](#), [434](#), [437](#)  
PMIX\_IOF\_BUFFERING\_TIME, [386](#), [403](#), [407](#), [414](#), [434](#), [437](#)  
PMIX\_IOF\_CACHE\_SIZE, [386](#), [402](#), [406](#), [414](#), [433](#), [436](#)  
PMIX\_IOF\_COMPLETE, [387](#), [412](#), [414](#), [414](#), [437](#), [447](#), [523](#)  
PMIX\_IOF\_COPY, [411](#), [414](#), [523](#)  
PMIX\_IOF\_DROP\_NEWEST, [386](#), [402](#), [407](#), [414](#), [433](#), [437](#)  
PMIX\_IOF\_DROP\_OLDEST, [386](#), [402](#), [406](#), [414](#), [433](#), [437](#)  
PMIX\_IOF\_PUSH\_STDIN, [412](#), [414](#), [437](#), [522](#)  
PMIX\_IOF\_REDIRECT, [411](#), [415](#), [523](#)  
PMIX\_IOF\_TAG\_OUTPUT, [403](#), [407](#), [411](#), [414](#), [434](#)  
PMIX\_IOF\_TIMESTAMP\_OUTPUT, [403](#), [407](#), [411](#), [414](#), [434](#)

PMIX\_IOF\_XML\_OUTPUT, 403, 407, 411, [414](#), 434  
PMIX\_JOB\_CONTINUOUS, 156, 161, [166](#), 351  
PMIX\_JOB\_CTRL\_CANCEL, 193, 196, [198](#), 375  
PMIX\_JOB\_CTRL\_CHECKPOINT, 194, 196, [198](#), 375  
PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT, 194, 196, [198](#), 376  
PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD, 194, 197, [198](#), 376  
PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL, 194, 197, [198](#), 376  
PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT, 194, 197, [198](#), 376  
PMIX\_JOB\_CTRL\_ID, 192, 193, 195, 196, [198](#), 198, 375  
PMIX\_JOB\_CTRL\_KILL, 193, 196, [198](#), 375  
PMIX\_JOB\_CTRL\_PAUSE, 193, 196, [198](#), 375  
PMIX\_JOB\_CTRL\_PREEMPTIBLE, 194, 197, [199](#), 376  
PMIX\_JOB\_CTRL\_PROVISION, 194, 197, [199](#), 376  
PMIX\_JOB\_CTRL\_PROVISION\_IMAGE, 194, 197, [199](#), 376  
PMIX\_JOB\_CTRL\_RESTART, 193, 196, [198](#), 375  
PMIX\_JOB\_CTRL\_RESUME, 193, 196, [198](#), 375  
PMIX\_JOB\_CTRL\_SIGNAL, 193, 196, [199](#), 375  
PMIX\_JOB\_CTRL\_TERMINATE, 193, 196, [199](#), 375  
PMIX\_JOB\_INFO, 68, 70, 75, 80, [91](#), 94  
PMIX\_JOB\_INFO\_ARRAY, 284, 286, [292](#), 293, 297, 510, 520  
PMIX\_JOB\_NUM\_APPS, [95](#), 287, 297  
PMIX\_JOB\_RECOVERABLE, 156, 161, [166](#), 351  
PMIX\_JOB\_SIZE, [95](#), 286, 297, 509, 512  
PMIX\_JOB\_TERM\_STATUS, 166, 167, 403–405, 421, 422, [423](#), 529  
PMIX\_JOBID, [94](#), 284–286, 292, 293, 297, 422, 520  
PMIX\_LAUNCHER, 395, [399](#), 401  
PMIX\_LAUNCHER\_DAEMON, 407, [409](#), 522  
PMIX\_LAUNCHER\_RENDEZVOUS\_FILE, 396, [400](#), 522  
PMIX\_LOCAL\_CPUSETS, [99](#), 289, 301  
PMIX\_LOCAL\_PEERS, [99](#), 99, 289, 299  
PMIX\_LOCAL\_PROCS, [99](#), 289  
PMIX\_LOCAL\_RANK, [97](#), 290, 418–420, 424, 425, 523  
PMIX\_LOCAL\_SIZE, [99](#), 289  
PMIX\_LOCAL\_TOPO (Deprecated), [532](#)  
PMIX\_LOCALITY (Deprecated), [532](#)  
PMIX\_LOCALITY\_STRING, 180, [181](#), 291, 324  
PMIX\_LOCALLDR, [95](#), 289  
PMIX\_LOG\_COMPLETION, [167](#), 404, 421, 529  
PMIX\_LOG\_EMAIL, 207, 210, [212](#), 370  
PMIX\_LOG\_EMAIL\_ADDR, 207, 210, [212](#), 370  
PMIX\_LOG\_EMAIL\_MSG, 207, 210, [212](#), 370  
PMIX\_LOG\_EMAIL\_SENDER\_ADDR, [212](#)  
PMIX\_LOG\_EMAIL\_SERVER, [212](#)

PMIX\_LOG\_EMAIL\_SRVR\_PORT, [212](#)  
PMIX\_LOG\_EMAIL SUBJECT, [207](#), [210](#), [212](#), [370](#)  
PMIX\_LOG\_GENERATE\_TIMESTAMP, [207](#), [210](#), [211](#)  
PMIX\_LOG\_GLOBAL\_DATASTORE, [207](#), [210](#), [212](#)  
PMIX\_LOG\_GLOBAL\_SYSLOG, [206](#), [209](#), [211](#)  
PMIX\_LOG\_JOB\_EVENTS, [166](#), [403](#), [421](#), [529](#)  
PMIX\_LOG\_JOB\_RECORD, [207](#), [210](#), [212](#)  
PMIX\_LOG\_LOCAL\_SYSLOG, [206](#), [209](#), [211](#)  
PMIX\_LOG\_MSG, [212](#), [370](#)  
PMIX\_LOG\_ONCE, [206](#), [209](#), [212](#)  
PMIX\_LOG\_PROC\_ABNORMAL\_TERMINATION, [166](#), [529](#)  
PMIX\_LOG\_PROC\_TERMINATION, [166](#), [529](#)  
PMIX\_LOG\_SOURCE, [207](#), [210](#), [211](#)  
PMIX\_LOG\_STDERR, [206](#), [209](#), [211](#), [369](#)  
PMIX\_LOG\_STDOUT, [206](#), [209](#), [211](#), [369](#)  
PMIX\_LOG\_SYSLOG, [206](#), [209](#), [211](#), [369](#)  
PMIX\_LOG\_SYSLOG\_PRI, [206](#), [209](#), [211](#)  
PMIX\_LOG\_TAG\_OUTPUT, [207](#), [210](#), [211](#)  
PMIX\_LOG\_TIMESTAMP, [207](#), [210](#), [211](#)  
PMIX\_LOG\_TIMESTAMP\_OUTPUT, [207](#), [210](#), [211](#)  
PMIX\_LOG\_XML\_OUTPUT, [207](#), [210](#), [211](#)  
PMIX\_MAP\_BLOB (**Deprecated**), [533](#)  
PMIX\_MAPBY, [155](#), [160](#), [164](#), [287](#), [349](#)  
PMIX\_MAPPER, [164](#), [349](#)  
PMIX\_MAPPER (**Deprecated**), [533](#)  
PMIX\_MAX\_PROCS, [93](#), [93–96](#), [285–287](#), [290](#), [316](#), [512](#)  
PMIX\_MAX\_RESTARTS, [156](#), [161](#), [166](#), [351](#)  
PMIX\_MAX\_VALUE, [317](#), [317](#), [513](#), [520](#)  
PMIX\_MERGE\_STDERR\_STDOUT, [156](#), [161](#), [165](#), [350](#)  
PMIX\_MIN\_VALUE, [317](#), [317](#), [513](#), [520](#)  
PMIX\_MODEL\_AFFINITY\_POLICY, [60](#)  
PMIX\_MODEL\_CPU\_TYPE, [59](#)  
PMIX\_MODEL\_LIBRARY\_NAME, [59](#), [288](#), [312](#)  
PMIX\_MODEL\_LIBRARY\_VERSION, [59](#), [288](#), [312](#)  
PMIX\_MODEL\_NUM\_CPUS, [59](#)  
PMIX\_MODEL\_NUM\_THREADS, [59](#)  
PMIX\_MODEL\_PHASE\_NAME, [59](#)  
PMIX\_MODEL\_PHASE\_TYPE, [60](#)  
PMIX\_MONITOR\_APP\_CONTROL, [200](#), [203](#), [204](#), [378](#)  
PMIX\_MONITOR\_CANCEL, [200](#), [203](#), [204](#), [378](#)  
PMIX\_MONITOR\_FILE, [201](#), [203](#), [205](#), [378](#)  
PMIX\_MONITOR\_FILE\_ACCESS, [201](#), [203](#), [205](#), [378](#)  
PMIX\_MONITOR\_FILE\_CHECK\_TIME, [201](#), [203](#), [205](#), [379](#)

PMIX\_MONITOR\_FILE\_DROPS, 201, 203, [205](#), 379  
PMIX\_MONITOR\_FILE MODIFY, 201, 203, [205](#), 378  
PMIX\_MONITOR\_FILE\_SIZE, 201, 203, [205](#), 378  
PMIX\_MONITOR\_HEARTBEAT, 201, 203, [204](#), 378  
PMIX\_MONITOR\_HEARTBEAT\_DROPS, 201, 203, [205](#), 378  
PMIX\_MONITOR\_HEARTBEAT\_TIME, 201, 203, [205](#), 378  
PMIX\_MONITOR\_ID, 200, 203, [204](#), 378  
PMIX\_NO\_OVERSUBSCRIBE, 156, 161, [165](#), 351  
PMIX\_NO\_PROCS\_ON\_HEAD, 156, 161, [165](#), 351  
PMIX\_NODE\_INFO, 68, 70, 75, 80, [92](#), 98, 290  
PMIX\_NODE\_INFO\_ARRAY, 285, 288, [293](#), 293, 297, 299, 304, 305, 520  
PMIX\_NODE\_LIST, [93](#), 95, 96  
PMIX\_NODE\_MAP, [93](#), 94, 96, 286, 296–298, 312, 313, 512  
PMIX\_NODE\_RANK, [97](#), 290, 419  
PMIX\_NODE\_SIZE, [99](#), 289  
PMIX\_NODEID, 68, 70, 75, 80, 85, 92, 95, [98](#), 98, 99, 244, 256, 259, 285, 288–290, 293, 305, 520, 526  
PMIX\_NOHUP, 403, 407, [408](#), 522  
PMIX\_NON\_PMI, 349  
PMIX\_NON\_PMI (Deprecated), [533](#)  
PMIX\_NOTIFY\_COMPLETION, [166](#), 403, 421  
PMIX\_NOTIFY\_JOB\_EVENTS, [166](#), 403, 421, 528  
PMIX\_NOTIFY\_PROC\_ABNORMAL\_TERMINATION, [166](#), 529  
PMIX\_NOTIFY\_PROC\_TERMINATION, [166](#), 529  
PMIX\_NPROC\_OFFSET, [94](#), 287  
PMIX\_NSDIR, [95](#), 98, 289, 291  
PMIX\_NSPACE, 75–77, 80–85, [94](#), 284–286, 292, 293, 297, 364, 365, 417, 422, 425, 518–520  
PMIX\_NUM\_ALLOCATED\_NODES, [93](#), 521  
PMIX\_NUM\_NODES, [91](#), [93](#), 94, 96, 296, 297, 521  
PMIX\_NUM\_SLOTS, [93](#), 94, 96  
PMIX\_OPTIONAL, 67, 70, [72](#), 109  
PMIX\_OUTPUT\_TO\_DIRECTORY, [165](#), 528  
PMIX\_OUTPUT\_TO\_FILE, 156, 161, [165](#), 351  
PMIX\_PACKAGE\_RANK, [97](#), 521  
PMIX\_PARENT\_ID, [97](#), 348  
PMIX\_PERSISTENCE, 112, [114](#), 114, 341  
PMIX\_PERSONALITY, 155, 160, [164](#), 349  
PMIX\_PPR, 155, 160, [164](#), 349  
PMIX\_PREFIX, 154, 159, [164](#), 348  
PMIX\_PRELOAD\_BIN, 155, 160, [164](#), 349  
PMIX\_PRELOAD\_FILES, 155, 160, [164](#), 349  
PMIX\_PREPEND\_ENVAR, 157, 162, [167](#)  
PMIX\_PRIMARY\_SERVER, [400](#), 430, 431, 522

PMIX\_PROC\_BLOB (**Degraded**), [533](#)  
PMIX\_PROC\_DATA (**Degraded**), [532](#)  
PMIX\_PROC\_INFO, [92](#)  
PMIX\_PROC\_INFO\_ARRAY, [285](#), [290](#), [293](#), [298](#), [520](#), [532](#)  
PMIX\_PROC\_MAP, [93](#), [95](#), [96](#), [287](#), [296](#), [297](#), [312](#), [313](#), [512](#)  
PMIX\_PROC\_PID, [98](#)  
PMIX\_PROC\_STATE\_STATUS, [423](#)  
PMIX\_PROC\_TERM\_STATUS, [422](#), [423](#)  
PMIX\_PROC\_URI, [78](#), [82](#)  
PMIX\_PROC\_URI (**Degraded**), [533](#)  
PMIX\_PROCDIR, [98](#), [291](#)  
PMIX\_PROCID, [75](#)–[77](#), [80](#)–[82](#), [84](#), [97](#), [166](#), [167](#), [285](#), [293](#), [365](#), [403](#)–[405](#), [421](#), [520](#), [529](#)  
PMIX\_PROGRAMMING\_MODEL, [59](#), [288](#), [312](#)  
PMIX\_PSET\_MEMBERS, [214](#), [215](#), [527](#)  
PMIX\_PSET\_NAME, [214](#), [215](#), [527](#)  
PMIX\_PSET\_NAMES, [213](#), [215](#), [288](#), [527](#)  
PMIX\_QUERY\_ALLOC\_STATUS, [77](#), [82](#), [84](#), [365](#)  
PMIX\_QUERY\_ATTRIBUTE\_SUPPORT, [76](#), [80](#), [84](#), [88](#), [416](#), [518](#)  
PMIX\_QUERY\_AUTHORIZATIONS, [84](#)  
PMIX\_QUERY\_AVAIL\_SERVERS, [85](#), [398](#), [518](#)  
PMIX\_QUERY\_DEBUG\_SUPPORT, [77](#), [82](#), [84](#), [364](#)  
PMIX\_QUERY\_GROUP\_MEMBERSHIP, [218](#), [527](#)  
PMIX\_QUERY\_GROUP\_NAMES, [218](#), [527](#)  
PMIX\_QUERY\_JOB\_STATUS, [76](#), [81](#), [83](#), [364](#)  
PMIX\_QUERY\_LOCAL\_ONLY, [84](#), [365](#)  
PMIX\_QUERY\_LOCAL\_PROC\_TABLE, [77](#), [81](#), [84](#), [364](#), [417](#), [425](#)  
PMIX\_QUERY\_MEMORY\_USAGE, [77](#), [82](#), [84](#), [365](#)  
PMIX\_QUERY\_NAMESPACE\_INFO, [83](#), [518](#)  
PMIX\_QUERY\_NAMESPACES, [76](#), [81](#), [83](#), [364](#), [417](#)  
PMIX\_QUERY\_NUM\_GROUPS, [218](#), [527](#)  
PMIX\_QUERY\_NUM\_PSETS, [84](#), [215](#), [526](#)  
PMIX\_QUERY\_PROC\_TABLE, [77](#), [81](#), [83](#), [364](#), [417](#), [425](#)  
PMIX\_QUERY\_PSET\_MEMBERSHIP, [85](#), [215](#), [527](#)  
PMIX\_QUERY\_PSET\_NAMES, [84](#), [215](#), [526](#)  
PMIX\_QUERY\_QUALIFIERS, [78](#), [83](#), [83](#), [518](#)  
PMIX\_QUERY\_QUEUE\_LIST, [77](#), [81](#), [83](#), [364](#)  
PMIX\_QUERY\_QUEUE\_STATUS, [77](#), [81](#), [83](#), [364](#)  
PMIX\_QUERY\_REFRESH\_CACHE, [75](#), [78](#), [79](#), [83](#), [88](#)  
PMIX\_QUERY\_REPORT\_AVG, [77](#), [82](#), [84](#), [365](#)  
PMIX\_QUERY\_REPORT\_MINMAX, [77](#), [82](#), [84](#), [365](#)  
PMIX\_QUERY\_RESULTS, [78](#), [83](#), [518](#)  
PMIX\_QUERY\_SPAWN\_SUPPORT, [77](#), [82](#), [84](#), [364](#)  
PMIX\_QUERY\_SUPPORTED\_KEYS, [83](#), [518](#)

PMIX\_QUERY\_SUPPORTED\_QUALIFIERS, [83](#), [518](#)  
PMIX\_RANGE, [112](#), [114](#), [114](#), [117](#), [119](#), [125](#), [127](#), [132](#), [201](#), [218](#), [341](#), [343](#), [346](#), [361](#), [391](#), [392](#), [527](#)  
PMIX\_RANK, [76](#), [77](#), [80–82](#), [84](#), [97](#), [285](#), [290](#), [293](#), [365](#), [420](#), [424](#), [520](#)  
PMIX\_RANKBY, [155](#), [160](#), [164](#), [287](#), [349](#)  
PMIX\_RECONNECT\_SERVER (Deprecated), [531](#)  
PMIX\_REGISTER\_CLEANUP, [193](#), [196](#), [199](#)  
PMIX\_REGISTER\_CLEANUP\_DIR, [193](#), [196](#), [199](#)  
PMIX\_REGISTER\_NODATA, [284](#), [292](#)  
PMIX\_REINCarnation, [98](#), [291](#), [521](#)  
PMIX\_REPORT\_BINDINGS, [156](#), [161](#), [165](#), [351](#)  
PMIX\_REQUESTOR\_IS\_CLIENT, [348](#), [352](#)  
PMIX\_REQUESTOR\_IS\_TOOL, [348](#), [352](#)  
PMIX\_REQUIRED\_KEY, [340](#)  
PMIX\_RM\_NAME, [93](#)  
PMIX\_RM\_VERSION, [93](#)  
PMIX\_SEND\_HEARTBEAT, [205](#)  
PMIX\_SERVER\_ATTRIBUTES, [76](#), [81](#), [85](#), [89](#), [513](#), [519](#)  
PMIX\_SERVER\_ENABLE\_MONITORING, [279](#), [281](#)  
PMIX\_SERVER\_FUNCTIONS, [76](#), [80](#), [84](#), [85](#), [89](#), [518](#), [519](#)  
PMIX\_SERVER\_GATEWAY, [277](#), [281](#)  
PMIX\_SERVER\_HOSTNAME, [400](#)  
PMIX\_SERVER\_INFO\_ARRAY, [85](#), [85](#), [519](#)  
PMIX\_SERVER\_NSPACE, [277](#), [281](#), [286](#), [397](#), [426](#), [430](#)  
PMIX\_SERVER\_PIDINFO, [397](#), [400](#), [426](#), [430](#)  
PMIX\_SERVER\_RANK, [277](#), [281](#), [286](#)  
PMIX\_SERVER\_REMOTE\_CONNECTIONS, [278](#), [280](#)  
PMIX\_SERVER\_SCHEDULER, [257](#), [260](#), [277](#), [281](#), [519](#), [523](#)  
PMIX\_SERVER\_SESSION\_SUPPORT, [277](#), [280](#), [519](#)  
PMIX\_SERVER\_SHARE\_TOPOLOGY, [279](#), [280](#), [519](#)  
PMIX\_SERVER\_START\_TIME, [280](#), [519](#)  
PMIX\_SERVER\_SYSTEM\_SUPPORT, [277](#), [280](#), [395](#)  
PMIX\_SERVER\_TMPDIR, [277](#), [279](#), [280](#), [395–397](#)  
PMIX\_SERVER\_TOOL\_SUPPORT, [268](#), [277](#), [279](#), [280](#)  
PMIX\_SERVER\_URI, [78](#), [82](#), [397](#), [398](#), [400](#), [426](#), [430](#)  
PMIX\_SESSION\_ID, [92](#), [92](#), [94](#), [101](#), [284](#), [285](#), [292](#), [296](#), [297](#), [422](#), [520](#)  
PMIX\_SESSION\_INFO, [68](#), [70](#), [75](#), [80](#), [91](#), [93](#), [100](#), [285](#), [286](#)  
PMIX\_SESSION\_INFO\_ARRAY, [284](#), [285](#), [292](#), [293](#), [296](#), [510](#)  
PMIX\_SET\_ENVAR, [156](#), [162](#), [167](#)  
PMIX\_SET\_SESSION\_CWD, [154](#), [159](#), [165](#), [348](#)  
PMIX\_SETUP\_APP\_ALL, [311](#), [314](#)  
PMIX\_SETUP\_APP\_ENVARS, [311](#), [314](#)  
PMIX\_SETUP\_APP\_NONENVARS, [311](#), [314](#)

PMIX\_SINGLE\_LISTENER, [57](#), [278](#), [280](#)  
PMIX\_SOCKET\_MODE, [57](#), [278](#), [280](#), [426](#)  
PMIX\_SPAWN\_TOOL, [166](#)  
PMIX\_SPAWNED, [98](#), [291](#), [348](#)  
PMIX\_STDIN\_TGT, [155](#), [160](#), [165](#), [350](#)  
PMIX\_SWITCH\_PEERS, [260](#), [260](#), [525](#)  
PMIX\_SYSTEM\_TMPDIR, [277](#), [281](#), [395](#), [397](#)  
PMIX\_TAG\_OUTPUT, [155](#), [160](#), [165](#), [350](#)  
PMIX\_TCP\_DISABLE\_IPV4, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_DISABLE\_IPV6, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_IF\_EXCLUDE, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_IF\_INCLUDE, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_IPV4\_PORT, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_IPV6\_PORT, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_REPORT\_URI, [57](#), [59](#), [278](#), [427](#)  
PMIX\_TCP\_URI, [59](#), [397](#), [398](#), [426](#), [430](#)  
PMIX\_TDIR\_RMCLEAN, [93](#)  
PMIX\_THREADING\_MODEL, [59](#)  
PMIX\_TIME\_REMAINING, [73](#), [77](#), [82](#), [84](#), [365](#)  
PMIX\_TIMEOUT, [4](#), [63](#), [66](#), [68](#), [71](#), [72](#), [100](#), [109](#), [110](#), [112](#), [114](#), [117](#), [119](#), [125](#), [127](#), [172](#), [175](#),  
[176](#), [178](#), [217](#), [221](#), [223](#), [226](#), [227](#), [229](#), [231](#), [234](#), [236](#), [238](#), [270](#), [271](#), [273](#), [275](#), [336](#), [339](#),  
[341](#), [344](#), [346](#), [351](#), [353](#), [356](#), [380](#), [383](#)  
PMIX\_TIMEOUT\_REPORT\_STATE, [166](#), [528](#)  
PMIX\_TIMEOUT\_STACKTRACES, [166](#), [528](#)  
PMIX\_TIMESTAMP\_OUTPUT, [155](#), [161](#), [165](#), [350](#)  
PMIX\_TMPDIR, [93](#), [95](#), [289](#)  
PMIX\_TOOL\_ATTACHMENT\_FILE, [397](#), [398](#), [400](#), [426](#), [430](#), [522](#)  
PMIX\_TOOL\_ATTRIBUTES, [76](#), [81](#), [85](#), [89](#), [513](#), [519](#)  
PMIX\_TOOL\_CONNECT\_OPTIONAL, [400](#), [522](#)  
PMIX\_TOOL\_DO\_NOT\_CONNECT, [397](#), [400](#), [426](#), [428](#)  
PMIX\_TOOL\_FUNCTIONS, [76](#), [80](#), [84](#), [85](#), [89](#), [518](#), [519](#)  
PMIX\_TOOL\_NSPACE, [366](#), [396](#), [399](#), [426](#), [428](#)  
PMIX\_TOOL\_RANK, [366](#), [396](#), [399](#), [426](#), [428](#)  
PMIX\_TOPOLOGY (Deprecated), [531](#)  
PMIX\_TOPOLOGY2, [279](#), [280](#), [519](#), [531](#)  
PMIX\_TOPOLOGY\_FILE (Deprecated), [532](#)  
PMIX\_TOPOLOGY\_SIGNATURE (Deprecated), [532](#)  
PMIX\_TOPOLOGY\_XML (Deprecated), [532](#)  
PMIX\_UNIV\_SIZE, [8](#), [92](#), [285](#), [296](#), [509](#), [510](#), [512](#)  
PMIX\_UNSET\_ENVAR, [157](#), [162](#), [167](#)  
PMIX\_USERID, [76](#), [81](#), [112](#), [113](#), [117](#), [119](#), [125](#), [126](#), [184](#), [187](#), [192](#), [195](#), [200](#), [203](#), [206](#), [209](#), [269](#),  
[271](#), [273](#), [274](#), [341](#)–[346](#), [348](#), [357](#), [364](#), [366](#), [368](#), [369](#), [371](#), [375](#), [378](#), [380](#), [382](#), [384](#), [386](#),  
[389](#)

PMIX\_USOCK\_DISABLE, [57](#), [278](#), [280](#)  
PMIX\_VERSION\_INFO, [367](#), [368](#)  
PMIX\_WAIT, [71](#), [72](#), [117](#), [119](#), [344](#)  
PMIX\_WDIR, [154](#), [159](#), [164](#), [288](#), [348](#)