



# Process Management Interface for Exascale (PMIx) Standard

**Version 5.0 (Draft)**

*Created on February 4, 2022*

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 5.0 (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*

Unofficial Draft

# 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
1.3.2. PMIx Roles . . . . .	5
<b>2. PMIx Terms and Conventions</b>	<b>7</b>
2.1. Notational Conventions . . . . .	9
2.2. Semantics . . . . .	10
2.3. Naming Conventions . . . . .	11
2.4. Procedure Conventions . . . . .	11
<b>3. Data Structures and Types</b>	<b>13</b>
3.1. Constants . . . . .	14
3.1.1. PMIx Return Status Constants . . . . .	15
3.1.1.1. User-Defined Error and Event Constants . . . . .	17
3.2. Data Types . . . . .	17
3.2.1. Key Structure . . . . .	17
3.2.1.1. Key support macros . . . . .	18
3.2.2. Namespace Structure . . . . .	19
3.2.2.1. Namespace support macros . . . . .	19
3.2.3. Rank Structure . . . . .	20
3.2.3.1. Rank support macros . . . . .	21
3.2.4. Process Structure . . . . .	21
3.2.4.1. Process structure support macros . . . . .	22
3.2.5. Process State Structure . . . . .	25
3.2.6. Process Information Structure . . . . .	26
3.2.6.1. Process information structure support macros . . . . .	27

3.2.7.	Job State Structure . . . . .	28
3.2.8.	Value Structure . . . . .	29
3.2.8.1.	Value structure support macros . . . . .	30
3.2.9.	Info Structure . . . . .	34
3.2.9.1.	Info structure support macros . . . . .	34
3.2.9.2.	Info structure list macros . . . . .	36
3.2.10.	Info Type Directives . . . . .	39
3.2.10.1.	Info Directive support macros . . . . .	40
3.2.11.	Environmental Variable Structure . . . . .	42
3.2.11.1.	Environmental variable support macros . . . . .	42
3.2.12.	Byte Object Type . . . . .	44
3.2.12.1.	Byte object support macros . . . . .	44
3.2.13.	Data Array Structure . . . . .	45
3.2.13.1.	Data array support macros . . . . .	45
3.2.14.	Argument Array Macros . . . . .	47
3.2.15.	Set Environment Variable . . . . .	50
3.3.	Generalized Data Types Used for Packing/Unpacking . . . . .	51
3.4.	General Callback Functions . . . . .	53
3.4.1.	Release Callback Function . . . . .	53
3.4.2.	Op Callback Function . . . . .	54
3.4.3.	Value Callback Function . . . . .	54
3.4.4.	Info Callback Function . . . . .	55
3.4.5.	Handler registration callback function . . . . .	55
3.5.	PMIx Datatype Value String Representations . . . . .	56
<b>4.</b>	<b>Client Initialization and Finalization</b>	<b>60</b>
4.1.	<b>PMIx_Initialized</b> . . . . .	60
4.2.	<b>PMIx_Get_version</b> . . . . .	61
4.3.	<b>PMIx_Init</b> . . . . .	61
4.3.1.	Initialization events . . . . .	64
4.3.2.	Initialization attributes . . . . .	64
4.3.2.1.	Connection attributes . . . . .	64
4.3.2.2.	Programming model attributes . . . . .	65

4.4.	<b>PMIx_Finalize</b> . . . . .	66
4.4.1.	Finalize attributes . . . . .	66
4.5.	<b>PMIx_Progress</b> . . . . .	66
<b>5.</b>	<b>Synchronization and Data Access Operations</b>	<b>68</b>
5.1.	<b>PMIx_Fence</b> . . . . .	68
5.2.	<b>PMIx_Fence_nb</b> . . . . .	70
5.2.1.	Fence-related attributes . . . . .	72
5.3.	<b>PMIx_Get</b> . . . . .	73
5.3.1.	<b>PMIx_Get_nb</b> . . . . .	75
5.3.2.	Retrieval attributes . . . . .	78
<b>6.</b>	<b>Reserved Keys</b>	<b>80</b>
6.1.	Data realms . . . . .	80
6.1.1.	Session realm attributes . . . . .	81
6.1.2.	Job realm attributes . . . . .	83
6.1.3.	Application realm attributes . . . . .	85
6.1.4.	Process realm attributes . . . . .	86
6.1.5.	Node realm keys . . . . .	87
6.2.	Retrieval rules for reserved keys . . . . .	89
6.2.1.	Accessing information: examples . . . . .	89
6.2.1.1.	Session-level information . . . . .	90
6.2.1.2.	Job-level information . . . . .	91
6.2.1.3.	Application-level information . . . . .	91
6.2.1.4.	Process-level information . . . . .	92
6.2.1.5.	Node-level information . . . . .	92
<b>7.</b>	<b>Process-Related Non-Reserved Keys</b>	<b>94</b>
7.1.	Posting Key/Value Pairs . . . . .	95
7.1.1.	<b>PMIx_Put</b> . . . . .	95
7.1.1.1.	Scope of Put Data . . . . .	96
7.1.2.	<b>PMIx_Store_internal</b> . . . . .	96
7.1.3.	<b>PMIx_Commit</b> . . . . .	97
7.2.	Retrieval rules for non-reserved keys . . . . .	98

<b>8. Query Operations</b>	<b>100</b>
8.1. <code>PMIx_Query_info</code> . . . . .	100
8.1.1. Query Structure . . . . .	101
8.1.2. <code>PMIx_Query_info</code> . . . . .	101
8.1.3. <code>PMIx_Query_info_nb</code> . . . . .	106
8.1.4. Query keys . . . . .	110
8.1.5. Query attributes . . . . .	112
8.1.5.1. Query structure support macros . . . . .	113
8.2. <code>PMIx_Resolve_peers</code> . . . . .	115
8.2.1. <code>PMIx_Resolve_nodes</code> . . . . .	116
8.3. Using Get vs Query . . . . .	116
8.4. Accessing attribute support information . . . . .	117
<b>9. Publish/Lookup Operations</b>	<b>119</b>
9.1. <code>PMIx_Publish</code> . . . . .	119
9.2. <code>PMIx_Publish_nb</code> . . . . .	121
9.3. Publish-specific constants . . . . .	122
9.4. Publish-specific attributes . . . . .	122
9.5. Publish-Lookup Datatypes . . . . .	123
9.5.1. Range of Published Data . . . . .	123
9.5.2. Data Persistence Structure . . . . .	123
9.6. <code>PMIx_Lookup</code> . . . . .	124
9.7. <code>PMIx_Lookup_nb</code> . . . . .	125
9.7.1. Lookup Returned Data Structure . . . . .	127
9.7.1.1. Lookup data structure support macros . . . . .	128
9.7.2. Lookup Callback Function . . . . .	130
9.8. Retrieval rules for published data . . . . .	131
9.9. <code>PMIx_Unpublish</code> . . . . .	132
9.10. <code>PMIx_Unpublish_nb</code> . . . . .	133
<b>10. Event Notification</b>	<b>135</b>
10.1. Notification and Management . . . . .	135
10.1.1. Events versus status constants . . . . .	137
10.1.2. <code>PMIx_Register_event_handler</code> . . . . .	137

10.1.3.	Event registration constants . . . . .	140
10.1.4.	System events . . . . .	141
10.1.5.	Event handler registration and notification attributes . . . . .	141
10.1.5.1.	Fault tolerance event attributes . . . . .	142
10.1.5.2.	Hybrid programming event attributes . . . . .	142
10.1.6.	Notification Function . . . . .	143
10.1.7.	<b>PMIx_Deregister_event_handler</b> . . . . .	144
10.1.8.	<b>PMIx_Notify_event</b> . . . . .	145
10.1.9.	Notification Handler Completion Callback Function . . . . .	149
10.1.9.1.	Completion Callback Function Status Codes . . . . .	149
<b>11.</b>	<b>Data Packing and Unpacking</b> . . . . .	<b>150</b>
11.1.	Data Buffer Type . . . . .	150
11.2.	Support Macros . . . . .	151
11.3.	General Routines . . . . .	152
11.3.1.	<b>PMIx_Data_pack</b> . . . . .	152
11.3.2.	<b>PMIx_Data_unpack</b> . . . . .	154
11.3.3.	<b>PMIx_Data_copy</b> . . . . .	156
11.3.4.	<b>PMIx_Data_print</b> . . . . .	156
11.3.5.	<b>PMIx_Data_copy_payload</b> . . . . .	157
11.3.6.	<b>PMIx_Data_load</b> . . . . .	158
11.3.7.	<b>PMIx_Data_unload</b> . . . . .	159
11.3.8.	<b>PMIx_Data_compress</b> . . . . .	159
11.3.9.	<b>PMIx_Data_decompress</b> . . . . .	160
<b>12.</b>	<b>Process Management</b> . . . . .	<b>162</b>
12.1.	Abort . . . . .	162
12.1.1.	<b>PMIx_Abort</b> . . . . .	162
12.2.	Process Creation . . . . .	163
12.2.1.	<b>PMIx_Spawn</b> . . . . .	164
12.2.2.	<b>PMIx_Spawn_nb</b> . . . . .	169
12.2.3.	Spawn-specific constants . . . . .	174
12.2.4.	Spawn attributes . . . . .	175

12.2.5.	Application Structure . . . . .	178
12.2.5.1.	App structure support macros . . . . .	179
12.2.5.2.	Spawn Callback Function . . . . .	181
12.3.	Connecting and Disconnecting Processes . . . . .	181
12.3.1.	<b>PMIx_Connect</b> . . . . .	182
12.3.2.	<b>PMIx_Connect_nb</b> . . . . .	184
12.3.3.	<b>PMIx_Disconnect</b> . . . . .	186
12.3.4.	<b>PMIx_Disconnect_nb</b> . . . . .	188
12.4.	Process Locality . . . . .	190
12.4.1.	<b>PMIx_Load_topology</b> . . . . .	190
12.4.2.	<b>PMIx_Get_relative_locality</b> . . . . .	191
12.4.2.1.	Topology description . . . . .	191
12.4.2.2.	Topology support macros . . . . .	192
12.4.2.3.	Relative locality of two processes . . . . .	193
12.4.2.4.	Locality keys . . . . .	193
12.4.3.	<b>PMIx_Parse_cpuset_string</b> . . . . .	193
12.4.4.	<b>PMIx_Get_cpuset</b> . . . . .	194
12.4.4.1.	Binding envelope . . . . .	194
12.4.5.	<b>PMIx_Compute_distances</b> . . . . .	195
12.4.6.	<b>PMIx_Compute_distances_nb</b> . . . . .	196
12.4.7.	Device Distance Callback Function . . . . .	197
12.4.8.	Device type . . . . .	197
12.4.9.	Device Distance Structure . . . . .	198
12.4.10.	Device distance support macros . . . . .	199
12.4.11.	Device distance attributes . . . . .	200
<b>13.</b>	<b>Job Management and Reporting</b> . . . . .	<b>201</b>
13.1.	Allocation Requests . . . . .	201
13.1.1.	<b>PMIx_Allocation_request</b> . . . . .	201
13.1.2.	<b>PMIx_Allocation_request_nb</b> . . . . .	204
13.1.3.	Job Allocation attributes . . . . .	207
13.1.4.	Job Allocation Directives . . . . .	209
13.2.	Job Control . . . . .	209
13.2.1.	<b>PMIx_Job_control</b> . . . . .	210



13.2.2.	<b>PMIx_Job_control_nb</b>	212
13.2.3.	Job control constants	215
13.2.4.	Job control events	215
13.2.5.	Job control attributes	216
13.3.	Process and Job Monitoring	217
13.3.1.	<b>PMIx_Process_monitor</b>	217
13.3.2.	<b>PMIx_Process_monitor_nb</b>	219
13.3.3.	<b>PMIx_Heartbeat</b>	221
13.3.4.	Monitoring events	222
13.3.5.	Monitoring attributes	222
13.4.	Logging	223
13.4.1.	<b>PMIx_Log</b>	223
13.4.2.	<b>PMIx_Log_nb</b>	226
13.4.3.	Log attributes	229
<b>14.</b>	<b>Process Sets and Groups</b>	<b>231</b>
14.1.	Process Sets	231
14.1.1.	Process Set Constants	232
14.1.2.	Process Set Attributes	233
14.2.	Process Groups	233
14.2.1.	Relation to the host environment	233
14.2.2.	Construction procedure	234
14.2.3.	Destruct procedure	235
14.2.4.	Process Group Events	235
14.2.5.	Process Group Attributes	236
14.2.6.	<b>PMIx_Group_construct</b>	238
14.2.7.	<b>PMIx_Group_construct_nb</b>	241
14.2.8.	<b>PMIx_Group_destruct</b>	244
14.2.9.	<b>PMIx_Group_destruct_nb</b>	245
14.2.10.	<b>PMIx_Group_invite</b>	247
14.2.11.	<b>PMIx_Group_invite_nb</b>	250
14.2.12.	<b>PMIx_Group_join</b>	252
14.2.13.	<b>PMIx_Group_join_nb</b>	254
14.2.13.1.	Group accept/decline directives	256

14.2.14. <b>PMIx_Group_leave</b> . . . . .	256
14.2.15. <b>PMIx_Group_leave_nb</b> . . . . .	257
<b>15. Fabric Support Definitions</b>	<b>259</b>
15.1. Fabric Support Events . . . . .	262
15.2. Fabric Support Datatypes . . . . .	262
15.2.1. Fabric Endpoint Structure . . . . .	262
15.2.2. Fabric endpoint support macros . . . . .	263
15.2.3. Fabric Coordinate Structure . . . . .	264
15.2.4. Fabric coordinate support macros . . . . .	264
15.2.5. Fabric Geometry Structure . . . . .	266
15.2.6. Fabric geometry support macros . . . . .	266
15.2.7. Fabric Coordinate Views . . . . .	267
15.2.8. Fabric Link State . . . . .	268
15.2.9. Fabric Operation Constants . . . . .	268
15.2.10. Fabric registration structure . . . . .	269
15.2.10.1. Initialize the fabric structure . . . . .	272
15.3. Fabric Support Attributes . . . . .	272
15.4. Fabric Support Functions . . . . .	275
15.4.1. <b>PMIx_Fabric_register</b> . . . . .	276
15.4.2. <b>PMIx_Fabric_register_nb</b> . . . . .	277
15.4.3. <b>PMIx_Fabric_update</b> . . . . .	278
15.4.4. <b>PMIx_Fabric_update_nb</b> . . . . .	279
15.4.5. <b>PMIx_Fabric_deregister</b> . . . . .	279
15.4.6. <b>PMIx_Fabric_deregister_nb</b> . . . . .	280
<b>16. Security</b>	<b>281</b>
16.1. Obtaining Credentials . . . . .	281
16.1.1. <b>PMIx_Get_credential</b> . . . . .	282
16.1.2. <b>PMIx_Get_credential_nb</b> . . . . .	283
16.1.3. Credential Attributes . . . . .	284
16.2. Validating Credentials . . . . .	285
16.2.1. <b>PMIx_Validate_credential</b> . . . . .	285
16.2.2. <b>PMIx_Validate_credential_nb</b> . . . . .	286

<b>17. Server-Specific Interfaces</b>	<b>289</b>
17.1. Server Initialization and Finalization	289
17.1.1. <b>PMIx_server_init</b>	289
17.1.2. <b>PMIx_server_finalize</b>	293
17.1.3. Server Initialization Attributes	293
17.2. Server Support Functions	294
17.2.1. <b>PMIx_generate_regex</b>	294
17.2.2. <b>PMIx_generate_ppn</b>	296
17.2.3. <b>PMIx_server_register_namespace</b>	296
17.2.3.1. Namespace registration attributes	307
17.2.3.2. Assembling the registration information	308
17.2.4. <b>PMIx_server_deregister_namespace</b>	317
17.2.5. <b>PMIx_server_register_resources</b>	317
17.2.6. <b>PMIx_server_deregister_resources</b>	318
17.2.7. <b>PMIx_server_register_client</b>	319
17.2.8. <b>PMIx_server_deregister_client</b>	321
17.2.9. <b>PMIx_server_setup_fork</b>	322
17.2.10. <b>PMIx_server_dmodex_request</b>	322
17.2.10.1. Server Direct Modex Response Callback Function	324
17.2.11. <b>PMIx_server_setup_application</b>	324
17.2.11.1. Server Setup Application Callback Function	328
17.2.11.2. Server Setup Application Attributes	328
17.2.12. <b>PMIx_Register_attributes</b>	329
17.2.12.1. Attribute registration constants	330
17.2.12.2. Attribute registration structure	330
17.2.12.3. Attribute registration structure descriptive attributes	331
17.2.12.4. Attribute registration structure support macros	331
17.2.13. <b>PMIx_server_setup_local_support</b>	333
17.2.14. <b>PMIx_server_IOF_deliver</b>	335
17.2.15. <b>PMIx_server_collect_inventory</b>	336
17.2.16. <b>PMIx_server_deliver_inventory</b>	337
17.2.17. <b>PMIx_server_generate_locality_string</b>	338

17.2.18.	<b>PMIx_server_generate_cpuset_string</b>	339
17.2.18.1.	Cpuset Structure	340
17.2.18.2.	Cpuset support macros	340
17.2.19.	<b>PMIx_server_define_process_set</b>	341
17.2.20.	<b>PMIx_server_delete_process_set</b>	342
17.3.	Server Function Pointers	342
17.3.1.	<b>pmix_server_module_t</b> Module	343
17.3.2.	<b>pmix_server_client_connected_fn_t</b>	344
17.3.3.	<b>pmix_server_client_connected2_fn_t</b>	345
17.3.4.	<b>pmix_server_client_finalized_fn_t</b>	347
17.3.5.	<b>pmix_server_abort_fn_t</b>	348
17.3.6.	<b>pmix_server_fencenb_fn_t</b>	350
17.3.6.1.	Modex Callback Function	353
17.3.7.	<b>pmix_server_dmodex_req_fn_t</b>	353
17.3.7.1.	Dmodex attributes	355
17.3.8.	<b>pmix_server_publish_fn_t</b>	355
17.3.9.	<b>pmix_server_lookup_fn_t</b>	357
17.3.10.	<b>pmix_server_unpublish_fn_t</b>	360
17.3.11.	<b>pmix_server_spawn_fn_t</b>	362
17.3.11.1.	Server spawn attributes	367
17.3.12.	<b>pmix_server_connect_fn_t</b>	367
17.3.13.	<b>pmix_server_disconnect_fn_t</b>	368
17.3.14.	<b>pmix_server_register_events_fn_t</b>	370
17.3.15.	<b>pmix_server_deregister_events_fn_t</b>	372
17.3.16.	<b>pmix_server_notify_event_fn_t</b>	374
17.3.17.	<b>pmix_server_listener_fn_t</b>	375
17.3.17.1.	PMIx Client Connection Callback Function	376
17.3.18.	<b>pmix_server_query_fn_t</b>	377
17.3.19.	<b>pmix_server_tool_connection_fn_t</b>	379
17.3.19.1.	Tool connection attributes	382
17.3.19.2.	PMIx Tool Connection Callback Function	382
17.3.20.	<b>pmix_server_log_fn_t</b>	382
17.3.21.	<b>pmix_server_alloc_fn_t</b>	384

17.3.22.	<code>pmix_server_job_control_fn_t</code>	387
17.3.23.	<code>pmix_server_monitor_fn_t</code>	390
17.3.24.	<code>pmix_server_get_cred_fn_t</code>	393
17.3.24.1.	Credential callback function	394
17.3.25.	<code>pmix_server_validate_cred_fn_t</code>	395
17.3.26.	Credential validation callback function	397
17.3.27.	<code>pmix_server_iof_fn_t</code>	398
17.3.27.1.	IOF delivery function	401
17.3.28.	<code>pmix_server_stdin_fn_t</code>	402
17.3.29.	<code>pmix_server_grp_fn_t</code>	403
17.3.29.1.	Group Operation Constants	406
17.3.30.	<code>pmix_server_fabric_fn_t</code>	406
<b>18.</b>	<b>Tools and Debuggers</b>	<b>408</b>
18.1.	Connection Mechanisms	408
18.1.1.	Rendezvousing with a local server	411
18.1.2.	Connecting to a remote server	412
18.1.3.	Attaching to running jobs	413
18.1.4.	Tool initialization attributes	413
18.1.5.	Tool initialization environmental variables	414
18.1.6.	Tool connection attributes	414
18.2.	Launching Applications with Tools	415
18.2.1.	Direct launch	415
18.2.2.	Indirect launch	419
18.2.2.1.	Initiator-based command line parsing	420
18.2.2.2.	Intermediate Launcher (IL)-based command line parsing	423
18.2.3.	Tool spawn-related attributes	424
18.2.4.	Tool rendezvous-related events	425
18.3.	IO Forwarding	425
18.3.1.	Forwarding stdout/stderr	426
18.3.2.	Forwarding stdin	428
18.3.3.	IO Forwarding Channels	429
18.3.4.	IO Forwarding constants	430
18.3.5.	IO Forwarding attributes	430

18.4. Debugger Support . . . . .	431
18.4.1. Co-Location of Debugger Daemons . . . . .	433
18.4.2. Co-Spawn of Debugger Daemons . . . . .	435
18.4.3. Debugger Agents . . . . .	436
18.4.4. Tracking the job lifecycle . . . . .	437
18.4.4.1. Job lifecycle events . . . . .	438
18.4.4.2. Job lifecycle attributes . . . . .	439
18.4.5. Debugger-related constants . . . . .	439
18.4.6. Debugger attributes . . . . .	439
18.5. Tool-Specific APIs . . . . .	441
18.5.1. <code>PMIx_tool_init</code> . . . . .	441
18.5.2. <code>PMIx_tool_finalize</code> . . . . .	444
18.5.3. <code>PMIx_tool_disconnect</code> . . . . .	445
18.5.4. <code>PMIx_tool_attach_to_server</code> . . . . .	446
18.5.5. <code>PMIx_tool_get_servers</code> . . . . .	447
18.5.6. <code>PMIx_tool_set_server</code> . . . . .	448
18.5.7. <code>PMIx_IOF_pull</code> . . . . .	449
18.5.8. <code>PMIx_IOF_deregister</code> . . . . .	451
18.5.9. <code>PMIx_IOF_push</code> . . . . .	452
<b>19. Storage Support Definitions</b> . . . . .	<b>455</b>
19.1. Storage support constants . . . . .	455
19.2. Storage support attributes . . . . .	457
<b>A. Python Bindings</b> . . . . .	<b>459</b>
A.1. Design Considerations . . . . .	459
A.1.1. Error Codes vs Python Exceptions . . . . .	459
A.1.2. Representation of Structured Data . . . . .	459
A.2. Datatype Definitions . . . . .	460
A.2.1. Example . . . . .	466
A.3. Callback Function Definitions . . . . .	467
A.3.1. IOF Delivery Function . . . . .	467
A.3.2. Event Handler . . . . .	467

A.3.3.	Server Module Functions . . . . .	468
A.3.3.1.	Client Connected . . . . .	468
A.3.3.2.	Client Finalized . . . . .	469
A.3.3.3.	Client Aborted . . . . .	469
A.3.3.4.	Fence . . . . .	470
A.3.3.5.	Direct Modex . . . . .	471
A.3.3.6.	Publish . . . . .	471
A.3.3.7.	Lookup . . . . .	472
A.3.3.8.	Unpublish . . . . .	472
A.3.3.9.	Spawn . . . . .	473
A.3.3.10.	Connect . . . . .	473
A.3.3.11.	Disconnect . . . . .	474
A.3.3.12.	Register Events . . . . .	474
A.3.3.13.	Deregister Events . . . . .	475
A.3.3.14.	Notify Event . . . . .	475
A.3.3.15.	Query . . . . .	475
A.3.3.16.	Tool Connected . . . . .	476
A.3.3.17.	Log . . . . .	476
A.3.3.18.	Allocate Resources . . . . .	477
A.3.3.19.	Job Control . . . . .	477
A.3.3.20.	Monitor . . . . .	478
A.3.3.21.	Get Credential . . . . .	478
A.3.3.22.	Validate Credential . . . . .	479
A.3.3.23.	IO Forward . . . . .	479
A.3.3.24.	IO Push . . . . .	480
A.3.3.25.	Group Operations . . . . .	480
A.3.3.26.	Fabric Operations . . . . .	481
A.4.	PMIxClient . . . . .	482
A.4.1.	Client.init . . . . .	482
A.4.2.	Client.initialized . . . . .	482
A.4.3.	Client.get_version . . . . .	483
A.4.4.	Client.finalize . . . . .	483
A.4.5.	Client.abort . . . . .	483

A.4.6.	Client.store_internal . . . . .	484
A.4.7.	Client.put . . . . .	484
A.4.8.	Client.commit . . . . .	485
A.4.9.	Client.fence . . . . .	485
A.4.10.	Client.get . . . . .	486
A.4.11.	Client.publish . . . . .	486
A.4.12.	Client.lookup . . . . .	487
A.4.13.	Client.unpublish . . . . .	487
A.4.14.	Client.spawn . . . . .	488
A.4.15.	Client.connect . . . . .	488
A.4.16.	Client.disconnect . . . . .	489
A.4.17.	Client.resolve_peers . . . . .	489
A.4.18.	Client.resolve_nodes . . . . .	490
A.4.19.	Client.query . . . . .	490
A.4.20.	Client.log . . . . .	491
A.4.21.	Client.allocation_request . . . . .	491
A.4.22.	Client.job_ctrl . . . . .	492
A.4.23.	Client.monitor . . . . .	492
A.4.24.	Client.get_credential . . . . .	493
A.4.25.	Client.validate_credential . . . . .	493
A.4.26.	Client.group_construct . . . . .	494
A.4.27.	Client.group_invite . . . . .	494
A.4.28.	Client.group_join . . . . .	495
A.4.29.	Client.group_leave . . . . .	496
A.4.30.	Client.group_destruct . . . . .	496
A.4.31.	Client.register_event_handler . . . . .	496
A.4.32.	Client.deregister_event_handler . . . . .	497
A.4.33.	Client.notify_event . . . . .	497
A.4.34.	Client.fabric_register . . . . .	498
A.4.35.	Client.fabric_update . . . . .	498
A.4.36.	Client.fabric_deregister . . . . .	499
A.4.37.	Client.load_topology . . . . .	499
A.4.38.	Client.get_relative_locality . . . . .	500



A.4.39.	Client.get_cpuset . . . . .	500
A.4.40.	Client.parse_cpuset_string . . . . .	500
A.4.41.	Client.compute_distances . . . . .	501
A.4.42.	Client.error_string . . . . .	501
A.4.43.	Client.proc_state_string . . . . .	502
A.4.44.	Client.scope_string . . . . .	502
A.4.45.	Client.persistence_string . . . . .	503
A.4.46.	Client.data_range_string . . . . .	503
A.4.47.	Client.info_directives_string . . . . .	503
A.4.48.	Client.data_type_string . . . . .	504
A.4.49.	Client.alloc_directive_string . . . . .	504
A.4.50.	Client.iof_channel_string . . . . .	505
A.4.51.	Client.job_state_string . . . . .	505
A.4.52.	Client.get_attribute_string . . . . .	505
A.4.53.	Client.get_attribute_name . . . . .	506
A.4.54.	Client.link_state_string . . . . .	506
A.4.55.	Client.device_type_string . . . . .	507
A.4.56.	Client.progress . . . . .	507
A.5.	PMIxServer . . . . .	507
A.5.1.	Server.init . . . . .	507
A.5.2.	Server.finalize . . . . .	508
A.5.3.	Server.generate_regex . . . . .	508
A.5.4.	Server.generate_ppn . . . . .	509
A.5.5.	Server.generate_locality_string . . . . .	509
A.5.6.	Server.generate_cpuset_string . . . . .	510
A.5.7.	Server.register_nspace . . . . .	510
A.5.8.	Server.deregister_nspace . . . . .	511
A.5.9.	Server.register_resources . . . . .	511
A.5.10.	Server.deregister_resources . . . . .	512
A.5.11.	Server.register_client . . . . .	512
A.5.12.	Server.deregister_client . . . . .	513
A.5.13.	Server.setup_fork . . . . .	513
A.5.14.	Server.dmodex_request . . . . .	513

A.5.15.	Server.setup_application . . . . .	514
A.5.16.	Server.register_attributes . . . . .	514
A.5.17.	Server.setup_local_support . . . . .	515
A.5.18.	Server.iof_deliver . . . . .	515
A.5.19.	Server.collect_inventory . . . . .	516
A.5.20.	Server.deliver_inventory . . . . .	516
A.5.21.	Server.define_process_set . . . . .	517
A.5.22.	Server.delete_process_set . . . . .	517
A.5.23.	Server.register_resources . . . . .	518
A.5.24.	Server.deregister_resources . . . . .	518
A.6.	PMIxTool . . . . .	519
A.6.1.	Tool.init . . . . .	519
A.6.2.	Tool.finalize . . . . .	519
A.6.3.	Tool.disconnect . . . . .	519
A.6.4.	Tool.attach_to_server . . . . .	520
A.6.5.	Tool.get_servers . . . . .	520
A.6.6.	Tool.set_server . . . . .	521
A.6.7.	Tool.iof_pull . . . . .	521
A.6.8.	Tool.iof_deregister . . . . .	522
A.6.9.	Tool.iof_push . . . . .	522
A.7.	Example Usage . . . . .	523
A.7.1.	Python Client . . . . .	523
A.7.2.	Python Server . . . . .	525
<b>B.</b>	<b>Use-Cases</b>	<b>529</b>
B.1.	Business Card Exchange for Process-to-Process Wire-up . . . . .	529
B.1.1.	Use Case Summary . . . . .	529
B.1.2.	Use Case Details . . . . .	530
B.2.	Debugging . . . . .	533
B.2.1.	Terminology . . . . .	533
B.2.1.1.	Tools vs Debuggers . . . . .	533
B.2.1.2.	Parallel Launching Methods . . . . .	534
B.2.1.3.	Process Synchronization . . . . .	534
B.2.1.4.	Process Acquisition . . . . .	534

B.2.2.	Use Case Details . . . . .	534
B.2.2.1.	Direct-Launch Debugger Tool . . . . .	534
B.2.2.2.	Indirect-Launch Debugger Tool . . . . .	539
B.2.2.3.	Attaching to a Running Job . . . . .	545
B.2.2.4.	Tool Interaction with RM . . . . .	548
B.2.2.5.	Environmental Parameter Directives for Applications and Launchers	550
B.3.	Hybrid Applications . . . . .	551
B.3.1.	Use Case Summary . . . . .	551
B.3.2.	Use Case Details . . . . .	551
B.3.2.1.	Identifying Active Parallel Runtime Systems . . . . .	551
B.3.2.2.	Coordinating at Runtime . . . . .	553
B.3.2.3.	Coordinating at Runtime with Multiple Event Handlers . . . . .	555
B.4.	MPI Sessions . . . . .	558
B.4.1.	Use Case Summary . . . . .	558
B.4.2.	Use Case Details . . . . .	559
<b>C.</b>	<b>Revision History</b>	<b>562</b>
C.1.	Version 1.0: June 12, 2015 . . . . .	562
C.2.	Version 2.0: Sept. 2018 . . . . .	563
C.2.1.	Removed/Modified Application Programming Interfaces (APIs) . . . . .	563
C.2.2.	Deprecated constants . . . . .	563
C.2.3.	Deprecated attributes . . . . .	564
C.3.	Version 2.1: Dec. 2018 . . . . .	564
C.4.	Version 2.2: Jan 2019 . . . . .	565
C.5.	Version 3.0: Dec. 2018 . . . . .	565
C.5.1.	Removed constants . . . . .	566
C.5.2.	Deprecated attributes . . . . .	566
C.5.3.	Removed attributes . . . . .	566
C.6.	Version 3.1: Jan. 2019 . . . . .	567
C.7.	Version 3.2: Oct. 2020 . . . . .	567
C.7.1.	Deprecated constants . . . . .	568
C.7.2.	Deprecated attributes . . . . .	569
C.8.	Version 4.0: Dec. 2020 . . . . .	570
C.8.1.	Added Constants . . . . .	572

C.8.2.	Added Attributes . . . . .	575
C.8.3.	Added Environmental Variables . . . . .	588
C.8.4.	Added Macros . . . . .	588
C.8.5.	Deprecated APIs . . . . .	588
C.8.6.	Deprecated constants . . . . .	589
C.8.7.	Removed constants . . . . .	589
C.8.8.	Deprecated attributes . . . . .	590
C.8.9.	Removed attributes . . . . .	591
C.9.	Version 4.1: TBD . . . . .	592
C.9.1.	Added Functions (Provisional) . . . . .	592
C.9.2.	Added Data Structures (Provisional) . . . . .	592
C.9.3.	Added Macros (Provisional) . . . . .	592
C.9.4.	Added Constants (Provisional) . . . . .	593
C.9.5.	Added Attributes (Provisional) . . . . .	593
<b>D.</b>	<b>Acknowledgements</b>	<b>596</b>
D.1.	Version 4.0 . . . . .	596
D.2.	Version 3.0 . . . . .	597
D.3.	Version 2.0 . . . . .	598
D.4.	Version 1.0 . . . . .	599
	<b>Bibliography</b>	<b>601</b>
	<b>Index</b>	<b>602</b>
	<b>Index of APIs</b>	<b>604</b>
	<b>Index of Support Macros</b>	<b>612</b>
	<b>Index of Data Structures</b>	<b>616</b>
	<b>Index of Constants</b>	<b>618</b>
	<b>Index of Environmental Variables</b>	<b>628</b>
	<b>Index of Attributes</b>	<b>629</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.

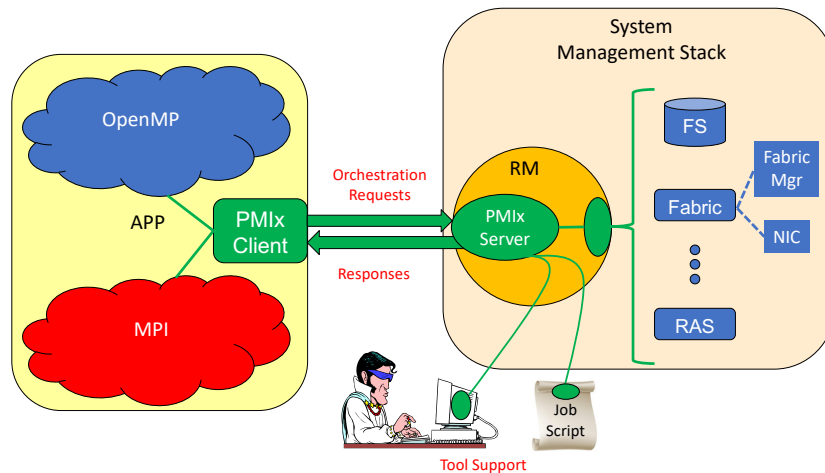


Figure 1.1.: PMIx-SMS Interactions

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 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 8.4). 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 implementation 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.

## 19 1.3.2 PMIx Roles

20 The role of a PMIx process in the PMIx universe is grouped into one of three categories based on  
21 how it operates in the PMIx environment namely as a *client*, *server*, or *tool*. As a result, there are  
22 three corresponding groupings of APIs each with their own initialization and finalization functions.  
23 If a process initializes as either a *server* or a *tool* that process may also access all of the *client* APIs.

24 A process operating as a *client* is connected to the PMIx server instance within an RM when the  
25 client calls the client PMIx initialization routine. The *client* is typically started directly or indirectly  
26 (for example, by an intermediate script) by that RM. Additionally, a *client* may be started directly  
27 by the user and then connect to an RM which is typically referred to as a *singleton* launch. A  
28 process operating as a *server* is responsible for starting client processes and coordinating with other  
29 server and tool processes in the same PMIx universe. Often processes operating as a *server* are part  
30 of the Resource Manager (RM) infrastructure. A process operating as a *tool* is started  
31 independently (e.g., via `fork/exec`) or by the RM and will connect to a PMIx *server* to interact with  
32 the processes in the PMIx universe. An example of a *tool* process is a parallel debugger that will  
33 connect to the server to assist with attaching to a set of client processes.

34 PMIx serves as a conduit between processes acting in these three different roles. As such, an API is  
35 often described by how it interacts with processes operating in other roles in the PMIx universe.

## Advice to PMIx library implementers

1 A PMIx implementation may support all or a subset of the API role groupings defined in the  
2 standard. A common nomenclature is defined here to aid in identifying levels of conformance of an  
3 implementation.

4 Note that it would not make sense for an implementation to exclude the *client* interfaces from their  
5 implementation since they are also used by the *server* and *tool* roles. Therefore the *client* interfaces  
6 represent the minimal set of required functionality for PMIx compliance.

7 A PMIx implementation that supports only the *client* APIs is said to be *client-role PMIx standard*  
8 *compliant*. Similarly, a PMIx implementation that only supports the *client* and *tool* APIs is said to  
9 be *client-role and tool-role PMIx standard compliant*. Finally, a PMIx implementation that only  
10 supports the *client* and *server* APIs is said to be *client-role and server-role PMIx standard*  
11 *compliant*.

12 A PMIx implementation that supports all three sets of the API role groupings is said to be  
13 *client-role, server-role, and tool-role PMIx standard compliant*. These *client-role, server-role, and*  
14 *tool-role PMIx standard compliant* implementations have the advantage of being able to support a  
15 broad set of PMIx consumers in the different roles.

## CHAPTER 2

# PMIx Terms and Conventions

---

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

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

13 The following terminology is used throughout this document:

- 14 • *session* refers to a set of resources assigned by the WorkLoad Manager (WLM) that has been  
15 reserved for one or more users. A session is identified by a *session ID* that is unique within the  
16 scope of the governing WLMs. Historically, High Performance Computing (HPC) sessions have  
17 consisted of a static allocation of resources - i.e., a block of resources assigned to a user in  
18 response to a specific request and managed as a unified collection. However, this is changing in  
19 response to the growing use of dynamic programming models that require on-the-fly allocation  
20 and release of system resources. Accordingly, the term *session* in this document refers to a  
21 potentially dynamic entity, perhaps comprised of resources accumulated as a result of multiple  
22 allocation requests that are managed as a single unit by the WLM.
- 23 • *job* refers to a set of one or more *applications* executed as a single invocation by the user within a  
24 session with a unique identifier, the *job ID*, assigned by the RM or launcher. For example, the  
25 command line “*mpiexec -n 1 app1 : -n 2 app2*” generates a single Multiple Program Multiple  
26 Data (MPMD) job containing two applications. A user may execute multiple *jobs* within a given  
27 session, either sequentially or concurrently.
- 28 • *namespace* refers to a character string value assigned by the RM to a *job*. All *applications*  
29 executed as part of that *job* share the same *namespace*. The *namespace* assigned to each *job* must  
30 be unique within the scope of the governing RM and often is implemented as a string  
31 representation of the numerical *emphJob* ID. The *namespace* and *job* terms will be used  
32 interchangeably throughout the document.

- 1       • *application* represents a set of identical, but not necessarily unique, execution contexts within a  
2       *job*.
- 3       • *process* is assumed for ease of presentation to be an operating system process, also commonly  
4       referred to as a *heavyweight* process. A process is often comprised of multiple *lightweight*  
5       *threads*, commonly known as simply *threads*. However, it is not the intent of the PMIx Standard  
6       to restrict the term process to a particular concept or implementation.
- 7       • *client* refers to a process that was registered with the PMIx server prior to being started, and  
8       connects to that PMIx server via **PMIx\_Init** using its assigned namespace and rank with the  
9       information required to connect to that server being provided to the process at time of start of  
10      execution.
- 11      • *clone* refers to a process that was directly started by a PMIx client (e.g., using *fork/exec*) and calls  
12      **PMIx\_Init**, thus connecting to its local PMIx server using the same namespace and rank as its  
13      parent process.
- 14      • *rank* refers to the numerical location (starting from zero) of a process within the defined scope.  
15      Thus, *job rank* is the rank of a process within its *job* and is synonymous with its unqualified  
16      *rank*, while *application rank* is the rank of that process within its *application*.
- 17      • *peer* refers to another process within the same *job*.
- 18      • *workflow* refers to an orchestrated execution plan typically involving multiple *jobs* carried out  
19      under the control of a *workflow manager*. An example workflow might first execute a  
20      computational job to generate the flow of liquid through a complex cavity, followed by a  
21      visualization job that takes the output of the first job as its input to produce an image output.
- 22      • *scheduler* refers to the component of the SMS responsible for scheduling of resource allocations.  
23      This is also generally referred to as the *system workflow manager* - for the purposes of this  
24      document, the *WLM* acronym will be used interchangeably to refer to the scheduler.
- 25      • *resource manager* is used in a generic sense to represent the subsystem that will host the PMIx  
26      server library. This could be a vendor-supplied resource manager or a third-party agent such as a  
27      programming model's runtime library.
- 28      • *host environment* is used interchangeably with *resource manager* to refer to the process hosting  
29      the PMIx server library.
- 30      • *node* refers to a single operating system instance. Note that this may encompass one or more  
31      physical objects.
- 32      • *package* refers to a single object that is either soldered or connected to a printed circuit board via  
33      a mechanical socket. Packages may contain multiple chips that include (but are not limited to)  
34      processing units, memory, and peripheral interfaces.
- 35      • *processing unit*, or *PU*, is the electronic circuitry within a computer that executes instructions.  
36      Depending upon architecture and configuration settings, it may consist of a single hardware  
37      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:

C specific text...

```
int foo = 42;
```

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

Note: General text...

### Rationale

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

### Advice to users

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

### Advice to PMIx library implementers

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

### Advice to PMIx server hosts

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

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

## 2.2 Semantics

The following terms will be taken to mean:

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

## 1 2.3 Naming Conventions

2 The PMIx standard has adopted the following conventions:

- 3 • PMIx constants and attributes are prefixed with "PMIX\_".
- 4 • Structures and type definitions are prefixed with "pmix\_".
- 5 • The string representation of attributes are prefixed with "pmix".
- 6 • Underscores are used to separate words in a function or variable name.
- 7 • Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the
- 8 first letter of the word following it. For example, `PMIx_Get_version`.
- 9 • PMIx server and tool APIs are all lower case letters following the prefix - e.g.,
- 10 `PMIx_server_register_namespace`.
- 11 • The `PMIX_` prefix is used to denote functions.
- 12 • The `pmix_` prefix is used to denote function pointer and type definitions.

13 Users shall not use the "PMIX\_", "PMIx\_", or "pmix\_" prefixes for symbols in their code so as

14 to avoid symbol conflicts with PMIx implementations.

## 15 2.4 Procedure Conventions

16 While the current APIs are based on the C programming language, it is not the intent of the PMIx

17 Standard to preclude the use of other languages. Accordingly, the procedure specifications in the

18 PMIx Standard are written in a language-independent syntax with the arguments marked as IN,

19 OUT, or INOUT. The meanings of these are:

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

24 Many PMIx interfaces, particularly nonblocking interfaces, use a **(void\*)** callback data object

25 passed to the function that is then passed to the associated callback. On the client side, the callback

26 data object is an opaque, client-provided context that the client can pass to a non-blocking call.

27 When the nonblocking call completes, the callback data object is passed back to the client without

28 modification by the PMIx library, thus allowing the client to associate a context with that callback.

29 This is useful if there are many outstanding nonblocking calls.

30 A similar model is used for the server module functions (see [17.3.1](#)). In this case, the PMIx library

31 is making an upcall into its host via the PMIx server module callback function and passing a

32 specific callback function pointer and callback data object. The PMIx library expects the host to

33 call the cbfunc with the necessary arguments and pass back the original callback data object upon

1 completing the operation. This gives the server-side PMIx library the ability to associate a context  
2 with the call back (since multiple operations may be outstanding). The host has no visibility into  
3 the contents of the callback data object object, nor is permitted to alter it in any way.

Unofficial Draft



## CHAPTER 3

# Data Structures and Types

---

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

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

### ▼ Advice to PMIx library implementers ▼

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

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



### ▼ Advice to users ▼

13 Use of increment/decrement operations on indices inside PMIx macros is discouraged due to  
14 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);
```

17 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)
```

26 since the index is cited more than once in the macro. The PMIx standard only governs the existence  
27 and syntax of macros - it does not specify their implementation. Given the freedom of  
28 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.

### 1 3.1.1 PMIx Return Status Constants

2 The `pmix_status_t` type is an `int` compatible value for return status values. PMIx return  
3 values other than `PMIX_SUCCESS` are required to always be negative. The return status value for a  
4 successful operation is `PMIX_SUCCESS`, which must have an integer value of 0:

5 `PMIX_SUCCESS` Success.

▼ Advice to PMIx library implementers ▼

6 A PMIx implementation must define all of the return status constants defined in the PMIx standard,  
7 even if the implementation will never return the specific value to the caller.

▲

▼ Advice to users ▼

8 Other than `PMIX_SUCCESS` (which is required to be zero), the integer value of any PMIx error  
9 constant is left to the PMIx library implementer with the constraint that it be negative and greater  
10 magnitude (i.e. of larger absolute value) than `PMIX_EXTERNAL_ERR_BASE`. Thus, users are  
11 advised to always refer to constants by name, and not by a specific implementation's integer value,  
12 for portability between implementations and compatibility across library versions.

▲

13 The presentation of each API in this document includes a list of return status constants which are  
14 either specific to that API or are expected to be returned by the API in normal use.

15 In addition, the following are general constants covering a variety of possible reasons an  
16 implementation of an API may return a constant other than one of the constants presented with the  
17 API. Although implementations can define and return additional error constants, implementations  
18 are encouraged to return one of the return constants listed with the API or in the list presented here  
19 to encourage portability across implementations.

20 `PMIX_ERROR` General Error.

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

23 `PMIX_ERR_EXISTS_OUTSIDE_SCOPE` The requested key exists, but was posted in a *scope*  
24 (see Section 7.1.1.1) that does not include the requester

25 `PMIX_ERR_INVALID_CRED` Invalid security credentials.

26 `PMIX_ERR_WOULD_BLOCK` Operation would block.

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

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

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

1 **PMIX\_ERR\_UNPACK\_READ\_PAST\_END\_OF\_BUFFER** Unpacking past the end of the  
2 provided buffer - the number of values in the buffer is less than the specified number to  
3 unpack, or a request was made to unpack a buffer beyond the buffer's end.

4 **PMIX\_ERR\_UNPACK\_FAILURE** The unpack operation failed for an unspecified reason.  
5 **PMIX\_ERR\_PACK\_FAILURE** The pack operation failed for an unspecified reason.  
6 **PMIX\_ERR\_NO\_PERMISSIONS** The user lacks permissions to execute the specified  
7 operation.

8 **PMIX\_ERR\_TIMEOUT** Either a user-specified or system-internal timeout expired.  
9 **PMIX\_ERR\_UNREACH** The specified target server or client process is not reachable - i.e., a  
10 suitable connection either has not been or can not be made.

11 **PMIX\_ERR\_BAD\_PARAM** One or more incorrect parameters (e.g., passing an attribute with a  
12 value of the wrong type), or multiple parameters containing conflicting directives (e.g.,  
13 multiple instances of the same attribute with different values, or different attributes specifying  
14 conflicting behaviors), were passed to a PMIx API.

15 **PMIX\_ERR\_EMPTY** An array or list was given that has no members in it - i.e., the object is  
16 empty.

17 **PMIX\_ERR\_RESOURCE\_BUSY** Resource busy - typically seen when an attempt to establish a  
18 connection to another process (e.g., a PMIx server) cannot be made due to a communication  
19 failure.

20 **PMIX\_ERR\_OUT\_OF\_RESOURCE** Resource exhausted.  
21 **PMIX\_ERR\_INIT** The requested operation requires that the PMIx library be initialized prior  
22 to being called.

23 **PMIX\_ERR\_NOMEM** Out of memory.  
24 **PMIX\_ERR\_NOT\_FOUND** The requested information was not found.  
25 **PMIX\_ERR\_NOT\_SUPPORTED** The requested operation is not supported by either the PMIx  
26 implementation or the host environment.

27 **PMIX\_ERR\_PARAM\_VALUE\_NOT\_SUPPORTED** The requested operation is supported by the  
28 PMIx implementation and (if applicable) the host environment. However, at least one  
29 supplied parameter was given an unsupported value, and the operation cannot therefore be  
30 executed as requested.

31 **PMIX\_ERR\_COMM\_FAILURE** Communication failure - a message failed to be sent or  
32 received, but the connection remains intact.

33 **PMIX\_ERR\_LOST\_CONNECTION** Lost connection between server and client or tool.  
34 **PMIX\_ERR\_INVALID\_OPERATION** The requested operation is supported by the  
35 implementation and host environment, but fails to meet a requirement (e.g., requesting to  
36 *disconnect* from processes without first *connecting* to them, inclusion of conflicting  
37 directives, or a request to perform an operation that conflicts with an ongoing one).

38 **PMIX\_OPERATION\_IN\_PROGRESS** A requested operation is already in progress - the  
39 duplicate request shall therefore be ignored.  
40 **PMIX\_OPERATION\_SUCCEEDED** The requested operation was performed atomically - no  
41 callback function will be executed.

1 **PMIX\_ERR\_PARTIAL\_SUCCESS** The operation is considered successful but not all elements  
2 of the operation were concluded (e.g., some members of a group construct operation chose  
3 not to participate).

### 4 3.1.1.1 User-Defined Error and Event Constants

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

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

## 13 3.2 Data Types

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

### 16 3.2.1 Key Structure

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

PMIx v2.0

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

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

#### Advice to users

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

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

### 1 3.2.1.1 Key support macros

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

#### 3 Check key macro

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

*PMIx v3.0*

▼ C ————— ▼

5 **PMIX\_CHECK\_KEY**(a, b)

▲ C ————— ▲

6 **IN a**

7 Pointer to the structure whose key is to be checked (pointer to `pmix_info_t`)

8 **IN b**

9 String value to be compared against (**char\***)

10 Returns **true** if the key matches the given value

#### 11 Check reserved key macro

12 Check if the given key is a PMIx *reserved* key as described in Chapter 6.

*PMIx v4.0*

▼ C ————— ▼

13 **PMIX\_CHECK\_RESERVED\_KEY**(a)

▲ C ————— ▲

14 **IN a**

15 String value to be checked (**char\***)

16 Returns **true** if the key is reserved by the Standard.

#### 17 Load key macro

18 Load a key into a `pmix_info_t`.

*PMIx v4.0*

▼ C ————— ▼

19 **PMIX\_LOAD\_KEY**(a, b)

▲ C ————— ▲

20 **IN a**

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

22 **IN b**

23 String value to be loaded (**char\***)

24 No return value.

## 1 3.2.2 Namespace Structure

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

```
▼ C  
5 typedef char pmix_nspace_t[PMIX_MAX_NSLEN+1];  
▲ C
```

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

### ▼ Advice to users ▲

8 References to namespace values in PMIx v1 were defined simply as an array of characters of size  
9 `PMIX_MAX_NSLEN+1`. The `pmix_nspace_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_nspace_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_nspace_t)` and instead rely  
14 on the `PMIX_MAX_NSLEN` constant.

### 15 3.2.2.1 Namespace support macros

16 The following macros are provided for convenience when working with PMIx namespace  
17 structures.

#### 18 Check namespace macro

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

*PMIx v3.0*

```
▼ C  
20 PMIX_CHECK_NAMESPACE(a, b)  
▲ C
```

21 **IN a**  
22 Pointer to the structure whose value is to be checked (pointer to `pmix_nspace_t`)

23 **IN b**  
24 String value to be compared against (`char*`)

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

## Check invalid namespace macro

Check if the provided `pmix_namespace_t` is invalid.

▼ C ▼

**PMIX\_NAMESPACE\_INVALID (a)**

▲ C ▲

**IN a**

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

Returns `true` if the namespace is invalid (i.e., starts with a `NULL` resulting in a zero-length string value)

## Load namespace macro

Load a namespace into a `pmix_namespace_t`.

*PMIx v4.0*

▼ C ▼

**PMIX\_LOAD\_NAMESPACE (a, b)**

▲ C ▲

**IN a**

Pointer to the target structure (pointer to `pmix_namespace_t`)

**IN b**

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

No return value.

## 3.2.3 Rank Structure

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

*PMIx v1.0*

▼ C ▼

**typedef uint32\_t pmix\_rank\_t;**

▲ C ▲

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

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

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

**PMIX\_RANK\_LOCAL\_NODE** Special rank value used to define groups of ranks. This constant defines the group of all ranks on a local node.



1 **PMIX\_RANK\_LOCAL\_PEERS** Special rank value used to define groups of ranks. This  
 2 constant defines the group of all ranks on a local node within the same namespace as the  
 3 current process.  
 4 **PMIX\_RANK\_INVALID** An invalid rank value.  
 5 **PMIX\_RANK\_VALID** Define an upper boundary for valid rank values.

### 6 3.2.3.1 Rank support macros

7 The following macros are provided for convenience when working with PMIx ranks.

#### 8 **Check rank macro**

9 Check two ranks for equality, taking into account wildcard values

*PMIx v4.0*

▼ **PMIX\_CHECK\_RANK**(a, b) C

10 **PMIX\_CHECK\_RANK**(a, b)

▲ C

11 **IN** a  
 12 Rank to be checked (**pmix\_rank\_t**)

13 **IN** b  
 14 Rank to be checked (**pmix\_rank\_t**)

15 Returns **true** if the ranks are equal, or at least one of the ranks is **PMIX\_RANK\_WILDCARD**

#### 16 **Check rank is valid macro**

17 Check if the given rank is a valid value

**Provisional**  
**v4.1**

▼ **PMIX\_RANK\_IS\_VALID**(a) C

18 **PMIX\_RANK\_IS\_VALID**(a)

▲ C

19 **IN** a  
 20 Rank to be checked (**pmix\_rank\_t**)

21 Returns **true** if the given rank is valid (i.e., less than **PMIX\_RANK\_VALID**)

### 22 3.2.4 Process Structure

23 The **pmix\_proc\_t** structure is used to identify a single process in the PMIx universe. It contains  
 24 a reference to the namespace and the **pmix\_rank\_t** within that namespace.

*PMIx v1.0*

▼ C

```
25 typedef struct pmix_proc {
26     pmix_namespace_t nspace;
27     pmix_rank_t rank;
28 } pmix_proc_t;
```

▲ C

### 1 3.2.4.1 Process structure support macros

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

#### 3 Initialize the proc structure

4 Initialize the `pmix_proc_t` fields.

*PMIx v1.0*

▼ `PMIX_PROC_CONSTRUCT` (m) C

5 **PMIX\_PROC\_CONSTRUCT** (m)

▲ C

6 **IN** m

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

#### 8 Destruct the proc structure

9 Destruct the `pmix_proc_t` fields.

▼ `PMIX_PROC_DESTRUCT` (m) C

10 **PMIX\_PROC\_DESTRUCT** (m)

▲ C

11 **IN** m

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

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

#### 16 Create a proc array

17 Allocate and initialize an array of `pmix_proc_t` structures.

*PMIx v1.0*

▼ `PMIX_PROC_CREATE` (m, n) C

18 **PMIX\_PROC\_CREATE** (m, n)

▲ C

19 **INOUT** m

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

21 **IN** n

22 Number of structures to be allocated (`size_t`)

#### 23 Free a proc structure

24 Release a `pmix_proc_t` structure.

*PMIx v4.0*

▼ `PMIX_PROC_RELEASE` (m) C

25 **PMIX\_PROC\_RELEASE** (m)

▲ C

26 **IN** m

27 Pointer to a `pmix_proc_t` structure (handle)

1 **Free a proc array**  
2 Release an array of `pmix_proc_t` structures.

*PMIx v1.0*

▼ `PMIX_PROC_FREE` C

3 **PMIX\_PROC\_FREE**(*m*, *n*)

▲ `PMIX_PROC_FREE` C

4 **IN** *m*  
5 Pointer to the array of `pmix_proc_t` structures (handle)  
6 **IN** *n*  
7 Number of structures in the array (`size_t`)

8 **Load a proc structure**  
9 Load values into a `pmix_proc_t`.

*PMIx v2.0*

▼ `PMIX_PROC_LOAD` C

10 **PMIX\_PROC\_LOAD**(*m*, *n*, *r*)

▲ `PMIX_PROC_LOAD` C

11 **IN** *m*  
12 Pointer to the structure to be loaded (pointer to `pmix_proc_t`)  
13 **IN** *n*  
14 Namespace to be loaded (`pmix_namespace_t`)  
15 **IN** *r*  
16 Rank to be assigned (`pmix_rank_t`)  
17 No return value. Deprecated in favor of `PMIX_LOAD_PROCID`

18 **Compare identifiers**  
19 Compare two `pmix_proc_t` identifiers.

*PMIx v3.0*

▼ `PMIX_CHECK_PROCID` C

20 **PMIX\_CHECK\_PROCID**(*a*, *b*)

▲ `PMIX_CHECK_PROCID` C

21 **IN** *a*  
22 Pointer to a structure whose ID is to be compared (pointer to `pmix_proc_t`)  
23 **IN** *b*  
24 Pointer to a structure whose ID is to be compared (pointer to `pmix_proc_t`)

25 Returns `true` if the two structures contain matching namespaces and:

- 26 • the ranks are the same value
- 27 • one of the ranks is `PMIX_RANK_WILDCARD`

1 **Check if a process identifier is valid**

2 Check for invalid namespace or rank value

▼ **C** ▼

3 **PMIX\_PROCID\_INVALID (a)**

▲ **C** ▲

4 **IN a**

5 Pointer to a structure whose ID is to be checked (pointer to **pmix\_proc\_t**)

6 Returns **true** if the process identifier contains either an empty (i.e., invalid) *n*space field or a *rank*  
7 field of **PMIX\_RANK\_INVALID**

8 **Load a proclD structure**

9 Load values into a **pmix\_proc\_t**.

*PMIx v4.0*

▼ **C** ▼

10 **PMIX\_LOAD\_PROCID (m, n, r)**

▲ **C** ▲

11 **IN m**

12 Pointer to the structure to be loaded (pointer to **pmix\_proc\_t**)

13 **IN n**

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

15 **IN r**

16 Rank to be assigned (**pmix\_rank\_t**)

17 **Transfer a proclD structure**

18 Transfer contents of one **pmix\_proc\_t** value to another **pmix\_proc\_t**.

**Provisional**  
**v4.1**

▼ **C** ▼

19 **PMIX\_PROCID\_XFER (d, s)**

▲ **C** ▲

20 **IN d**

21 Pointer to the target structure (pointer to **pmix\_proc\_t**)

22 **IN s**

23 Pointer to the source structure (pointer to **pmix\_proc\_t**)

## Construct a multi-cluster namespace

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

▼ C ▼

**PMIX\_MULTICLUSTER\_NAMESPACE\_CONSTRUCT**(m, n, r)

▲ C ▲

**IN** m

`pmix_namespace_t` structure that will contain the multi-cluster identifier (`pmix_namespace_t`)

**IN** n

Cluster identifier (`char*`)

**IN** n

Namespace to be loaded (`pmix_namespace_t`)

Combined length of the cluster identifier and namespace must be less than `PMIX_MAX_NSLEN-2`.

## Parse a multi-cluster namespace

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

▼ C ▼

*PMIx v4.0*

**PMIX\_MULTICLUSTER\_NAMESPACE\_PARSE**(m, n, r)

▲ C ▲

**IN** m

`pmix_namespace_t` structure containing the multi-cluster identifier (pointer to `pmix_namespace_t`)

**IN** n

Location where the cluster ID is to be stored (`pmix_namespace_t`)

**IN** n

Location where the namespace is to be stored (`pmix_namespace_t`)

## 3.2.5 Process State Structure

*PMIx v2.0*

The `pmix_proc_state_t` structure is a `uint8_t` type for process state values. The following constants can be used to set a variable of the type `pmix_proc_state_t`.

### Advice to users

The fine-grained nature of the following constants may exceed the ability of an RM to provide updated process state values during the process lifetime. This is particularly true of states for 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 its name, location, and state.

*PMIx v2.0*

```

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*

```

19 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*

```

24 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.

▼ C ————— ▼

3 **PMIX\_PROC\_INFO\_CREATE (m, n)**

▲ C ————— ▲

4 **INOUT m**

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

7 **IN n**

Number of structures to be allocated (`size_t`)

9 **Free a process information structure**

10 Release a `pmix_proc_info_t` structure.

*PMIx v2.0*

▼ C ————— ▼

11 **PMIX\_PROC\_INFO\_RELEASE (m)**

▲ C ————— ▲

12 **IN m**

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.

*PMIx v2.0*

▼ C ————— ▼

16 **PMIX\_PROC\_INFO\_FREE (m, n)**

▲ C ————— ▲

17 **IN m**

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

19 **IN n**

Number of structures in the array (`size_t`)

## 21 3.2.7 Job State Structure

*PMIx v4.0*

22 The `pmix_job_state_t` structure is a `uint8_t` type for job state values. The following  
23 constants can be used to set a variable of the type `pmix_job_state_t`.

▼ Advice to users ————— ▼

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



---

1 **PMIX\_JOB\_STATE\_UNDEF** Undefined job state.  
2 **PMIX\_JOB\_STATE\_AWAITING\_ALLOC** Job is waiting for resources to be allocated to it.  
3 **PMIX\_JOB\_STATE\_LAUNCH\_UNDERWAY** Job launch is underway.  
4 **PMIX\_JOB\_STATE\_RUNNING** All processes in the job have been spawned and are executing.  
5 **PMIX\_JOB\_STATE\_SUSPENDED** All processes in the job have been suspended.  
6 **PMIX\_JOB\_STATE\_CONNECTED** All processes in the job have connected to their PMIx  
7 server.  
8 **PMIX\_JOB\_STATE\_UNTERMINATED** Define a “boundary” between the terminated states  
9 and **PMIX\_JOB\_STATE\_TERMINATED** so users can easily and quickly determine if a job is  
10 still running or not. Any value less than this constant means that the job has not terminated.  
11 **PMIX\_JOB\_STATE\_TERMINATED** All processes in the job have terminated and are no  
12 longer running - typically will be accompanied by the job exit status in response to a query.  
13 **PMIX\_JOB\_STATE\_TERMINATED\_WITH\_ERROR** Define a boundary so users can easily  
14 and quickly determine if a job abnormally terminated - typically will be accompanied by a  
15 job-related error code in response to a query Any value above this constant means that the job  
16 terminated abnormally.

### 17 3.2.8 Value Structure

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

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

*PMIx v1.0*

C

```

24 typedef struct pmix_value {
25     pmix_data_type_t type;
26     union {
27         bool flag;
28         uint8_t byte;
29         char *string;
30         size_t size;
31         pid_t pid;
32         int integer;
33         int8_t int8;
34         int16_t int16;
35         int32_t int32;
36         int64_t int64;
37         unsigned int uint;

```

```

1      uint8_t uint8;
2      uint16_t uint16;
3      uint32_t uint32;
4      uint64_t uint64;
5      float fval;
6      double dval;
7      struct timeval tv;
8      time_t time; // version 2.0
9      pmix_status_t status; // version 2.0
10     pmix_rank_t rank; // version 2.0
11     pmix_proc_t *proc; // version 2.0
12     pmix_byte_object_t bo;
13     pmix_persistence_t persist; // version 2.0
14     pmix_scope_t scope; // version 2.0
15     pmix_data_range_t range; // version 2.0
16     pmix_proc_state_t state; // version 2.0
17     pmix_proc_info_t *pinfo; // version 2.0
18     pmix_data_array_t *darray; // version 2.0
19     void *ptr; // version 2.0
20     pmix_alloc_directive_t adir; // version 2.0
21     } data;
22 } pmix_value_t;

```

### 23 3.2.8.1 Value structure support macros

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

#### 25 Initialize the value structure

26 Initialize the `pmix_value_t` fields.

*PMIx v1.0*

27 **PMIX\_VALUE\_CONSTRUCT** (m)

28 **IN** m

29 Pointer to the structure to be initialized (pointer to `pmix_value_t`)

#### 30 Destruct the value structure

31 Destruct the `pmix_value_t` fields.

*PMIx v1.0*

32 **PMIX\_VALUE\_DESTRUCT** (m)

33 **IN** m

34 Pointer to the structure to be destructed (pointer to `pmix_value_t`)

1 **Create a value array**  
2 Allocate and initialize an array of `pmix_value_t` structures.

*PMIx v1.0*



3 **PMIX\_VALUE\_CREATE (m, n)**



4 **INOUT m**

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

6 **IN n**

Number of structures to be allocated (`size_t`)

8 **Free a value structure**

Release a `pmix_value_t` structure.

*PMIx v4.0*



10 **PMIX\_VALUE\_RELEASE (m)**



11 **IN m**

Pointer to a `pmix_value_t` structure (handle)

13 **Free a value array**

Release an array of `pmix_value_t` structures.

*PMIx v1.0*



15 **PMIX\_VALUE\_FREE (m, n)**



16 **IN m**

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

18 **IN n**

Number of structures in the array (`size_t`)

20 **Load a value structure**

Load data into a `pmix_value_t` structure.

*PMIx v2.0*

▼ C ▼

1 **PMIX\_VALUE\_LOAD**(*v*, *d*, *t*);

▲ C ▲

2 **IN** *v*  
3 The **pmix\_value\_t** into which the data is to be loaded (pointer to **pmix\_value\_t**)  
4 **IN** *d*  
5 Pointer to the data value to be loaded (handle)  
6 **IN** *t*  
7 Type of the provided data value (**pmix\_data\_type\_t**)

8 This macro simplifies the loading of data into a **pmix\_value\_t** by correctly assigning values to  
9 the structure's fields.

▼ Advice to users ▼

10 The data will be copied into the **pmix\_value\_t** - thus, any data stored in the source value can be  
11 modified or free'd without affecting the copied data once the macro has completed.

▲ ▲

### Unload a value structure

12 Unload data from a **pmix\_value\_t** structure.

PMIx v2.2

▼ C ▼

14 **PMIX\_VALUE\_UNLOAD**(*r*, *v*, *d*, *t*);

▲ C ▲

15 **OUT** *r*  
16 Status code indicating result of the operation **pmix\_status\_t**  
17 **IN** *v*  
18 The **pmix\_value\_t** from which the data is to be unloaded (pointer to **pmix\_value\_t**)  
19 **INOUT** *d*  
20 Pointer to the location where the data value is to be returned (handle)  
21 **INOUT** *t*  
22 Pointer to return the data type of the unloaded value (handle)

23 This macro simplifies the unloading of data from a **pmix\_value\_t**.

▼ Advice to users ▼

24 Memory will be allocated and the data will be in the **pmix\_value\_t** returned - the source  
25 **pmix\_value\_t** will not be altered.

▲ ▲

1 **Transfer data between value structures**

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

▼ `C` ▼

3 `PMIX_VALUE_XFER(r, d, s);`

▲ `C` ▲

4 **OUT** `r`

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

6 **IN** `d`

7 Pointer to the `pmix_value_t` destination (handle)

8 **IN** `s`

9 Pointer to the `pmix_value_t` source (handle)

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

▼ **Advice to users** ▼

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

▲

14 **Retrieve a numerical value from a value struct**

15 Retrieve a numerical value from a `pmix_value_t` structure.

*PMIx v3.0*

▼ `C` ▼

16 `PMIX_VALUE_GET_NUMBER(s, m, n, t)`

▲ `C` ▲

17 **OUT** `s`

18 Status code for the request (`pmix_status_t`)

19 **IN** `m`

20 Pointer to the `pmix_value_t` structure (handle)

21 **OUT** `n`

22 Variable to be set to the value (match expected type)

23 **IN** `t`

24 Type of number expected in `m` (`pmix_data_type_t`)

25 Sets the provided variable equal to the numerical value contained in the given `pmix_value_t`,  
26 returning success if the data type of the value matches the expected type and

27 `PMIX_ERR_BAD_PARAM` if it doesn't

## 1 3.2.9 Info Structure

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

```
4 typedef struct pmix_info_t {  
5     pmix_key_t key;  
6     pmix_info_directives_t flags; // version 2.0  
7     pmix_value_t value;  
8 } pmix_info_t;
```

### 9 3.2.9.1 Info structure support macros

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

#### 11 Initialize the info structure

12 Initialize the `pmix_info_t` fields.

*PMIx v1.0*

```
13 PMIX_INFO_CONSTRUCT (m)
```

14 **IN** m

15 Pointer to the structure to be initialized (pointer to `pmix_info_t`)

#### 16 Destruct the info structure

17 Destruct the `pmix_info_t` fields.

*PMIx v1.0*

```
18 PMIX_INFO_DESTRUCT (m)
```

19 **IN** m

20 Pointer to the structure to be destructed (pointer to `pmix_info_t`)

#### 21 Create an info array

22 Allocate and initialize an array of info structures.

*PMIx v1.0*

```
23 PMIX_INFO_CREATE (m, n)
```

24 **INOUT** m

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

26 **IN** n

27 Number of structures to be allocated (`size_t`)

1 **Free an info array**  
2 Release an array of `pmix_info_t` structures.

*PMIx v1.0*

▼  C 

3 **PMIX\_INFO\_FREE**(`m`, `n`)

▲  C 

4 **IN** `m`  
5 Pointer to the array of `pmix_info_t` structures (handle)  
6 **IN** `n`  
7 Number of structures in the array (`size_t`)

8 **Load key and value data into a info struct**

*PMIx v1.0*

▼  C 

9 **PMIX\_INFO\_LOAD**(`v`, `k`, `d`, `t`);

▲  C 

10 **IN** `v`  
11 Pointer to the `pmix_info_t` into which the key and data are to be loaded (pointer to  
12 `pmix_info_t`)  
13 **IN** `k`  
14 String key to be loaded - must be less than or equal to `PMIX_MAX_KEYLEN` in length  
15 (handle)  
16 **IN** `d`  
17 Pointer to the data value to be loaded (handle)  
18 **IN** `t`  
19 Type of the provided data value (`pmix_data_type_t`)

20 This macro simplifies the loading of key and data into a `pmix_info_t` by correctly assigning  
21 values to the structure's fields.

▼  **Advice to users** 

22 Both key and data will be copied into the `pmix_info_t` - thus, the key and any data stored in the  
23 source value can be modified or free'd without affecting the copied data once the macro has  
24 completed.

▲ 





1 **Start a list of `pmix_info_t` structures**

2 Initialize a list of `pmix_info_t` structures. The actual list is opaque to the caller and is  
3 implementation-dependent.

▼ `PMIX_INFO_LIST_START` C

4 **`PMIX_INFO_LIST_START` (m)**

▲ `PMIX_INFO_LIST_START` C

5 **IN m**

6 A `void*` pointer (handle)

7 Note that the pointer will be initialized to an opaque structure whose elements are  
8 implementation-dependent. The caller must not modify or dereference the object.

9 **Add a `pmix_info_t` structure to a list**

10 Add a `pmix_info_t` structure containing the specified value to the provided list.

*PMIx v4.0*

▼ `PMIX_INFO_LIST_ADD` C

11 **`PMIX_INFO_LIST_ADD` (rc, m, k, d, t)**

▲ `PMIX_INFO_LIST_ADD` C

12 **INOUT rc**

13 Return status for the operation (`pmix_status_t`)

14 **IN m**

15 A `void*` pointer initialized via `PMIX_INFO_LIST_START` (handle)

16 **IN k**

17 String key to be loaded - must be less than or equal to `PMIX_MAX_KEYLEN` in length  
18 (handle)

19 **IN d**

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

21 **IN t**

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

▼ **Advice to users** ▼

23 Both key and data will be copied into the `pmix_info_t` on the list - thus, the key and any data  
24 stored in the source value can be modified or free'd without affecting the copied data once the  
25 macro has completed.

▲

1 **Transfer a `pmix_info_t` structure to a list**

2 Transfer the information in a `pmix_info_t` structure to the provided list.

▼ `PMIX_INFO_LIST_XFER(rc, m, s)` C

3 **`PMIX_INFO_LIST_XFER(rc, m, s)`**

▲ `PMIX_INFO_LIST_XFER(rc, m, s)` C

4 **INOUT `rc`**

5 Return status for the operation (`pmix_status_t`)

6 **IN `m`**

7 A `void*` pointer initialized via `PMIX_INFO_LIST_START` (handle)

8 **IN `s`**

9 Pointer to the source `pmix_info_t` (pointer to `pmix_info_t`)

▼ **Advice to users** ▼

10 All data (including key, value, and directives) will be copied into the destination `pmix_info_t`  
11 on the list - thus, the source `pmix_info_t` may be free'd without affecting the copied data once  
12 the macro has completed.

▲

13 **Convert a `pmix_info_t` list to an array**

14 Transfer the information in the provided `pmix_info_t` list to a `pmix_data_array_t` array

*PMIx v4.0*

▼ `PMIX_INFO_LIST_CONVERT(rc, m, d)` C

15 **`PMIX_INFO_LIST_CONVERT(rc, m, d)`**

▲ `PMIX_INFO_LIST_CONVERT(rc, m, d)` C

16 **INOUT `rc`**

17 Return status for the operation (`pmix_status_t`)

18 **IN `m`**

19 A `void*` pointer initialized via `PMIX_INFO_LIST_START` (handle)

20 **IN `d`**

21 Pointer to an instantiated `pmix_data_array_t` structure where the `pmix_info_t` array  
22 is to be stored (pointer to `pmix_data_array_t`)

23 **Release a `pmix_info_t` list**

24 Release the provided `pmix_info_t` list

*PMIx v4.0*

▼ `PMIX_INFO_LIST_RELEASE(m)` C

25 **`PMIX_INFO_LIST_RELEASE(m)`**

▲ `PMIX_INFO_LIST_RELEASE(m)` C

26 **IN `m`**

27 A `void*` pointer initialized via `PMIX_INFO_LIST_START` (handle)

28 Information contained in the `pmix_info_t` on the list shall be released in addition to whatever  
29 backing storage the implementation may have allocated to support construction of the list.

## 1 3.2.10 Info Type Directives

2 *PMIx v2.0*

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

### ▼ Advice to users ▼

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

### ▼ Advice to PMIx library implementers ▼

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

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

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

21 **PMIX\_INFO\_REQD\_PROCESSED** Mark that this required attribute has been processed. A  
22 required attribute can be handled at any level - the PMIx client library might take care of it, or  
23 it may be resolved by the PMIx server library, or it may pass up to the host environment for  
24 handling. If a level does not recognize or support the required attribute, it is required to pass it  
25 upwards to give the next level an opportunity to process it. Thus, the host environment (or the  
26 server library if the host does not support the given operation) must know if a lower level has  
27 handled the requirement so it can return a `PMIX_ERR_NOT_SUPPORTED` error status if the  
28 host itself cannot meet the request. Upon processing the request, the level must therefore mark  
29 the attribute with this directive to alert any subsequent levels that the requirement has been  
30 met.

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

33 **PMIX\_INFO\_DIR\_RESERVED** A bit-mask identifying the bits reserved for internal use by  
34 implementers - these currently are set as `0xffff0000`.

## Advice to PMIx server hosts

Host environments are advised to use the provided `PMIX_INFO_IS_REQUIRED` 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.

### 3.2.10.1 Info Directive support macros

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

#### Mark an info structure as required

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

PMIx v2.0

```
PMIX_INFO_REQUIRED (info);
```

**IN** `info`

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

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

#### Mark an info structure as optional

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

PMIx v2.0

```
PMIX_INFO_OPTIONAL (info);
```

**IN** `info`

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

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

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

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

PMIx v2.0

```
PMIX_INFO_IS_REQUIRED (info);
```

**IN** `info`

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

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

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

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

▼ `C` \_\_\_\_\_ ▼

`PMIX_INFO_IS_OPTIONAL(info);`

▲ \_\_\_\_\_ `C` \_\_\_\_\_ ▲

**IN** `info`

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

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

### Mark a required attribute as processed

Mark that a required `pmix_info_t` structure has been processed.

*PMIx v4.0*

▼ `C` \_\_\_\_\_ ▼

`PMIX_INFO_PROCESSED(info);`

▲ \_\_\_\_\_ `C` \_\_\_\_\_ ▲

**IN** `info`

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

Set the `PMIX_INFO_REQD_PROCESSED` flag in a `pmix_info_t` structure indicating that it has been processed.

### Test if a required attribute has been processed

Test that a required `pmix_info_t` structure has been processed.

*PMIx v4.0*

▼ `C` \_\_\_\_\_ ▼

`PMIX_INFO_WAS_PROCESSED(info);`

▲ \_\_\_\_\_ `C` \_\_\_\_\_ ▲

**IN** `info`

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

Test the `PMIX_INFO_REQD_PROCESSED` flag in a `pmix_info_t` structure.

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

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.

*PMIx v2.2*

▼ `C` \_\_\_\_\_ ▼

`PMIX_INFO_IS_END(info);`

▲ \_\_\_\_\_ `C` \_\_\_\_\_ ▲

**IN** `info`

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

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

## 1 3.2.11 Environmental Variable Structure

2 *PMIx v3.0*

3 Define a structure for specifying environment variable modifications. Standard environment  
4 variables (e.g., **PATH**, **LD\_LIBRARY\_PATH**, and **LD\_PRELOAD**) take multiple arguments  
5 separated by delimiters. Unfortunately, the delimiters depend upon the variable itself - some use  
6 semi-colons, some colons, etc. Thus, the operation requires not only the name of the variable to be  
7 modified and the value to be inserted, but also the separator to be used when composing the  
aggregate value.

```
8 typedef struct {  
9     char *envar;  
10    char *value;  
11    char separator;  
12 } pmix_envar_t;
```

### 13 3.2.11.1 Environmental variable support macros

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

#### 15 Initialize the envar structure

16 Initialize the `pmix_envar_t` fields.

17 *PMIx v3.0*

```
17 PMIX_ENVAR_CONSTRUCT (m)
```

18 **IN** m

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

#### 20 Destruct the envar structure

21 Clear the `pmix_envar_t` fields.

22 *PMIx v3.0*

```
22 PMIX_ENVAR_DESTRUCT (m)
```

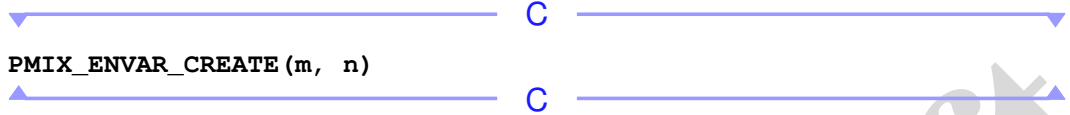
23 **IN** m

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

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

### Create an envar array

Allocate and initialize an array of `pmix_envvar_t` structures.



#### INOUT m

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

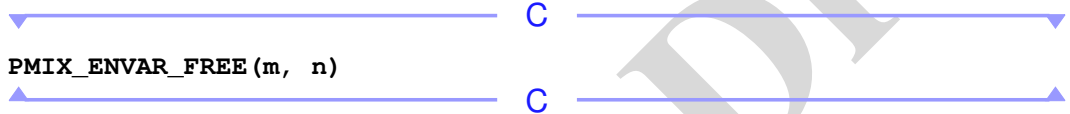
#### IN n

Number of structures to be allocated (`size_t`)

### Free an envar array

Release an array of `pmix_envvar_t` structures.

*PMIx v3.0*



#### IN m

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

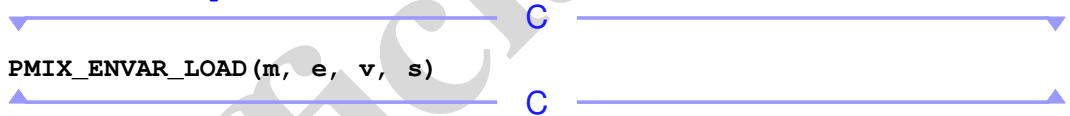
#### IN n

Number of structures in the array (`size_t`)

### Load an envar structure

Load values into a `pmix_envvar_t`.

*PMIx v2.0*



#### IN m

Pointer to the structure to be loaded (pointer to `pmix_envvar_t`)

#### IN e

Environmental variable name (`char*`)

#### IN v

Value of variable (`char*`)

#### IN v

Separator character (`char`)

## 1 3.2.12 Byte Object Type

2 The `pmix_byte_object_t` structure describes a raw byte sequence.

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

### 7 3.2.12.1 Byte object support macros

8 The following macros support the `pmix_byte_object_t` structure.

#### 9 Initialize the byte object structure

10 Initialize the `pmix_byte_object_t` fields.

*PMIx v2.0*

```
11 #define PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

12 **IN** `m`

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

#### 14 Destruct the byte object structure

15 Clear the `pmix_byte_object_t` fields.

*PMIx v2.0*

```
16 #define PMIX_BYTE_OBJECT_DESTRUCT(m)
```

17 **IN** `m`

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

#### 19 Create a byte object structure

20 Allocate and initialize an array of `pmix_byte_object_t` structures.

*PMIx v2.0*

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

22 **INOUT** `m`

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

25 **IN** `n`

26 Number of structures to be allocated (`size_t`)



1 **Free a byte object array**  
2 Release an array of `pmix_byte_object_t` structures.

▼ `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`.

*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.

*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.

▼ `PMIX_DATA_ARRAY_CONSTRUCT` C

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

▲ `PMIX_DATA_ARRAY_CONSTRUCT` C

4 **IN** m

5 Pointer to the structure to be initialized (pointer to `pmix_data_array_t`)

6 **IN** n

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

8 **IN** t

9 PMIx data type of the array elements (`pmix_data_type_t`)

10 **Destruct a data array structure**

11 Destruct the `pmix_data_array_t`, releasing the memory in the array.

*PMIx v2.2* ▼ `PMIX_DATA_ARRAY_DESTRUCT` C

12 **PMIX\_DATA\_ARRAY\_DESTRUCT**(m)

▲ `PMIX_DATA_ARRAY_DESTRUCT` C

13 **IN** m

14 Pointer to the structure to be destructed (pointer to `pmix_data_array_t`)

15 **Create a data array structure**

16 Allocate memory for the `pmix_data_array_t` object itself, and then allocate memory for the  
17 array of the indicated type.

*PMIx v2.2* ▼ `PMIX_DATA_ARRAY_CREATE` C

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

▲ `PMIX_DATA_ARRAY_CREATE` 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`)

25 **Free a data array structure**

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

*PMIx v2.2* ▼ `PMIX_DATA_ARRAY_FREE` C

27 **PMIX\_DATA\_ARRAY\_FREE**(m)

▲ `PMIX_DATA_ARRAY_FREE` C

28 **IN** m

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

## 1 3.2.14 Argument Array Macros

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

### 4 Argument array extension

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

▼ **C** ▲

```
6 PMIX_ARGV_APPEND(r, a, b);
```

▲ **C** ▼

7 **OUT** r

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

9 **INOUT** a

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

11 **IN** b

12 Argument to append to the list (string)

13 This function helps the caller build the **argv** portion of **pmix\_app\_t** structure, arrays of keys for  
14 querying, or other places where argv-style string arrays are required.

▼ **Advice to users** ▲

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

▲

### 17 Argument array prepend

18 Prepend a string to a NULL-terminated, argv-style array of strings.

▼ **C** ▲

```
19 PMIX_ARGV_PREPEND(r, a, b);
```

▲ **C** ▼

20 **OUT** r

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

22 **INOUT** a

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

24 **IN** b

25 Argument to append to the list (string)

26 This function helps the caller build the **argv** portion of **pmix\_app\_t** structure, arrays of keys for  
27 querying, or other places where argv-style string arrays are required.

## Advice to users

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

### Argument array extension - unique

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

```
PMIX_ARGV_APPEND_UNIQUE (r, a, b);
```

**OUT** r

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

**INOUT** a

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

**IN** b

Argument to append to the list (string)

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

## Advice to users

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

### Argument array release

Free an argv-style array and all of the strings that it contains.

```
PMIX_ARGV_FREE (a);
```

**IN** a

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

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

## Argument array split

Split a string into a NULL-terminated argv array.

▼ C ▼

`PMIX_ARGV_SPLIT(a, b, c);`

▲ C ▲

**OUT a**

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

**IN b**

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

**IN c**

Delimiter character (**char**)

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

▼ **Advice to users** ▼

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

▲

## Argument array join

Join all the elements of an argv array into a single newly-allocated string.

▼ C ▼

`PMIX_ARGV_JOIN(a, b, c);`

▲ C ▲

**OUT a**

Resulting string (**char\***)

**IN b**

Argv-style array to be joined (**char\*\***)

**IN c**

Delimiter character (**char**)

Join all the elements of an argv array into a single newly-allocated string.

1 **Argument array count**  
2 Return the length of a NULL-terminated argv array.



3 **PMIX\_ARGV\_COUNT**(r, a);



4 **OUT** r  
5 Number of strings in the array (integer)

6 **IN** a  
7 Argv-style array (**char\*\***)

8 Count the number of elements in an argv array

9 **Argument array copy**  
10 Copy an argv array, including copying all of its strings.



11 **PMIX\_ARGV\_COPY**(a, b);



12 **OUT** a  
13 New argv-style array (**char\*\***)

14 **IN** b  
15 Argv-style array (**char\*\***)

16 Copy an argv array, including copying all of its strings.

### 17 3.2.15 Set Environment Variable

18 **Summary**  
19 Set an environment variable in a **NULL**-terminated, env-style array.



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



21 **OUT** r  
22 Status code indicating success or failure of the operation (**pmix\_status\_t**)

23 **IN** name  
24 Argument name (string)

25 **IN** value  
26 Argument value (string)

27 **INOUT** env  
28 Environment array to update (pointer to array of strings)

## Description

Similar to `setenv` from the C API, this allows the caller to set an environment variable in the specified `env` array, which could then be passed to the `pmix_app_t` structure or any other destination.

### Advice to users

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

## 3.3 Generalized Data Types Used for Packing/Unpacking

The `pmix_data_type_t` structure is a `uint16_t` type for identifying the data type for packing/unpacking purposes. New data type values introduced in this version of the Standard are shown in **magenta**.

### Advice to PMIx library implementers

The following constants can be used to set a variable of the type `pmix_data_type_t`. Data types 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. Additionally, a PMIx implementation may choose to add additional types.

<code>PMIX_UNDEF</code>	Undefined.
<code>PMIX_BOOL</code>	Boolean (converted to/from native <code>true/false</code> ) ( <code>bool</code> ).
<code>PMIX_BYTE</code>	A byte of data ( <code>uint8_t</code> ).
<code>PMIX_STRING</code>	<code>NULL</code> terminated string ( <code>char*</code> ).
<code>PMIX_SIZE</code>	Size <code>size_t</code> .
<code>PMIX_PID</code>	Operating Process Identifier (PID) ( <code>pid_t</code> ).
<code>PMIX_INT</code>	Integer ( <code>int</code> ).
<code>PMIX_INT8</code>	8-byte integer ( <code>int8_t</code> ).
<code>PMIX_INT16</code>	16-byte integer ( <code>int16_t</code> ).
<code>PMIX_INT32</code>	32-byte integer ( <code>int32_t</code> ).
<code>PMIX_INT64</code>	64-byte integer ( <code>int64_t</code> ).
<code>PMIX_UINT</code>	Unsigned integer ( <code>unsigned int</code> ).
<code>PMIX_UINT8</code>	Unsigned 8-byte integer ( <code>uint8_t</code> ).
<code>PMIX_UINT16</code>	Unsigned 16-byte integer ( <code>uint16_t</code> ).
<code>PMIX_UINT32</code>	Unsigned 32-byte integer ( <code>uint32_t</code> ).
<code>PMIX_UINT64</code>	Unsigned 64-byte integer ( <code>uint64_t</code> ).
<code>PMIX_FLOAT</code>	Float ( <code>float</code> ).
<code>PMIX_DOUBLE</code>	Double ( <code>double</code> ).

1 **PMIX\_TIMEVAL** Time value (**struct timeval**).

2 **PMIX\_TIME** Time (**time\_t**).

3 **PMIX\_STATUS** Status code (**pmix\_status\_t**).

4 **PMIX\_VALUE** Value (**pmix\_value\_t**).

5 **PMIX\_PROC** Process (**pmix\_proc\_t**).

6 **PMIX\_APP** Application context.

7 **PMIX\_INFO** Info object.

8 **PMIX\_PDATA** Pointer to data.

9 **PMIX\_BUFFER** Buffer.

10 **PMIX\_BYTE\_OBJECT** Byte object (**pmix\_byte\_object\_t**).

11 **PMIX\_KVAL** Key/value pair.

12 **PMIX\_PERSIST** Persistence (**pmix\_persistence\_t**).

13 **PMIX\_POINTER** Pointer to an object (**void\***).

14 **PMIX\_SCOPE** Scope (**pmix\_scope\_t**).

15 **PMIX\_DATA\_RANGE** Range for data (**pmix\_data\_range\_t**).

16 **PMIX\_COMMAND** PMIx command code (used internally).

17 **PMIX\_INFO\_DIRECTIVES** Directives flag for **pmix\_info\_t**

18 (**pmix\_info\_directives\_t**).

19 **PMIX\_DATA\_TYPE** Data type code (**pmix\_data\_type\_t**).

20 **PMIX\_PROC\_STATE** Process state (**pmix\_proc\_state\_t**).

21 **PMIX\_PROC\_INFO** Process information (**pmix\_proc\_info\_t**).

22 **PMIX\_DATA\_ARRAY** Data array (**pmix\_data\_array\_t**).

23 **PMIX\_PROC\_RANK** Process rank (**pmix\_rank\_t**).

24 **PMIX\_PROC\_NAMESPACE** Process namespace (**pmix\_namespace\_t**). %

25 **PMIX\_QUERY** Query structure (**pmix\_query\_t**).

26 **PMIX\_COMPRESSED\_STRING** String compressed with zlib (**char\***).

27 **Provisional** **PMIX\_COMPRESSED\_BYTE\_OBJECT** Byte object whose bytes have been compressed with

28 zlib (**pmix\_byte\_object\_t**).

29 **PMIX\_ALLOC\_DIRECTIVE** Allocation directive (**pmix\_alloc\_directive\_t**).

30 **PMIX\_IOF\_CHANNEL** Input/output forwarding channel (**pmix\_iof\_channel\_t**).

31 **PMIX\_ENVAR** Environmental variable structure (**pmix\_envar\_t**).

32 **PMIX\_COORD** Structure containing fabric coordinates (**pmix\_coord\_t**).

33 **PMIX\_REGATTR** Structure supporting attribute registrations (**pmix\_regattr\_t**).

34 **PMIX\_REGEX** Regular expressions - can be a valid NULL-terminated string or an arbitrary

35 array of bytes.

36 **PMIX\_JOB\_STATE** Job state (**pmix\_job\_state\_t**).

37 **PMIX\_LINK\_STATE** Link state (**pmix\_link\_state\_t**).

38 **PMIX\_PROC\_CPuset** Structure containing the binding bitmap of a process

39 (**pmix\_cpuset\_t**).

40 **PMIX\_GEOMETRY** Geometry structure containing the fabric coordinates of a specified

41 device. (**pmix\_geometry\_t**).

42 **PMIX\_DEVICE\_DIST** Structure containing the minimum and maximum relative distance

43 from the caller to a given fabric device. (**pmix\_device\_distance\_t**).



1 **PMIX\_ENDPOINT** Structure containing an assigned endpoint for a given fabric device.  
 2 ([pmix\\_endpoint\\_t](#)).  
 3 **PMIX\_TOPO** Structure containing the topology for a given node. ([pmix\\_topology\\_t](#)).  
 4 **PMIX\_DEVTYPE** Bitmask containing the types of devices being referenced.  
 5 ([pmix\\_device\\_type\\_t](#)).  
 6 **PMIX\_LOCTYPE** Bitmask describing the relative location of another process.  
 7 ([pmix\\_locality\\_t](#)).  
 8 **PMIX\_DATA\_TYPE\_MAX** A starting point for implementer-specific data types. Values above  
 9 this are guaranteed not to conflict with PMIx values. Definitions should always be based on  
 10 the [PMIX\\_DATA\\_TYPE\\_MAX](#) constant and not a specific value as the value of the constant  
 11 may change.

## 12 3.4 General Callback Functions

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

### 16 3.4.1 Release Callback Function

#### 17 Summary

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

#### 21 Format

*PMIx v1.0*

C

```
22 typedef void (*pmix_release_cbfunc_t)
23         (void *cbdata);
```

C

#### 24 INOUT cbdata

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

#### 26 Description

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

## 1 3.4.2 Op Callback Function

### 2 Summary

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

```
4 typedef void (*pmix_op_cbfunc_t)
5     (pmix_status_t status, void *cbdata);
```

### 6 IN status

7 Status associated with the operation (handle)

### 8 IN cbdata

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

### 10 Description

11 Used by a wide range of PMIx API's including `PMIx_Fence_nb`,  
12 `pmix_server_client_connected2_fn_t`, `PMIx_server_register_nspace`. This  
13 callback function is used to return a status to an often nonblocking operation.

## 14 3.4.3 Value Callback Function

### 15 Summary

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

*PMIx v1.0*

```
17 typedef void (*pmix_value_cbfunc_t)
18     (pmix_status_t status,
19      pmix_value_t *kv, void *cbdata);
```

### 20 IN status

21 Status associated with the operation (handle)

### 22 IN kv

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

### 24 IN cbdata

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

### 26 Description

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

## 1 3.4.4 Info Callback Function

### 2 Summary

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

```
4 typedef void (*pmix_info_cbfunc_t)
5     (pmix_status_t status,
6      pmix_info_t info[], size_t ninfo,
7      void *cbdata,
8      pmix_release_cbfunc_t release_fn,
9      void *release_cbdata);
```

### 10 IN status

11 Status associated with the operation (`pmix_status_t`)

### 12 IN info

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

### 14 IN ninfo

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

### 16 IN cbdata

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

### 18 IN release\_fn

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

### 20 IN release\_cbdata

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

### 22 Description

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

## 25 3.4.5 Handler registration callback function

### 26 Summary

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

### 28 Format

*PMIx v3.0*

```
29 typedef void (*pmix_hdlr_reg_cbfunc_t)
30     (pmix_status_t status,
31      size_t refid,
32      void *cbdata);
```

1        **IN**    **status**  
2            **PMIX\_SUCCESS** or an appropriate error constant (**pmix\_status\_t**)  
3        **IN**    **refid**  
4            reference identifier assigned to the handler by PMIx, used to deregister the handler (**size\_t**)  
5        **IN**    **cbdata**  
6            object provided to the registration call (pointer)

7        **Description**  
8            Callback function for calls to register handlers, e.g., event notification and IOF requests.

### 9    3.5   PMIx Datatype Value String Representations

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

12        **Summary**  
13        String representation of a **pmix\_status\_t**.

14        *PMIx v1.0*    **const char\***  
15            **PMIx\_Error\_string(pmix\_status\_t status);**

16        **Summary**  
17        String representation of a **pmix\_proc\_state\_t**.

18        *PMIx v2.0*    **const char\***  
19            **PMIx\_Proc\_state\_string(pmix\_proc\_state\_t state);**

20        **Summary**  
21        String representation of a **pmix\_scope\_t**.

22        *PMIx v2.0*    **const char\***  
23            **PMIx\_Scope\_string(pmix\_scope\_t scope);**

1 **Summary**  
2 String representation of a `pmix_persistence_t`.

▼ `C` ▼

3 `const char*`  
4 `PMIx_Persistence_string(pmix_persistence_t persist);`

▲ `C` ▲

5 **Summary**  
6 String representation of a `pmix_data_range_t`.

*PMIx v2.0* ▼ `C` ▼

7 `const char*`  
8 `PMIx_Data_range_string(pmix_data_range_t range);`

▲ `C` ▲

9 **Summary**  
10 String representation of a `pmix_info_directives_t`.

*PMIx v2.0* ▼ `C` ▼

11 `const char*`  
12 `PMIx_Info_directives_string(pmix_info_directives_t directives);`

▲ `C` ▲

13 **Summary**  
14 String representation of a `pmix_data_type_t`.

*PMIx v2.0* ▼ `C` ▼

15 `const char*`  
16 `PMIx_Data_type_string(pmix_data_type_t type);`

▲ `C` ▲

17 **Summary**  
18 String representation of a `pmix_alloc_directive_t`.

*PMIx v2.0* ▼ `C` ▼

19 `const char*`  
20 `PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);`

▲ `C` ▲

1 **Summary**  
2 String representation of a `pmix_iof_channel_t`.

3 `const char*`  
4 `PMIx_IOF_channel_string(pmix_iof_channel_t channel);`

5 **Summary**  
6 String representation of a `pmix_job_state_t`.

*PMIx v4.0*  
7 `const char*`  
8 `PMIx_Job_state_string(pmix_job_state_t state);`

9 **Summary**  
10 String representation of a PMIx attribute.

*PMIx v4.0*  
11 `const char*`  
12 `PMIx_Get_attribute_string(char *attributename);`

13 **Summary**  
14 Return the PMIx attribute name corresponding to the given attribute string.

*PMIx v4.0*  
15 `const char*`  
16 `PMIx_Get_attribute_name(char *attributestring);`

17 **Summary**  
18 String representation of a `pmix_link_state_t`.

*PMIx v4.0*  
19 `const char*`  
20 `PMIx_Link_state_string(pmix_link_state_t state);`

1 **Summary**  
2 String representation of a `pmix_device_type_t`.

▼ C ▼

3 `const char*`  
4 `PMIx_Device_type_string(pmix_device_type_t type);`

▲ C ▲

Unofficial Draft

## CHAPTER 4

# Client Initialization and Finalization

---

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

4 There are three sets of initialization and finalization functions depending upon the role of the  
5 process in the PMIx Standard - those associated with the PMIx *client* are defined in this chapter.  
6 Similar functions corresponding to the roles of *server* and *tool* are defined in Chapters 17 and 18,  
7 respectively.

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

## 13 4.1 PMIx\_Initialized

### 14 Summary

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

### 17 Format

*PMIx v1.0*

18 **int PMIx\_Initialized(void)**

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

### Rationale

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

### 21 Description

22 Check to see if the PMIx library has been initialized using any of the init functions: **PMIx\_Init**,  
23 **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

*PMIx v1.0*

C

6 `const char* PMIx_Get_version(void)`

C

### 7 Description

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

## 10 4.3 PMIx\_Init

### 11 Summary

12 Initialize the PMIx client library

### 13 Format

*PMIx v1.2*

C

14 `pmix_status_t`

15 `PMIx_Init(pmix_proc_t *proc,`  
16 `pmix_info_t info[], size_t ninfo)`

C

### 17 INOUT `proc`

18 `proc` structure (handle)

### 19 IN `info`

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

### 21 IN `ninfo`

22 Number of elements in the `info` array (`size_t`)

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

## Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**PMIX\_EXTERNAL\_PROGRESS** "pmix.evext" (bool)

1 The host shall progress the PMIx library via calls to **PMIx\_Progress**

2 **PMIX\_EVENT\_BASE** "pmix.evbase" (void\*)

3 Pointer to an **event\_base** to use in place of the internal progress thread. All PMIx library  
4 events are to be assigned to the provided event base. The event base *must* be compatible with  
5 the event library used by the PMIx implementation - e.g., either both the host and PMIx  
6 library must use libevent, or both must use libev. Cross-matches are unlikely to work and  
7 should be avoided - it is the responsibility of the host to ensure that the PMIx  
8 implementation supports (and was built with) the appropriate event library.

9 If provided, the following attributes are used by the event notification system for inter-library  
10 coordination:

11 **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (char\*)

12 Programming model being initialized (e.g., "MPI" or "OpenMP").

13 **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (char\*)

14 Programming model implementation ID (e.g., "OpenMPI" or "MPICH").

15 **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (char\*)

16 Programming model version string (e.g., "2.1.1").

17 **PMIX\_THREADING\_MODEL** "pmix.threads" (char\*)

18 Threading model used (e.g., "pthreads").

19 **PMIX\_MODEL\_NUM\_THREADS** "pmix.mdl.nthrds" (uint64\_t)

20 Number of active threads being used by the model.

21 **PMIX\_MODEL\_NUM\_CPUS** "pmix.mdl.ncpu" (uint64\_t)

22 Number of cpus being used by the model.

23 **PMIX\_MODEL\_CPU\_TYPE** "pmix.mdl.cputype" (char\*)

24 Granularity - "hwthread", "core", etc.

25 **PMIX\_MODEL\_AFFINITY\_POLICY** "pmix.mdl.tap" (char\*)

26 Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread  
27 located on cpu close to master thread), "spread" (threads load-balanced across available  
28 cpus).

## Description

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

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

If successful, the function shall return `PMIX_SUCCESS` and fill the `proc` structure (if provided) with the server-assigned namespace and rank of the process within the application. In addition, all startup information provided by the resource manager shall be made available to the client process via subsequent calls to `PMIx_Get`.

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

Each call to `PMIx_Init` may contain an array of `pmix_info_t` structures passing directives to the PMIx client library as per the above attributes.

Multiple calls to `PMIx_Init` shall not include conflicting directives. The `PMIx_Init` function will return an error when directives that conflict with prior directives are encountered.

### 4.3.1 Initialization events

The following events are typically associated with calls to `PMIx_Init`:

<code>PMIX_MODEL_DECLARED</code>	Model declared.
<code>PMIX_MODEL_RESOURCES</code>	Resource usage by a programming model has changed.
<code>PMIX_OPENMP_PARALLEL_ENTERED</code>	An OpenMP parallel code region has been entered.
<code>PMIX_OPENMP_PARALLEL_EXITED</code>	An OpenMP parallel code region has completed.

### 4.3.2 Initialization attributes

The following attributes influence the behavior of `PMIx_Init`.

#### 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.

<code>PMIX_TCP_REPORT_URI</code>	" <code>pmix.tcp.repuri</code> " ( <code>char*</code> )
	If provided, directs that the TCP URI be reported and indicates the desired method of reporting: <code>'-'</code> for stdout, <code>'+'</code> for stderr, or filename.

1 **PMIX\_TCP\_URI** "pmix.tcp.uri" (char\*)  
 2 The URI of the PMIX server to connect to, or a file name containing it in the form of  
 3 file:<name of file containing it>.

4 **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)  
 5 Comma-delimited list of devices and/or CIDR notation to include when establishing the  
 6 TCP connection.

7 **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)  
 8 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the  
 9 TCP connection.

10 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)  
 11 The IPv4 port to be used..

12 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)  
 13 The IPv6 port to be used.

14 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)  
 15 Set to **true** to disable IPv4 family of addresses.

16 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)  
 17 Set to **true** to disable IPv6 family of addresses.

#### 18 4.3.2.2 Programming model attributes

19 These attributes are associated with programming models.

20 **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (char\*)  
 21 Programming model being initialized (e.g., "MPI" or "OpenMP").

22 **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (char\*)  
 23 Programming model implementation ID (e.g., "OpenMPI" or "MPICH").

24 **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mdl.vrs" (char\*)  
 25 Programming model version string (e.g., "2.1.1").

26 **PMIX\_THREADING\_MODEL** "pmix.threads" (char\*)  
 27 Threading model used (e.g., "pthreads").

28 **PMIX\_MODEL\_NUM\_THREADS** "pmix.mdl.nthrds" (uint64\_t)  
 29 Number of active threads being used by the model.

30 **PMIX\_MODEL\_NUM\_CPUS** "pmix.mdl.ncpu" (uint64\_t)  
 31 Number of cpus being used by the model.

32 **PMIX\_MODEL\_CPU\_TYPE** "pmix.mdl.cputype" (char\*)  
 33 Granularity - "hwthread", "core", etc.

34 **PMIX\_MODEL\_PHASE\_NAME** "pmix.mdl.phase" (char\*)  
 35 User-assigned name for a phase in the application execution (e.g., "cfd reduction").

36 **PMIX\_MODEL\_PHASE\_TYPE** "pmix.mdl.ptype" (char\*)  
 37 Type of phase being executed (e.g., "matrix multiply").

38 **PMIX\_MODEL\_AFFINITY\_POLICY** "pmix.mdl.tap" (char\*)  
 39 Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread  
 40 located on cpu close to master thread), "spread" (threads load-balanced across available  
 41 cpus).

## 1 4.4 PMIx\_Finalize

### 2 Summary

3 Finalize the PMIx client library.

### 4 *PMIx v1.0* Format

C

5 `pmix_status_t`

6 `PMIx_Finalize(const pmix_info_t info[], size_t ninfo)`

C

7 **IN** `info`

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

9 **IN** `ninfo`

10 Number of elements in the `info` array (`size_t`)

11 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### Optional Attributes

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

13 `PMIX_EMBED_BARRIER` "`pmix.embed.barrier`" (`bool`)

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

15 `PMIx_Finalize` does not include an internal barrier operation by default. This attribute  
16 directs `PMIx_Finalize` to execute a barrier as part of the finalize operation.

### 17 Description

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

## 21 4.4.1 Finalize attributes

22 The following attribute influences the behavior of `PMIx_Finalize`.

23 `PMIX_EMBED_BARRIER` "`pmix.embed.barrier`" (`bool`)

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

25 `PMIx_Finalize` does not include an internal barrier operation by default. This attribute  
26 directs `PMIx_Finalize` to execute a barrier as part of the finalize operation.

## 27 4.5 PMIx\_Progress

### 28 Summary

29 Progress the PMIx library.

1  
2  
3  
4  
5  
6

**Format**

C

`void  
PMIx_Progress(void)`

C

**Description**

Progress the PMIx library. Note that special care must be taken to avoid deadlocking in PMIx callback functions and APIs.

Unofficial Draft

## CHAPTER 5

# Synchronization and Data Access Operations

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

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

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

## 5.1 PMIx\_Fence

### Summary

Execute a blocking barrier across the processes identified in the specified array, collecting information posted via `PMIx_Put` as directed.

### Format

*PMIx v1.0*

C

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

C

**IN** `procs`  
Array of `pmix_proc_t` 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)

Returns `PMIX_SUCCESS` or a negative value indicating the error.



## Required Attributes

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

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

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

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

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

## 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:

**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

Passing a **NULL** pointer as the *procs* parameter indicates that the fence is to span all processes in the client's namespace. Each provided `pmix_proc_t` struct can pass `PMIX_RANK_WILDCARD` to indicate that all processes in the given namespace are participating.

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

### Advice to PMIx library implementers

`PMIx_Fence` 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. 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.

## 5.2 PMIx\_Fence\_nb

### Summary

Execute a nonblocking `PMIx_Fence` across the processes identified in the specified array of processes, collecting information posted via `PMIx_Put` as directed.

## Format

C

```
pmix_status_t
PMIx_Fence_nb(const pmix_proc_t procs[], size_t nprocs,
              const pmix_info_t info[], size_t ninfo,
              pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

**IN** **procs**  
Array of `pmix_proc_t` 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 (function reference)

**IN** **cbdata**  
Data to be passed to the callback function (memory reference)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called. This can occur if the collective involved only processes on the local node.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Required Attributes

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

`PMIX_COLLECT_DATA` "pmix.collect" (bool)  
Collect all data posted by the participants using `PMIx_Put` that has been committed via `PMIx_Commit`, making the collection locally available to each participant at the end of the operation. By default, this will include all job-level information that was locally generated by PMIx servers unless excluded using the `PMIX_COLLECT_GENERATED_JOB_INFO` attribute.

`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.  
17

## 18 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 *PMIx v1.0* Format

C

5 `pmix_status_t`

```
6 PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,  
7          const pmix_info_t info[], size_t ninfo,  
8          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 A successful return indicates that the requested data has been returned in the manner requested  
20 (.e.g., in a provided static memory location ).

21 Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- 22 • `PMIX_ERR_BAD_PARAM` A bad parameter was passed to the function call - e.g., the request  
23 included the `PMIX_GET_STATIC_VALUES` directive, but the provided storage location was  
24 **NULL**
- 25 • `PMIX_ERR_EXISTS_OUTSIDE_SCOPE` The requested key exists, but was posted in a *scope*  
26 (see Section 7.1.1.1) that does not include the requester.
- 27 • `PMIX_ERR_NOT_FOUND` The requested data was not available.

28 If none of the above return codes are appropriate, then an implementation must return either a  
29 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Required Attributes

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

31 `PMIX_OPTIONAL` "pmix.optional" (**bool**)

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

1 **PMIX\_IMMEDIATE** "pmix.immediate" (bool)  
2 Specified operation should immediately return an error from the PMIx server if the requested  
3 data cannot be found - do not request it from the host RM.

4 **PMIX\_DATA\_SCOPE** "pmix.scope" (pmix\_scope\_t)  
5 Scope of the data to be searched in a **PMIx\_Get** call.

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

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

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

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

25 **PMIX\_GET\_STATIC\_VALUES** "pmix.get.static" (bool)  
26 Request that the data be returned in the provided storage location. The caller is responsible  
27 for destructing the **pmix\_value\_t** using the **PMIX\_VALUE\_DESTRUCT** macro when  
28 done.

29 **PMIX\_GET\_POINTER\_VALUES** "pmix.get.pntrs" (bool)  
30 Request that any pointers in the returned value point directly to values in the key-value store.  
31 The user *must not* release any returned data pointers.

32 **PMIX\_GET\_REFRESH\_CACHE** "pmix.get.refresh" (bool)  
33 When retrieving data for a remote process, refresh the existing local data cache for the  
34 process in case new values have been put and committed by the process since the last refresh.  
35 Local process information is assumed to be automatically updated upon posting by the  
36 process. A **NULL** key will cause all values associated with the process to be refreshed -  
37 otherwise, only the indicated key will be updated. A process rank of  
38 **PMIX\_RANK\_WILDCARD** can be used to update job-related information in dynamic

1 environments. The user is responsible for subsequently updating refreshed values they may  
2 have cached in their own local memory.

### Optional Attributes

3 The following attributes are optional for host environments:

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

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

### Description

8 Retrieve information for the specified *key* associated with the process identified in the given  
9 **pmix\_proc\_t**. See Chapters 6 and 7 for details on rules governing retrieval of information.

10 Information will be returned according to provided directives:

- 11 • In the absence of any directive, the returned **pmix\_value\_t** shall be an allocated memory  
12 object. The caller is responsible for releasing the object when done.
- 13 • If **PMIX\_GET\_POINTER\_VALUES** is given, then the function shall return a pointer to a  
14 **pmix\_value\_t** in the PMIx library's memory that contains the requested information.
- 15 • If **PMIX\_GET\_STATIC\_VALUES** is given, then the function shall return the information in the  
16 provided **pmix\_value\_t** pointer. In this case, the caller must provide storage for the structure  
17 and pass the pointer to that storage in the *val* parameter. If the implementation cannot return a  
18 static value, then the call to **PMIx\_Get** must return the **PMIX\_ERR\_NOT\_SUPPORTED** status.

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

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

## 21 5.3.1 PMIx\_Get\_nb

### 22 Summary

23 Nonblocking **PMIx\_Get** operation.  
24

## Format

C

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

C

- IN proc**  
Process identifier - a **NULL** value may be used in place of the caller's ID (handle)
- IN key**  
Key to retrieve (string)
- IN info**  
Array of info structures (array of handles)
- IN ninfo**  
Number of elements in the *info* array (integer)
- IN cbfunc**  
Callback function (function reference)
- IN cbdata**  
Data to be passed to the callback function (memory reference)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

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

- **PMIX\_SUCCESS** The requested data has been returned.
- **PMIX\_ERR\_EXISTS\_OUTSIDE\_SCOPE** The requested key exists, but was posted in a *scope* (see Section 7.1.1.1) that does not include the requester.
- **PMIX\_ERR\_NOT\_FOUND** The requested data was not available.
- 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:

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

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

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

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



1 **PMIX\_DATA\_SCOPE** "pmix.scope" (pmix\_scope\_t)  
 2 Scope of the data to be searched in a **PMIx\_Get** call.

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

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

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

22 **PMIX\_GET\_POINTER\_VALUES** "pmix.get.pntrs" (bool)  
 23 Request that any pointers in the returned value point directly to values in the key-value store.  
 24 The user *must not* release any returned data pointers.

25 **PMIX\_GET\_REFRESH\_CACHE** "pmix.get.refresh" (bool)  
 26 When retrieving data for a remote process, refresh the existing local data cache for the  
 27 process in case new values have been put and committed by the process since the last refresh.  
 28 Local process information is assumed to be automatically updated upon posting by the  
 29 process. A **NULL** key will cause all values associated with the process to be refreshed -  
 30 otherwise, only the indicated key will be updated. A process rank of  
 31 **PMIX\_RANK\_WILDCARD** can be used to update job-related information in dynamic  
 32 environments. The user is responsible for subsequently updating refreshed values they may  
 33 have cached in their own local memory.

---

34

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

36 **PMIX\_WAIT** "pmix.wait" (int)  
 37 Caller requests that the PMIx server wait until at least the specified number of values are  
 38 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:

**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

The callback function will be executed once the retrieval rules of Chapters 6 or 7 are met. See **PMIx\_Get** for a full description. Note that the non-blocking form of this function cannot support the **PMIX\_GET\_STATIC\_VALUES** attribute as the user cannot pass in the required pointer to storage for the result.

## 5.3.2 Retrieval attributes

The following attributes are defined for use by retrieval APIs:

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

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

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

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

**PMIX\_GET\_POINTER\_VALUES** "pmix.get.pntrs" (bool)

Request that any pointers in the returned value point directly to values in the key-value store. The user *must not* release any returned data pointers.

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

Request that the data be returned in the provided storage location. The caller is responsible for destructing the **pmix\_value\_t** using the **PMIX\_VALUE\_DESTRUCT** macro when done.

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

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

Local process information is assumed to be automatically updated upon posting by the process. A **NULL** key will cause all values associated with the process to be refreshed - otherwise, only the indicated key will be updated. A process rank of

**PMIX\_RANK\_WILDCARD** can be used to update job-related information in dynamic environments. The user is responsible for subsequently updating refreshed values they may have cached in their own local memory.

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

1 Scope of the data to be searched in a `PMIx_Get` call.

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\_WAIT** "`pmix.wait`" (`int`)

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

## 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 17.2.3.2).

## 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  
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 Maximum number of process that can be simultaneously executing in a session. Note that  
2 this attribute is equivalent to the **PMIX\_MAX\_PROCS** attribute for the *session* realm - it is  
3 included in the PMIx Standard for historical reasons.

4 **PMIX\_TMPDIR** "pmix.tmpdir" (char\*)

5 Full path to the top-level temporary directory assigned to the session.

6 **PMIX\_TDIR\_RMCLEAN** "pmix.tdir.rmclean" (bool)

7 Resource Manager will cleanup assigned temporary directory trees.

8 **PMIX\_HOSTNAME\_KEEP\_FQDN** "pmix.fqdn" (bool)

9 Fully Qualified Domain Names (FQDNs) are being retained by the PMIx library.

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

12 **PMIX\_RM\_NAME** "pmix.rm.name" (char\*)

13 String name of the RM.

14 **PMIX\_RM\_VERSION** "pmix.rm.version" (char\*)

15 RM version string.

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

18 **PMIX\_ALLOCATED\_NODELIST** "pmix.alist" (char\*)

19 Comma-delimited list or regular expression of all nodes in the specified realm regardless of  
20 whether or not they currently host processes. Defaults to the *job* realm.

21 **PMIX\_NUM\_ALLOCATED\_NODES** "pmix.num.anodes" (uint32\_t)

22 Number of nodes in the specified realm regardless of whether or not they currently host  
23 processes. Defaults to the *job* realm.

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

25 Maximum number of processes that can be executed in the specified realm. Typically, this is  
26 a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
27 description. Defaults to the *job* realm.

28 **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)

29 Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults  
30 to the *job* realm.

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

32 Maximum number of processes that can simultaneously be executing in the specified realm.  
33 Note that this attribute is the equivalent to **PMIX\_MAX\_PROCS** - it is included in the PMIx  
34 Standard for historical reasons. Defaults to the *job* realm.

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

36 Number of nodes currently hosting processes in the specified realm. Defaults to the *job*  
37 realm.

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

39 Regular expression of nodes currently hosting processes in the specified realm - see 17.2.3.2  
40 for an explanation of its generation. Defaults to the *job* realm.

1 **PMIX\_NODE\_MAP\_RAW** "pmix.nmap.raw" (char\*)  
 2 Comma-delimited list of nodes containing procs within the specified realm. Defaults to the  
 3 *job* realm.  
 4 **PMIX\_PROC\_MAP** "pmix.pmap" (char\*)  
 5 Regular expression describing processes on each node in the specified realm - see 17.2.3.2  
 6 for an explanation of its generation. Defaults to the *job* realm.  
 7 **PMIX\_PROC\_MAP\_RAW** "pmix.pmap.raw" (char\*)  
 8 Semi-colon delimited list of strings, each string containing a comma-delimited list of ranks  
 9 on the corresponding node within the specified realm. Defaults to the *job* realm.  
 10 **PMIX\_ANL\_MAP** "pmix.anlmap" (char\*)  
 11 Process map equivalent to **PMIX\_PROC\_MAP** expressed in Argonne National Laboratory's  
 12 PMI-1/PMI-2 notation. Defaults to the *job* realm.

## 13 6.1.2 Job realm attributes

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

21 Job-level information includes the following attributes:

22 **PMIX\_NAMESPACE** "pmix.namespace" (char\*)  
 23 Namespace of the job - may be a numerical value expressed as a string, but is often an  
 24 alphanumeric string carrying information solely of use to the system. Required to be unique  
 25 within the scope of the host environment.  
 26 **PMIX\_JOBID** "pmix.jobid" (char\*)  
 27 Job identifier assigned by the scheduler to the specified job - may be identical to the  
 28 namespace, but is often a numerical value expressed as a string (e.g., "12345.3").  
 29 **PMIX\_NPROC\_OFFSET** "pmix.offset" (pmix\_rank\_t)  
 30 Starting global rank of the specified job.  
 31 **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)  
 32 Maximum number of processes that can be executed in the specified realm. Typically, this is  
 33 a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
 34 description. Defaults to the *job* realm. In this context, this is the maximum number of  
 35 processes that can be simultaneously executed in the specified job, which may be a subset of  
 36 the number allocated to the overall session.  
 37 **PMIX\_NUM\_SLOTS** "pmix.num.slots" (uint32\_t)  
 38 Maximum number of processes that can simultaneously be executing in the specified realm.  
 39 Note that this attribute is the equivalent to **PMIX\_MAX\_PROCS** - it is included in the PMIx  
 40 Standard for historical reasons. Defaults to the *job* realm. In this context, this is the

1 maximum number of process that can be simultaneously executing within the specified job,  
2 which may be a subset of the number allocated to the overall session. Jobs may reserve a  
3 subset of their assigned maximum processes for dynamic operations such as [PMIx\\_Spawn](#).

4 **PMIX\_NUM\_NODES** "`pmix.num.nodes`" (`uint32_t`)

5 Number of nodes currently hosting processes in the specified realm. Defaults to the *job*  
6 realm. In this context, this is the number of nodes currently hosting processes in the  
7 specified job, which may be a subset of the nodes allocated to the overall session. Jobs may  
8 reserve a subset of their assigned nodes for dynamic operations such as [PMIx\\_Spawn](#) - i.e.,  
9 not all nodes may have executing processes from this job at a given point in time.

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

11 Regular expression of nodes currently hosting processes in the specified realm - see [17.2.3.2](#)  
12 for an explanation of its generation. Defaults to the *job* realm. In this context, this is the  
13 regular expression of nodes currently hosting processes in the specified job.

14 **PMIX\_NODE\_LIST** "`pmix.nlist`" (`char*`)

15 Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults  
16 to the *job* realm. In this context, this is the comma-delimited list of nodes currently hosting  
17 processes in the specified job.

18 **PMIX\_PROC\_MAP** "`pmix.pmap`" (`char*`)

19 Regular expression describing processes on each node in the specified realm - see [17.2.3.2](#)  
20 for an explanation of its generation. Defaults to the *job* realm. In this context, this is the  
21 regular expression describing processes on each node in the specified job.

22 **PMIX\_ANL\_MAP** "`pmix.anlmap`" (`char*`)

23 Process map equivalent to [PMIX\\_PROC\\_MAP](#) expressed in Argonne National Laboratory's  
24 PMI-1/PMI-2 notation. Defaults to the *job* realm. In this context, this is the process  
25 mapping in Argonne National Laboratory's PMI-1/PMI-2 notation of the processes in the  
26 specified job.

27 **PMIX\_CMD\_LINE** "`pmix.cmd.line`" (`char*`)

28 Command line used to execute the specified job (e.g., "mpirun -n 2 --map-by foo ./myapp : -n  
29 4 ./myapp2").

30 **PMIX\_NSDIR** "`pmix.nmdir`" (`char*`)

31 Full path to the temporary directory assigned to the specified job, under [PMIX\\_TMPDIR](#).

32 **PMIX\_JOB\_SIZE** "`pmix.job.size`" (`uint32_t`)

33 Total number of processes in the specified job across all contained applications. Note that  
34 this value can be different from [PMIX\\_MAX\\_PROCS](#). For example, users may choose to  
35 subdivide an allocation (running several jobs in parallel within it), and dynamic  
36 programming models may support adding and removing processes from a running *job*  
37 on-the-fly. In the latter case, PMIx events may be used to notify processes within the job that  
38 the job size has changed.

39 **PMIX\_JOB\_NUM\_APPS** "`pmix.job.napps`" (`uint32_t`)

40 Number of applications in the specified job.



## 6.1.3 Application realm attributes

Application-related information can only be retrieved by including the `PMIX_APP_INFO` attribute in the `info` array passed to `PMIx_Get`. If the `PMIX_APPNUM` qualifier is given, then the query shall return the corresponding value for the given application within the namespace specified in the `proc` argument of the query (a `NULL` value for the `proc` argument equates to the namespace of the caller). If the `PMIX_APPNUM` qualifier is not included, then the retrieval shall default to the application containing the specified process. If the rank of the specified process is `PMIX_RANK_WILDCARD`, then the application number shall default to that of the calling process if the namespace is its own job, or a value of zero if the namespace is that of a different job.

Application-level information includes the following attributes:

**PMIX\_APPNUM** "pmix.appnum" (`uint32_t`)

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

**PMIX\_NUM\_NODES** "pmix.num.nodes" (`uint32_t`)

Number of nodes currently hosting processes in the specified realm. Defaults to the `job` realm. In this context, this is the number of nodes currently hosting processes in the specified application, which may be a subset of the nodes allocated to the overall session.

**PMIX\_APPLDR** "pmix.aldr" (`pmix_rank_t`)

Lowest rank in the specified application.

**PMIX\_APP\_SIZE** "pmix.app.size" (`uint32_t`)

Number of processes in the specified application, regardless of their execution state - i.e., this number may include processes that either failed to start or have already terminated.

**PMIX\_APP\_ARGV** "pmix.app.argv" (`char*`)

Consolidated argv passed to the spawn command for the given application (e.g., `"/myapp arg1 arg2 arg3"`).

**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. Defaults to the `job` realm. In this context, this is the maximum number of processes that can be executed in the specified application, which may be a subset of the number allocated to the overall session and job.

**PMIX\_NUM\_SLOTS** "pmix.num.slots" (`uint32_t`)

Maximum number of processes that can simultaneously be executing in the specified realm. Note that this attribute is the equivalent to `PMIX_MAX_PROCS` - it is included in the PMIX Standard for historical reasons. Defaults to the `job` realm. In this context, this is the number of slots assigned to the specified application, which may be a subset of the slots allocated to the overall session and job.

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

Regular expression of nodes currently hosting processes in the specified realm - see [17.2.3.2](#) for an explanation of its generation. Defaults to the `job` realm. In this context, this is the regular expression of nodes currently hosting processes in the specified application.

1 **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)

2 Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults  
3 to the *job* realm. In this context, this is the comma-delimited list of nodes currently hosting  
4 processes in the specified application.

5 **PMIX\_PROC\_MAP** "pmix.pmap" (char\*)

6 Regular expression describing processes on each node in the specified realm - see 17.2.3.2  
7 for an explanation of its generation. Defaults to the *job* realm. In this context, this is the  
8 regular expression describing processes on each node in the specified application.

9 **PMIX\_APP\_MAP\_TYPE** "pmix.apmap.type" (char\*)

10 Type of mapping used to layout the application (e.g., *cyclic*).

11 **PMIX\_APP\_MAP\_REGEX** "pmix.apmap.regex" (char\*)

12 Regular expression describing the result of the process mapping.

## 13 6.1.4 Process realm attributes

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

19 Process-level information includes the following attributes:

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

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

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

23 Process rank within the job, starting from zero.

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

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

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

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

30 Rank of the specified process within its application.

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

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

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

36 Exit code returned when the specified process terminated.

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

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

1 **PMIX\_LOCAL\_RANK** "pmix.lrank" (uint16\_t)  
 2 Rank of the specified process on its node - refers to the numerical location (starting from  
 3 zero) of the process on its node when counting only those processes from the same job that  
 4 share the node, ordered by their overall rank within that job.

5 **PMIX\_NODE\_RANK** "pmix.nrank" (uint16\_t)  
 6 Rank of the specified process on its node spanning all jobs- refers to the numerical location  
 7 (starting from zero) of the process on its node when counting all processes (regardless of  
 8 job) that share the node, ordered by their overall rank within the job. The value represents a  
 9 snapshot in time when the specified process was started on its node and is not dynamically  
 10 adjusted as processes from other jobs are started or terminated on the node.

11 **PMIX\_PACKAGE\_RANK** "pmix.pkgrank" (uint16\_t)  
 12 Rank of the specified process on the *package* where this process resides - refers to the  
 13 numerical location (starting from zero) of the process on its package when counting only  
 14 those processes from the same job that share the package, ordered by their overall rank  
 15 within that job. Note that processes that are not bound to Processing Units (PUs) within a  
 16 single specific package cannot have a package rank.

17 **PMIX\_PROC\_PID** "pmix.ppid" (pid\_t)  
 18 Operating system PID of specified process.

19 **PMIX\_PROCDIR** "pmix.pdir" (char\*)  
 20 Full path to the subdirectory under **PMIX\_NSDIR** assigned to the specified process.

21 **PMIX\_CPuset** "pmix.cpuset" (char\*)  
 22 A string representation of the PU binding bitmap applied to the process upon launch. The  
 23 string shall begin with the name of the library that generated it (e.g., "hwloc") followed by a  
 24 colon and the bitmap string itself.

25 **PMIX\_CPuset\_BITMAP** "pmix.bitmap" (pmix\_cpuset\_t\*)  
 26 Bitmap applied to the process upon launch.

27 **PMIX\_CREDENTIAL** "pmix.cred" (char\*)  
 28 Security credential assigned to the process.

29 **PMIX\_SPawnED** "pmix.spawned" (bool)  
 30 **true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return  
 31 status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.

32 **PMIX\_REINcARNATION** "pmix.reinc" (uint32\_t)  
 33 Number of times this process has been re-instantiated - i.e, a value of zero indicates that the  
 34 process has never been restarted. 5

35 In addition, process-level information includes functional attributes directly associated with a  
 36 process - for example, the process-related fabric attributes included in Section 15.3 or the distance  
 37 attributes of Section 12.4.11.

## 38 6.1.5 Node realm keys

39 Information regarding the local node can be retrieved by directly requesting the node realm key in  
 40 the call to **PMIx\_Get** - the keys for node-related information are not shared across other realms.

1 The target process identifier will be ignored for keys that are not dependent upon it. Information  
2 about a node other than the local node can be retrieved by specifying the [PMIX\\_NODE\\_INFO](#)  
3 attribute in the *info* array along with either the [PMIX\\_HOSTNAME](#) or [PMIX\\_NODEID](#) qualifiers for  
4 the node of interest.

5 Node-level information includes the following keys:

6 **PMIX\_HOSTNAME** "pmix.hname" (char\*)

7 Name of the host, as returned by the `gethostname` utility or its equivalent.

8 **PMIX\_HOSTNAME\_ALIASES** "pmix.alias" (char\*)

9 Comma-delimited list of names by which the target node is known.

10 **PMIX\_NODEID** "pmix.nodeid" (uint32\_t)

11 Node identifier expressed as the node's index (beginning at zero) in an array of nodes within  
12 the active session. The value must be unique and directly correlate to the [PMIX\\_HOSTNAME](#)  
13 of the node - i.e., users can interchangeably reference the same location using either the  
14 [PMIX\\_HOSTNAME](#) or corresponding [PMIX\\_NODEID](#).

15 **PMIX\_NODE\_SIZE** "pmix.node.size" (uint32\_t)

16 Number of processes across all jobs that are executing upon the node.

17 **PMIX\_AVAIL\_PHYS\_MEMORY** "pmix.pmem" (uint64\_t)

18 Total available physical memory on a node.

19 The following attributes only return information regarding the *caller's* node - any node-related  
20 qualifiers shall be ignored. In addition, these attributes require specification of the namespace in the  
21 target process identifier except where noted - the value of the rank is ignored in all cases.

22 **PMIX\_LOCAL\_PEERS** "pmix.lpeers" (char\*)

23 Comma-delimited list of ranks that are executing on the local node within the specified  
24 namespace - shortcut for [PMIx\\_Resolve\\_peers](#) for the local node.

25 **PMIX\_LOCAL\_PROCS** "pmix.lprocs" (pmix\_proc\_t array)

26 Array of [pmix\\_proc\\_t](#) of all processes executing on the local node - shortcut for  
27 [PMIx\\_Resolve\\_peers](#) for the local node and a **NULL** namespace argument. The process  
28 identifier is ignored for this attribute.

29 **PMIX\_LOCALLDR** "pmix.llldr" (pmix\_rank\_t)

30 Lowest rank within the specified job on the node (defaults to current node in absence of  
31 [PMIX\\_HOSTNAME](#) or [PMIX\\_NODEID](#) qualifier).

32 **PMIX\_LOCAL\_CPUSSETS** "pmix.lcpus" (pmix\_data\_array\_t)

33 A [pmix\\_data\\_array\\_t](#) array of string representations of the PU binding bitmaps  
34 applied to each local *peer* on the caller's node upon launch. Each string shall begin with the  
35 name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap string  
36 itself. The array shall be in the same order as the processes returned by  
37 [PMIX\\_LOCAL\\_PEERS](#) for that namespace.

38 **PMIX\_LOCAL\_SIZE** "pmix.local.size" (uint32\_t)

39 Number of processes in the specified job or application realm on the caller's node. Defaults  
40 to job realm unless the [PMIX\\_APP\\_INFO](#) and the [PMIX\\_APPNUM](#) qualifiers are given.

1 In addition, node-level information includes functional attributes directly associated with a node -  
2 for example, the node-related fabric attributes included in Section 15.3.

## 3 6.2 Retrieval rules for reserved keys

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

9 In the case where the target and requester's namespaces differ, then the request is forwarded to the  
10 local PMIx server. Upon receiving the request, the server shall check its data storage for the  
11 specified namespace. If it already knows about this namespace, then it shall attempt to lookup the  
12 specified key, returning the value if it is found or the `PMIX_ERR_NOT_FOUND` error constant.

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

- 15 1. If the request included the `PMIX_IMMEDIATE` attribute, then the server will respond to the  
16 client with the `PMIX_ERR_NOT_FOUND` status.
- 17 2. If the host has provided the Direct Business Card Exchange (DBCX) module function interface  
18 (`pmix_server_dmodex_req_fn_t`), then the server shall pass the request to its host for  
19 servicing. The host is responsible for identifying a source of information on the specified  
20 namespace and retrieving it. The host is required to retrieve *all* of the information regarding the  
21 target namespace and return it to the requesting server in anticipation of follow-on requests. If  
22 the host cannot retrieve the namespace information, then it must respond with the  
23 `PMIX_ERR_NOT_FOUND` error constant unless the `PMIX_TIMEOUT` is given and reached (in  
24 which case, the host must respond with the `PMIX_ERR_TIMEOUT` constant).

25 Once the the PMIx server receives the namespace information, the server shall search it (again  
26 adhering to the prior data realm rules) for the requested key, returning the value if it is found or  
27 the `PMIX_ERR_NOT_FOUND` error constant.

- 28 3. If the host does not support the DBCX interface, then the server will respond to the client with  
29 the `PMIX_ERR_NOT_FOUND` status

### 30 6.2.1 Accessing information: examples

31 This section provides examples illustrating methods for accessing information from the various  
32 realms. The intent of the examples is not to provide comprehensive coding guidance, but rather to  
33 further illustrate the use of `PMIx_Get` for obtaining information on a *session*, *job*, *application*,  
34 *process*, and *node*.

## 1 6.2.1.1 Session-level information

2 The `PMIx_Get` API does not include an argument for specifying the *session* associated with the  
3 information being requested. Thus, requests for keys that are not specifically for session-level  
4 information must be accompanied by the `PMIX_SESSION_INFO` qualifier.

5 Example requests are shown below:

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

21 Information regarding a different session can be requested by adding the `PMIX_SESSION_ID`  
22 attribute identifying the target session. In this case, the *proc* argument to `PMIx_Get` will be  
23 ignored:

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

C

### 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:

C

```
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);
```

C

### 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:

C

```
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".
```



```

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

```

C

#### 22 6.2.1.4 Process-level information

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

#### 26 6.2.1.5 Node-level information

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

```

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

```

C



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

▲ C ▲

## 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. Beyond the restriction on name prefix, non-reserved keys are required to be unique across  
6 conflicting *scopes* as defined in Section 7.1.1.1 - e.g., a non-reserved key cannot be posted by the  
7 same process in both the **PMIX\_LOCAL** and **PMIX\_REMOTE** scopes (note that posting the key in  
8 the **PMIX\_GLOBAL** scope would have met the desired objective).

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

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

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

31 Note that the retrieval rules for non-reserved keys (detailed in Section 7.2) differ significantly from  
32 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

*PMIx v1.0*

C

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

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 one of the following error codes when the condition described occurs:

- 20 • [PMIX\\_ERR\\_BAD\\_PARAM](#) indicating a reserved key is provided in the *key* argument.

21 If none of the above return codes are appropriate, then an implementation must return either a  
22 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

#### 23 Description

24 Post a key-value pair for distribution. Depending upon the PMIx implementation, the posted value  
25 may be locally cached in the client's PMIx library until [PMIx\\_Commit](#) is called.

26 The provided *scope* determines the ability of other processes to access the posted data, as defined in  
27 Section 7.1.1.1 on page 96. Specific implementations may support different scope values, but all  
28 implementations must support at least [PMIX\\_GLOBAL](#).

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

## Advice to users

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\_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

C

**pmix\_status\_t**

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

C

1 **IN** `proc`  
2 process reference (handle)  
3 **IN** `key`  
4 key to retrieve (string)  
5 **IN** `val`  
6 Value to store (handle)

7 Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- `PMIX_ERR_BAD_PARAM` indicating a reserved key is provided in the `key` argument.

9 If none of the above return codes are appropriate, then an implementation must return either a  
10 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 11 **Description**

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

## 15 **7.1.3 PMIx\_Commit**

### 16 **Summary**

17 Post all previously `PMIx_Put` values for distribution.

### 18 **Format**

*PMIx v1.0*

19 `pmix_status_t PMIx_Commit(void);`

20 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 21 **Description**

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

### Advice to users

27 Users are advised to always include the call to `PMIx_Commit` in case the local implementation  
28 requires it. Note that posted data will not be circulated during `PMIx_Commit`. Availability of the  
29 data by other processes upon completion of `PMIx_Commit` therefore still relies upon the exchange  
30 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 within the appropriate scope as defined by the process that originally posted the  
13 key. If the value exists in a scope that contains the requesting process, then the value shall be  
14 returned. If the value exists, but in a scope that excludes the requesting process, then the server  
15 shall immediately return the **PMIX\_ERR\_EXISTS\_OUTSIDE\_SCOPE**.

16 If the value still isn't found and the **PMIX\_IMMEDIATE** attribute was given, then the library  
17 shall return the **PMIX\_ERR\_NOT\_FOUND** error constant to the requester. Otherwise, the PMIx  
18 server library will take one of the following actions:

- 19 • If the target process has a rank of **PMIX\_RANK\_UNDEF**, then this indicates that the key being  
20 requested is globally unique and *not* associated with a specific process. In this case, the server  
21 shall hold the request until either the data appears at the server or, if given, the  
22 **PMIX\_TIMEOUT** is reached. In the latter case, the server will return the  
23 **PMIX\_ERR\_TIMEOUT** status. Note that the server may, depending on PMIx implementation,  
24 never respond if the caller failed to specify a **PMIX\_TIMEOUT** and the requested key fails to  
25 arrive at the server.
- 26 • If the target process is *local* (i.e., attached to the same PMIx server), then the server will hold  
27 the request until either the target process provides the data or, if given, the **PMIX\_TIMEOUT**  
28 is reached. In the latter case, the server will return the **PMIX\_ERR\_TIMEOUT** status. Note  
29 that data which is posted via **PMIx\_Put** but not staged with **PMIx\_Commit** may, depending  
30 upon implementation, never appear at the server.
- 31 • If the target process is *remote* (i.e., not attached to the same PMIx server), the server will  
32 either:
  - 33 – If the host has provided the **pmix\_server\_dmodex\_req\_fn\_t** module function  
34 interface, then the server shall pass the request to its host for servicing. The host is  
35 responsible for determining the location of the target process and passing the request to the  
36 PMIx server at that location.

37 When the remote data request is received, the target PMIx server will check its cache for  
38 the specified key. If the key is not present, the request shall be held until either the target  
39 process provides the data or, if given, the **PMIX\_TIMEOUT** is reached. In the latter case,

1 the server will return the `PMIX_ERR_TIMEOUT` status. The host shall convey the result  
2 back to the originating PMIx server, which will reply to the requesting client with the result  
3 of the request when the host provides it.

4 Note that the target server may, depending on PMIx implementation, never respond if the  
5 caller failed to specify a `PMIX_TIMEOUT` and the target process fails to post the requested  
6 key.

- 7 – if the host does not support the `pmix_server_dmodex_req_fn_t` interface, then the  
8 server will immediately respond to the client with the `PMIX_ERR_NOT_FOUND` status

#### ▼ Advice to PMIx library implementers ▼

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

#### ▼ Advice to users ▼

12 Users are advised to always specify the `PMIX_TIMEOUT` value when retrieving non-reserved keys  
13 to avoid potential deadlocks should the specified key not become available.

## CHAPTER 8

# Query Operations

---

This chapter presents mechanisms for generalized queries that access information about the host environment and the system in general. The chapter presents the concept of a query followed by a detailed explanation of the queries APIs provided. The chapter compares the use of these APIs with `PMIx_Get`. The chapter concludes with detailed information about how to use the query interface to access information about what PMIx APIs an implementation supports as well as what attributes each supported API supports.

## 8.1 `PMIx_Query_info`

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

An example use-case for the `PMIx_Query_info_nb` API is to ensure clean job completion. Time-shared systems frequently impose maximum run times when assigning jobs to resource allocations. To shut down gracefully (e.g., to write a checkpoint before termination) it is necessary for an application to periodically query the resource manager for the time remaining in its allocation. This is especially true on systems for which allocation times may be shortened or lengthened from the original time limit. Many resource managers provide APIs to dynamically obtain this information, but each API is specific to the resource manager. PMIx supports this use-case by defining an attribute key (`PMIX_TIME_REMAINING`) that can be used with the `PMIx_Query_info_nb` interface to obtain the number of seconds remaining in the current job allocation.

PMIx sometimes provides multiple methods by which an application can obtain information or services. For this example, note that one could alternatively use the `PMIx_Register_event_handler` API to register for an event indicating incipient job termination, and then use the `PMIx_Job_control_nb` API to request that the host SMS generate an event a specified amount of time prior to reaching the maximum run time.



## 1 8.1.1 Query Structure

2 A PMIx query structure is composed of one or more keys and a list of qualifiers which provide  
3 additional information to describe the query. Keys which use the same qualifiers can be placed in  
4 the same query for compactness, though it is permissible to put each key in its own query.

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

*PMIx v2.0*

```
7 typedef struct pmix_query {  
8     char **keys;  
9     pmix_info_t *qualifiers;  
10    size_t nqual;  
11 } pmix_query_t;
```

12 where:

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

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

## 17 8.1.2 PMIx\_Query\_info

### 18 Summary

19 Query information about the system in general.

### 20 Format

```
21 pmix_status_t  
22 PMIx_Query_info(pmix_query_t queries[], size_t nqueries,  
23                pmix_info_t *info[], size_t *ninfo);
```

#### 24 IN queries

25 Array of query structures (array of handles)

#### 26 IN nqueries

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

#### 28 INOUT info

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

## INOUT *ninfo*

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

A successful return indicates that all data was found and has been returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- `PMIX_ERR_NOT_FOUND` None of the requested data was available.
- `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.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

If a value other than `PMIX_SUCCESS` or `PMIX_ERR_PARTIAL_SUCCESS` is returned, the *info* array shall be `NULL` and *ninfo* zero.

### Required Attributes

A call to this API can specify multiple queries. Each query is composed of a list of keys and a list of attributes which can influence that query. PMIx libraries and host environments that support this API are required to support the following attributes which are specified on a per-query basis:

`PMIX_QUERY_REFRESH_CACHE` "`pmix.qry.rfsh`" (bool)

Retrieve updated information from server. NO QUALIFIERS.

`PMIX_SESSION_INFO` "`pmix.ssn.info`" (bool)

Return information regarding the session realm of the target process.

`PMIX_JOB_INFO` "`pmix.job.info`" (bool)

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

`PMIX_APP_INFO` "`pmix.app.info`" (bool)

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

`PMIX_NODE_INFO` "`pmix.node.info`" (bool)

Return information from the node realm regarding the node upon which the specified process is executing. If information about a node other than the one containing the specified process is desired, then the attribute array must also contain either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute identifying the desired target. This is useful for requesting information about a specific node even if the identity of processes running on that node are not known..

1 **PMIX\_PROC\_INFO** "pmix.proc.info" (bool)

2 Return information regarding the target process. This attribute is technically not required as  
3 the **PMIX\_Get** API specifically identifies the target process in its parameters. However, it is  
4 included here for completeness.

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

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

12 **PMIX\_NAMESPACE** "pmix.namespace" (char\*)

13 Namespace of the job - may be a numerical value expressed as a string, but is often an  
14 alphanumeric string carrying information solely of use to the system. Required to be unique  
15 within the scope of the host environment. Specifies the namespace of the process whose  
16 information is being requested. Must be accompanied by the **PMIX\_RANK** attribute. Only  
17 required when the request is for information on a specific process.

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

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

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

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

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

27 Request attributes supported by the PMIx client library.

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

29 Request attributes supported by the PMIx server library.

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

31 Request attributes supported by the host environment.

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

33 Request attributes supported by the PMIx tool library functions.

34 Note that inclusion of both the **PMIX\_PROCID** directive and either the **PMIX\_NAMESPACE** or the  
35 **PMIX\_RANK** attribute will return a **PMIX\_ERR\_BAD\_PARAM** result, and that the inclusion of a  
36 process identifier must apply to all keys in that **pmix\_query\_t**. Queries for information on  
37 multiple specific processes therefore requires submitting multiple **pmix\_query\_t** structures,  
38 each referencing one process. Directives which are not applicable to a key are ignored.

▲-----▲

1 An implementation is not required to support any keys. If a key is unsupported, the implementation  
2 should handle that key in the same way that it is required to handle a key which it cannot find. The  
3 following keys may be specified in a query:

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

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

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\_NAMESPACE**  
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.phtable" (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\_NAMESPACE** indicating the namespace whose process table is being queried.

23 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (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\_NAMESPACE** 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.

34 **PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)

35 Return information on memory usage for the processes indicated in the qualifiers.  
36 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** and **PMIX\_RANK**, or **PMIX\_PROCID** of  
37 specific process(es) whose memory usage is being requested.



1 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)  
2 Report only average values for sampled information. NO QUALIFIERS.

3 **PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.minmax" (bool)  
4 Report minimum and maximum values. NO QUALIFIERS.

5 **PMIX\_QUERY\_ALLOC\_STATUS** "pmix.query.alloc" (char\*)  
6 String identifier of the allocation whose status is being requested. NO QUALIFIERS.

7 **PMIX\_TIME\_REMAINING** "pmix.time.remaining" (char\*)  
8 Query number of seconds (`uint32_t`) remaining in allocation for the specified namespace.  
9 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** of the namespace whose info is being  
10 requested (defaults to allocation containing the caller).

11 **PMIX\_SERVER\_URI** "pmix.srvr.uri" (char\*)  
12 URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's  
13 PMIx connection. Defaults to requesting the information for the local PMIx server.

14 **PMIX\_CLIENT\_AVG\_MEMORY** "pmix.cl.mem.avg" (float)  
15 Average Megabytes of memory used by client processes on node. OPTIONAL  
16 QUALIFIERS: **PMIX\_HOSTNAME** or **PMIX\_NODEID** (defaults to caller's node).

17 **PMIX\_DAEMON\_MEMORY** "pmix.dmn.mem" (float)  
18 Megabytes of memory currently used by the RM daemon on the node. OPTIONAL  
19 QUALIFIERS: **PMIX\_HOSTNAME** or **PMIX\_NODEID** (defaults to caller's node).

20 **PMIX\_QUERY\_AUTHORIZATIONS** "pmix.qry.auths" (bool)  
21 Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.

22 **PMIX\_PROC\_PID** "pmix.ppid" (`pid_t`)  
23 Operating system PID of specified process.

24 **PMIX\_PROC\_STATE\_STATUS** "pmix.proc.state" (`pmix_proc_state_t`)  
25 State of the specified process as of the last report - may not be the actual current state based  
26 on update rate.

## Description

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

31  
32 The returned *status* indicates if requested data was found or not. The returned *info* array will  
33 contain a **PMIX\_QUERY\_RESULTS** element for each query of the *queries* array. If qualifiers were  
34 included in the query, then the first element of each results array shall contain the  
35 **PMIX\_QUERY\_QUALIFIERS** key with a `pmix_data_array_t` containing the qualifiers. The  
36 remaining `pmix_info_t` shall contain the results of the query, one entry for each key that was  
37 found. Note that duplicate keys in the *queries* array shall result in duplicate responses within the  
38 constraints of the accompanying qualifiers. The caller is responsible for releasing the returned array.

## Advice to PMIx library implementers

It is recommended that information returned from `PMIx_Query_info` be locally cached so that retrieval by subsequent calls to `PMIx_Get`, `PMIx_Query_info`, or `PMIx_Query_info_nb` can succeed with minimal overhead. The local cache shall be checked prior to querying the PMIx server and/or the host environment. Queries that include the `PMIX_QUERY_REFRESH_CACHE` attribute shall bypass the local cache and retrieve a new value for the query, refreshing the values in the cache upon return.

### 8.1.3 `PMIx_Query_info_nb`

#### Summary

Query information about the system in general.

#### Format

*PMIx v2.0*

C

```
pmix_status_t
PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,
                  pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

#### IN `queries`

Array of query structures (array of handles)

#### IN `nqueries`

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

#### IN `cbfunc`

Callback function `pmix_info_cbfunc_t` (function reference)

#### IN `cbdata`

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

A successful return indicates that the request has been accepted for processing. The provided callback function will only be executed upon successful return of the operation. Note that the library must not invoke the callback function prior to returning from the API.

Returns `PMIX_SUCCESS` or a negative value indicating the error.

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

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

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

### Required Attributes

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

9 **PMIX\_QUERY\_REFRESH\_CACHE** "pmix.qry.rfsh" (bool)

10 Retrieve updated information from server. NO QUALIFIERS.

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

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

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

14 Return information regarding the job realm corresponding to the namespace in the target  
15 process' identifier.

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

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

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

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

30 **PMIX\_PROC\_INFO** "pmix.proc.info" (bool)

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

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




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

7 **PMIX\_NAMESPACE** "pmix.namespace" (char\*)

8 Namespace of the job - may be a numerical value expressed as a string, but is often an  
9 alphanumeric string carrying information solely of use to the system. Required to be unique  
10 within the scope of the host environment. Specifies the namespace of the process whose  
11 information is being requested. Must be accompanied by the **PMIX\_RANK** attribute. Only  
12 required when the request is for information on a specific process.

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

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

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

18 Request attributes supported by the PMIx client library.

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

20 Request attributes supported by the PMIx server library.

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

22 Request attributes supported by the host environment.

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

24 Request attributes supported by the PMIx tool library functions.

25 Note that inclusion of both the **PMIX\_PROCID** directive and either the **PMIX\_NAMESPACE** or the  
26 **PMIX\_RANK** attribute will return a **PMIX\_ERR\_BAD\_PARAM** result, and that the inclusion of a  
27 process identifier must apply to all keys in that **pmix\_query\_t**. Queries for information on  
28 multiple specific processes therefore requires submitting multiple **pmix\_query\_t** structures,  
29 each referencing one process. Directives which are not applicable to a key are ignored.

▲-----▲  
▼-----▼ **Optional Attributes** -----▼ 

30 An implementation is not required to support any keys. If a key is unsupported, the implementation  
31 should handle that key in the same way that it is required to handle a key which it cannot find. The  
32 following keys may be specified in a query:

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**.



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

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

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

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

12 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.phtable" (char\*)  
13 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
14 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
15 **PMIX\_NAMESPACE** indicating the namespace whose process table is being queried.

16 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (char\*)  
17 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
18 process in the specified namespace executing on the same node as the requester, ordered by  
19 process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace  
20 whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME**  
21 indicating the host whose local process table is being queried. By default, the query assumes  
22 that the host upon which the request was made is to be used.

23 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)  
24 Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

25 **PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)  
26 Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

27 **PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)  
28 Return information on memory usage for the processes indicated in the qualifiers.  
29 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** and **PMIX\_RANK**, or **PMIX\_PROCID** of  
30 specific process(es) whose memory usage is being requested.

31 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)  
32 Report only average values for sampled information. NO QUALIFIERS.

33 **PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.minmax" (bool)  
34 Report minimum and maximum values. NO QUALIFIERS.

35 **PMIX\_QUERY\_ALLOC\_STATUS** "pmix.query.alloc" (char\*)  
36 String identifier of the allocation whose status is being requested. NO QUALIFIERS.

37 **PMIX\_TIME\_REMAINING** "pmix.time.remaining" (char\*)

1 Query number of seconds (`uint32_t`) remaining in allocation for the specified namespace.  
2 OPTIONAL QUALIFIERS: `PMIX_NAMESPACE` of the namespace whose info is being  
3 requested (defaults to allocation containing the caller).

4 `PMIX_SERVER_URI` "`pmix.srvr.uri`" (`char*`)

5 URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's  
6 PMIx connection. Defaults to requesting the information for the local PMIx server.

7 `PMIX_CLIENT_AVG_MEMORY` "`pmix.cl.mem.avg`" (`float`)

8 Average Megabytes of memory used by client processes on node. OPTIONAL  
9 QUALIFIERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller's node).

10 `PMIX_DAEMON_MEMORY` "`pmix.dmn.mem`" (`float`)

11 Megabytes of memory currently used by the RM daemon on the node. OPTIONAL  
12 QUALIFIERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller's node).

13 `PMIX_QUERY_AUTHORIZATIONS` "`pmix.qry.auths`" (`bool`)

14 Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.

15 `PMIX_PROC_PID` "`pmix.ppid`" (`pid_t`)

16 Operating system PID of specified process.

17 `PMIX_PROC_STATE_STATUS` "`pmix.proc.state`" (`pmix_proc_state_t`)

18 State of the specified process as of the last report - may not be the actual current state based  
19 on update rate.



## 20 Description

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



## 22 8.1.4 Query keys

23 The following keys may be queried using the `PMIx_Query_info` and  
24 `PMIx_Query_info_nb` APIs:



25 `PMIX_QUERY_SUPPORTED_KEYS` "`pmix.qry.keys`" (`char*`)

26 Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS.

27 `PMIX_QUERY_SUPPORTED_QUALIFIERS` "`pmix.qryquals`" (`char*`)

28 Return comma-delimited list of qualifiers supported by a query on the provided key, instead  
29 of actually performing the query on the key. NO QUALIFIERS.

30 `PMIX_QUERY_NAMESPACES` "`pmix.qry.ns`" (`char*`)

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

32 `PMIX_QUERY_NAMESPACE_INFO` "`pmix.qry.nsinfo`" (`pmix_data_array_t*`)

33 Return an array of active namespace information - each element will itself contain an array  
34 including the namespace plus the command line of the application executing within it.

35 OPTIONAL QUALIFIERS: `PMIX_NAMESPACE` of specific namespace whose info is being  
36 requested.

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

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

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

10 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.phtable" (char\*)  
11 Returns a (pmix\_data\_array\_t) array of pmix\_proc\_info\_t, one entry for each  
12 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
13 **PMIX\_NAMESPACE** indicating the namespace whose process table is being queried.

14 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (char\*)  
15 Returns a (pmix\_data\_array\_t) array of pmix\_proc\_info\_t, one entry for each  
16 process in the specified namespace executing on the same node as the requester, ordered by  
17 process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace  
18 whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME**  
19 indicating the host whose local process table is being queried. By default, the query assumes  
20 that the host upon which the request was made is to be used.

21 **PMIX\_QUERY\_AUTHORIZATIONS** "pmix.qry.auths" (bool)  
22 Return operations the PMIX tool is authorized to perform. NO QUALIFIERS.

23 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)  
24 Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

25 **PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)  
26 Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

27 **PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)  
28 Return information on memory usage for the processes indicated in the qualifiers.  
29 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** and **PMIX\_RANK**, or **PMIX\_PROCID** of  
30 specific process(es) whose memory usage is being requested.

31 **PMIX\_TIME\_REMAINING** "pmix.time.remaining" (char\*)  
32 Query number of seconds (uint32\_t) remaining in allocation for the specified namespace.  
33 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** of the namespace whose info is being  
34 requested (defaults to allocation containing the caller).

35 **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** "pmix.qry.attrs" (bool)  
36 Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or  
37 more of **PMIX\_CLIENT\_FUNCTIONS**, **PMIX\_SERVER\_FUNCTIONS**,  
38 **PMIX\_TOOL\_FUNCTIONS**, and **PMIX\_HOST\_FUNCTIONS**.

39 **PMIX\_QUERY\_NUM\_PSETS** "pmix.qry.psetnum" (size\_t)  
40 Return the number of process sets defined in the specified range (defaults to  
41 **PMIX\_RANGE\_SESSION**).

42 **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_QUERY_AVAIL_SERVERS` "pmix.qry.asrvrs" (`pmix_data_array_t*`)

6 Return an array of `pmix_info_t`, each element itself containing a  
7 `PMIX_SERVER_INFO_ARRAY` entry holding all available data for a server on this node to  
8 which the caller might be able to connect.

9 These keys are used to query memory available and used in the system.

10 `PMIX_AVAIL_PHYS_MEMORY` "pmix.pmem" (`uint64_t`)

11 Total available physical memory on a node. OPTIONAL QUALIFERS: `PMIX_HOSTNAME`  
12 or `PMIX_NODEID` (defaults to caller's node).

13 `PMIX_DAEMON_MEMORY` "pmix.dmn.mem" (`float`)

14 Megabytes of memory currently used by the RM daemon on the node. OPTIONAL  
15 QUALIFERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller's node).

16 `PMIX_CLIENT_AVG_MEMORY` "pmix.cl.mem.avg" (`float`)

17 Average Megabytes of memory used by client processes on node. OPTIONAL  
18 QUALIFERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller's node).

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

## 21 8.1.5 Query attributes

22 Attributes used to direct behavior of the `PMIx_Query_info` and `PMIx_Query_info_nb`  
23 APIs:

24 `PMIX_QUERY_RESULTS` "pmix.qry.res" (`pmix_data_array_t`)

25 Contains an array of query results for a given `pmix_query_t` passed to the  
26 `PMIx_Query_info` APIs. If qualifiers were included in the query, then the first element  
27 of the array shall be the `PMIX_QUERY_QUALIFIERS` attribute containing those qualifiers.  
28 Each of the remaining elements of the array is a `pmix_info_t` containing the query key  
29 and the corresponding value returned by the query. This attribute is solely for reporting  
30 purposes and cannot be used in `PMIx_Get` or other query operations.

31 `PMIX_QUERY_QUALIFIERS` "pmix.qryquals" (`pmix_data_array_t`)

32 Contains an array of qualifiers that were included in the query that produced the provided  
33 results. This attribute is solely for reporting purposes and cannot be used in `PMIx_Get` or  
34 other query operations.

35 `PMIX_QUERY_REFRESH_CACHE` "pmix.qry.rfsh" (`bool`)

36 Retrieve updated information from server. NO QUALIFIERS.

37 `PMIX_QUERY_LOCAL_ONLY` "pmix.qry.local" (`bool`)

1 Constrain the query to local information only. NO QUALIFIERS.

2 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)

3 Report only average values for sampled information. NO QUALIFIERS.

4 **PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.minmax" (bool)

5 Report minimum and maximum values. NO QUALIFIERS.

6 **PMIX\_QUERY\_ALLOC\_STATUS** "pmix.query.alloc" (char\*)

7 String identifier of the allocation whose status is being requested. NO QUALIFIERS.

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_NAMESPACE` and  
10 including at least one of the rendezvous-required pieces of information.

11 **PMIX\_CLIENT\_FUNCTIONS** "pmix.client.fns" (bool)

12 Request a list of functions supported by the PMIx client library.

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

14 Request attributes supported by the PMIx client library.

15 **PMIX\_SERVER\_FUNCTIONS** "pmix.srvr.fns" (bool)

16 Request a list of functions supported by the PMIx server library.

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

18 Request attributes supported by the PMIx server library.

19 **PMIX\_HOST\_FUNCTIONS** "pmix.srvr.fns" (bool)

20 Request a list of functions supported by the host environment.

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

22 Request attributes supported by the host environment.

23 **PMIX\_TOOL\_FUNCTIONS** "pmix.tool.fns" (bool)

24 Request a list of functions supported by the PMIx tool library.

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

26 Request attributes supported by the PMIx tool library functions.

### 27 8.1.5.1 Query structure support macros

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

#### 29 Initialize the query structure

30 Initialize the `pmix_query_t` fields

PMIx v2.0

31 **PMIX\_QUERY\_CONSTRUCT** (m)

32 **IN** m

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

1 **Destruct the query structure**

2 Destruct the `pmix_query_t` fields



3 **PMIX\_QUERY\_DESTRUCT (m)**



4 **IN** m

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

6 **Create a query array**

7 Allocate and initialize an array of `pmix_query_t` structures

*PMIx v2.0*



8 **PMIX\_QUERY\_CREATE (m, n)**



9 **INOUT** m

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

11 **IN** n

12 Number of structures to be allocated (`size_t`)

13 **Free a query structure**

14 Release a `pmix_query_t` structure

*PMIx v4.0*



15 **PMIX\_QUERY\_RELEASE (m)**



16 **IN** m

17 Pointer to a `pmix_query_t` structure (handle)

18 **Free a query array**

19 Release an array of `pmix_query_t` structures

*PMIx v2.0*



20 **PMIX\_QUERY\_FREE (m, n)**



21 **IN** m

22 Pointer to the array of `pmix_query_t` structures (handle)

23 **IN** n

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

## 1 Create the info array of query qualifiers

2 Create an array of `pmix_info_t` structures for passing query qualifiers, updating the `nqual` field  
3 of the `pmix_query_t` structure.

```
4 PMIX_QUERY_QUALIFIERS_CREATE(m, n) C
```

5 **IN** `m`  
6 Pointer to the `pmix_query_t` structure (handle)  
7 **IN** `n`  
8 Number of qualifiers to be allocated (`size_t`)

## 9 8.2 PMIx\_Resolve\_peers

10 There are a number of common queries for which PMIx provides convenience routines. These APIs  
11 provide simplified access to commonly requested queries. Due to their simplified interface, these  
12 APIs cannot be customized through the use of attributes. If a more specialized version of these  
13 queries are required, similar functionality can often be accessed through the `PMIx_Query_info`  
14 or `PMIx_Query_info_nb` APIs.

### 15 Summary

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

### 17 Format

*PMIx v1.0*

```
18 pmix_status_t  
19 PMIx_Resolve_peers(const char *nodename,  
20 const pmix_namespace_t nspace,  
21 pmix_proc_t **procs, size_t *nprocs); C
```

22 **IN** `nodename`  
23 Name of the node to query - `NULL` can be used to denote the current local node (string)  
24 **IN** `namespace`  
25 namespace (string)  
26 **OUT** `procs`  
27 Array of process structures (array of handles)  
28 **OUT** `nprocs`  
29 Number of elements in the `procs` array (integer)  
30 Returns `PMIX_SUCCESS` or a negative value indicating the error.

## Description

Given a *nodename*, return the array of processes within the specified *nspace* that are executing on that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will 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.

## 8.2.1 PMIx\_Resolve\_nodes

### Summary

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

### Format

PMIx v1.0

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

### IN nspace

Namespace (string)

### OUT nodelist

Comma-delimited list of nodenames (string)

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

### Description

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

## 8.3 Using Get vs Query

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

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

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

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



- system-level information (such as the available WLM queues) that would generally not be included in job-level information provided at job start.
- dynamic information such as application and queue status, and resource utilization statistics. Note that the `PMIX_QUERY_REFRESH_CACHE` attribute must be provided on each query to ensure current data is returned.
- information created post job start, such as process tables.
- information requiring more complex search criteria than supported by the simpler `PMIx_Get` API.
- queries focused on retrieving multi-attribute blocks of data with a single request, thus bypassing the single-key limitation of the `PMIx_Get` API.

In theory, all information can be accessed via `PMIx_Query_info` as the local cache is typically the same datastore searched by `PMIx_Get`. However, in practice, the overhead associated with the *query* operation may (depending upon implementation) be higher than the simpler *get* operation due to the need to construct and process the more complex `pmix_query_t` structure. Thus, requests for a single key value are likely to be accomplished faster with `PMIx_Get` versus the *query* operation.

## 8.4 Accessing attribute support information

Information as to which attributes are supported by either the PMIx implementation or its host environment can be obtained via the `PMIx_Query_info` APIs. The `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute must be listed as the first entry in the *keys* field of the `pmix_query_t` structure, followed by the name of the function whose attribute support is being requested - support for multiple functions can be requested simultaneously by simply adding the function names to the array of *keys*. Function names *must* be given as user-level API names - e.g., “`PMIx_Get`”, “`PMIx_server_setup_application`”, or “`PMIx_tool_attach_to_server`”.

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

- `PMIX_CLIENT_FUNCTIONS` "`pmix.client.fns`" (`bool`)  
Request a list of functions supported by the PMIx client library.
- `PMIX_CLIENT_ATTRIBUTES` "`pmix.client.attrs`" (`bool`)  
Request attributes supported by the PMIx client library.
- `PMIX_SERVER_FUNCTIONS` "`pmix.srvr.fns`" (`bool`)  
Request a list of functions supported by the PMIx server library.
- `PMIX_SERVER_ATTRIBUTES` "`pmix.srvr.attrs`" (`bool`)

Request attributes supported by the PMIx server library.

- **PMIX\_HOST\_FUNCTIONS** "pmix.srvr.fns" (bool)  
Request a list of functions supported by the host environment.
- **PMIX\_HOST\_ATTRIBUTES** "pmix.host.attrs" (bool)  
Request attributes supported by the host environment.
- **PMIX\_TOOL\_FUNCTIONS** "pmix.tool.fns" (bool)  
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.

Unlike other queries, queries for attribute support can result in the number of returned `pmix_info_t` structures being different from the number of queries. Each element in the returned array will correspond to a pair of specified attribute level and function in the query, where the *key* is the function and the *value* contains a `pmix_data_array_t` of `pmix_info_t`. Each element of the array is marked by a *key* indicating the requested attribute *level* with a *value* composed of a `pmix_data_array_t` of `pmix_regattr_t`, each describing a supported attribute for that function, as illustrated in Fig. 8.1 below where the requestor asked for supported attributes of `PMIx_Get` at the *client* and *server* levels, plus attributes of `PMIx_Allocation_request` at all levels.

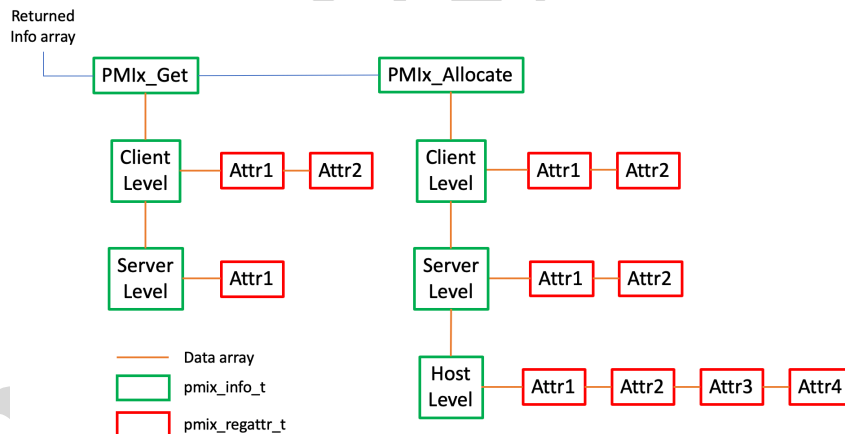


Figure 8.1.: Returned information hierarchy for attribute support request

The array of returned structures, and their child arrays, are subject to the return rules for the `PMIx_Query_info_nb` API. For example, a request for supported attributes of the `PMIx_Get` function that includes the *host* level will return values for the *client* and *server* levels, plus an array element with a *key* of `PMIX_HOST_ATTRIBUTES` and a value type of `PMIX_UNDEF` indicating that no attributes are supported at that level.

## CHAPTER 9

# 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*.

## 9.1 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 indicating the error.

## 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_GRPID` 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 9.2 PMIx\_Publish\_nb

### 2 Summary

3 Nonblocking [PMIx\\_Publish](#) routine.

### 4 *PMIx v1.0* Format

C

5 `pmix_status_t`

```
6 PMIx_Publish_nb(const pmix_info_t info[], size_t ninfo,  
7                 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

8 **IN** `info`

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

10 **IN** `ninfo`

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

12 **IN** `cbfunc`

13 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

14 **IN** `cbdata`

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

16 A successful return indicates that the request is being processed and the result will be returned in  
17 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
18 from the API. The callback function, *cbfunc*, is only called when [PMIX\\_SUCCESS](#) is returned.

19 Returns [PMIX\\_SUCCESS](#) or one of the following error codes when the condition described occurs:

- 20 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
21 returned *success* - the *cbfunc* will *not* be called.

22 If none of the above return codes are appropriate, then an implementation must return either a  
23 general PMIx error code or an implementation defined error code as described in Section [3.1.1](#).

### 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\\_GRPID](#) attributes of the client process that  
28 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

Nonblocking **PMIx\_Publish** routine.

## 9.3 Publish-specific constants

The following constants are defined for use with the **PMIx\_Publish** APIs:

**PMIX\_ERR\_DUPLICATE\_KEY** The provided key has already been published on the same data range.

## 9.4 Publish-specific attributes

The following attributes are defined for use with the **PMIx\_Publish** APIs:

**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.

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 9.5 Publish-Lookup Datatypes

6 The following data types are defined for use with the **PMIx\_Publish** APIs.

### 7 9.5.1 Range of Published Data

8 *PMIx v1.0*

9 The **pmix\_data\_range\_t** structure is a **uint8\_t** type that defines a range for both data  
10 *published* via the **PMIx\_Publish** API and generated events. The following constants can be used  
11 to set a variable of the type **pmix\_data\_range\_t**.

12 **PMIX\_RANGE\_UNDEF** Undefined range.  
13 **PMIX\_RANGE\_RM** Data is intended for the host environment, or lookup is restricted to data  
14 published by the host environment.  
15 **PMIX\_RANGE\_LOCAL** Data is only available to processes on the local node, or lookup is  
16 restricted to data published by processes on the local node of the requester.  
17 **PMIX\_RANGE\_NAMESPACE** Data is only available to processes in the same namespace, or  
18 lookup is restricted to data published by processes in the same namespace as the requester.  
19 **PMIX\_RANGE\_SESSION** Data is only available to all processes in the session, or lookup is  
20 restricted to data published by other processes in the same session as the requester.  
21 **PMIX\_RANGE\_GLOBAL** Data is available to all processes, or lookup is open to data published  
22 by anyone.  
23 **PMIX\_RANGE\_CUSTOM** Data is available only to processes as specified in the  
24 **pmix\_info\_t** associated with this call, or lookup is restricted to data published by  
25 processes as specified in the **pmix\_info\_t**.  
26 **PMIX\_RANGE\_PROC\_LOCAL** Data is only available to this process, or lookup is restricted to  
27 data published by this process.  
28 **PMIX\_RANGE\_INVALID** Invalid value - typically used to indicate that a range has not yet  
29 been set.

### 29 9.5.2 Data Persistence Structure

30 *PMIx v1.0*

31 The **pmix\_persistence\_t** structure is a **uint8\_t** type that defines the policy for data  
32 published by clients via the **PMIx\_Publish** API. The following constants can be used to set a  
33 variable of the type **pmix\_persistence\_t**.

34 **PMIX\_PERSIST\_INDEF** Retain data until specifically deleted.  
35 **PMIX\_PERSIST\_FIRST\_READ** Retain data until the first access, then the data is deleted.  
36 **PMIX\_PERSIST\_PROC** Retain data until the publishing process terminates.  
37 **PMIX\_PERSIST\_APP** Retain data until the application terminates.  
38 **PMIX\_PERSIST\_SESSION** Retain data until the session/allocation terminates.  
39 **PMIX\_PERSIST\_INVALID** Invalid value - typically used to indicate that a persistence has  
not yet been set.

## 1 9.6 PMIx\_Lookup

### 2 Summary

3 Lookup information published by this or another process with [PMIx\\_Publish](#) or  
4 [PMIx\\_Publish\\_nb](#).

### 5 *PMIx v1.0* Format

C

6 `pmix_status_t`

7 `PMIx_Lookup(pmix_pdata_t data[], size_t ndata,`  
8 `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 `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- 18 • [PMIX\\_ERR\\_NOT\\_FOUND](#) None of the requested data could be found within the requester's  
19     range.
- 20 • [PMIX\\_ERR\\_PARTIAL\\_SUCCESS](#) Some of the requested data was found. Any key that cannot  
21     be found will return with a data type of [PMIX\\_UNDEF](#) in the associated *value* struct. Note that  
22     the specific reason for a particular piece of missing information (e.g., lack of permissions) cannot  
23     be communicated back to the requester in this situation.
- 24 • [PMIX\\_ERR\\_NO\\_PERMISSIONS](#) All of the requested data was found and range restrictions  
25     were met for each specified key, but none of the matching data could be returned due to lack of  
26     access permissions.

27 If none of the above return codes are appropriate, then an implementation must return either a  
28 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Required Attributes

29 PMIx libraries are not required to directly support any attributes for this function. However, any  
30 provided attributes must be passed to the host environment for processing, and the PMIx library is  
31 required to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process that is  
32 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

Lookup information published by this or another process. By default, the search will be constrained to publishers that fall within the **PMIX\_RANGE\_SESSION** range in case duplicate keys exist on different ranges. Changes to the range (e.g., expanding the search to all potential publishers via the **PMIX\_RANGE\_GLOBAL** constant), and any additional directives, can be provided in the **pmix\_info\_t** array. Data is returned per the retrieval rules of Section 9.8.

The *data* parameter consists of an array of **pmix\_pdata\_t** structures with the keys specifying the requested information. Data will be returned for each **key** field in the associated **value** field of this structure as per the above description of return values. The **proc** field in each **pmix\_pdata\_t** structure will contain the namespace/rank of the process that published the data.

### Advice to users

Although this is a blocking function, it will not wait by default for the requested data to be published. Instead, it will block for the time required by the datastore to lookup its current data and return any found items. Thus, the caller is responsible for either ensuring that data is published prior to executing a lookup, using **PMIX\_WAIT** to instruct the datastore to wait for the data to be published, or retrying until the requested data is found.

## 9.7 PMIx\_Lookup\_nb

### Summary

Nonblocking version of **PMIx\_Lookup**.

## Format

C

```
pmix_status_t
PMIx_Lookup_nb(char **keys,
               const pmix_info_t info[], size_t ninfo,
               pmix_lookup_cbfunc_t cbfunc, void *cbdata);
```

C

- IN keys**  
NULL-terminated array of keys (array of strings)
- IN info**  
Array of info structures (array of handles)
- IN ninfo**  
Number of elements in the *info* array (integer)
- IN cbfunc**  
Callback function (handle)
- IN cbdata**  
Callback data to be provided to the callback function (pointer)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

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

- **PMIX\_SUCCESS** All data was found and has been returned.
- **PMIX\_ERR\_NOT\_FOUND** None of the requested data was available within the requester's range. The *pdata* array in the callback function shall be **NULL** and the *npdata* parameter set to zero.
- **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the requested data was found. Only found data will be included in the returned *pdata* array. 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.
- **PMIX\_ERR\_NOT\_SUPPORTED** There is no available datastore (either at the host environment 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_GRPID` 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.

### 9.7.1 Lookup Returned Data Structure

The `pmix_pdata_t` structure is used by `PMIx_Lookup` to describe the data being accessed.

*PMIx v1.0*

```
typedef struct pmix_pdata {
    pmix_proc_t proc;
    pmix_key_t key;
    pmix_value_t value;
} 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*.

## 1 9.7.1.1 Lookup data structure support macros

2 The following macros are provided to support the `pmix_pdata_t` structure.

### 3 Initialize the pdata structure

4 Initialize the `pmix_pdata_t` fields

*PMIx v1.0*

▼ `PMIX_PDATA_CONSTRUCT` C

5 `PMIX_PDATA_CONSTRUCT` (m)

▲ `PMIX_PDATA_CONSTRUCT` C

6 **IN** m

7 Pointer to the structure to be initialized (pointer to `pmix_pdata_t`)

### 8 Destruct the pdata structure

9 Destruct the `pmix_pdata_t` fields

*PMIx v1.0*

▼ `PMIX_PDATA_DESTRUCT` C

10 `PMIX_PDATA_DESTRUCT` (m)

▲ `PMIX_PDATA_DESTRUCT` C

11 **IN** m

12 Pointer to the structure to be destructed (pointer to `pmix_pdata_t`)

### 13 Create a pdata array

14 Allocate and initialize an array of `pmix_pdata_t` structures

*PMIx v1.0*

▼ `PMIX_PDATA_CREATE` C

15 `PMIX_PDATA_CREATE` (m, n)

▲ `PMIX_PDATA_CREATE` C

16 **INOUT** m

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

18 **IN** n

19 Number of structures to be allocated (`size_t`)

### 20 Free a pdata structure

21 Release a `pmix_pdata_t` structure

*PMIx v4.0*

▼ `PMIX_PDATA_RELEASE` C

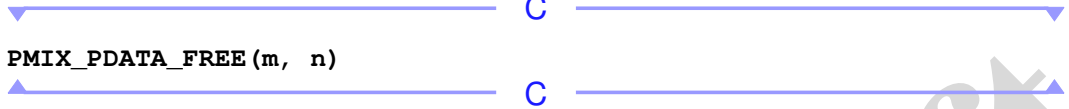
22 `PMIX_PDATA_RELEASE` (m)

▲ `PMIX_PDATA_RELEASE` C

23 **IN** m

24 Pointer to a `pmix_pdata_t` structure (handle)

1 **Free a pdata array**  
2 Release an array of `pmix_pdata_t` structures



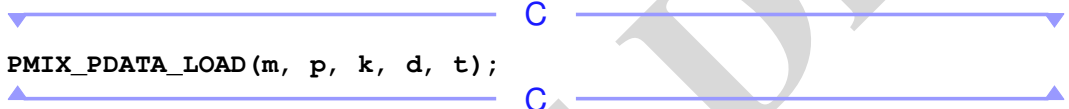
3 **PMIX\_PDATA\_FREE(m, n)**

4 **IN m**  
5 Pointer to the array of `pmix_pdata_t` structures (handle)  
6 **IN n**  
7 Number of structures in the array (`size_t`)

### 8 **Load a lookup data structure**

9 This macro simplifies the loading of key, process identifier, and data into a `pmix_pdata_t` by  
10 correctly assigning values to the structure's fields.

*PMIx v1.0*



11 **PMIX\_PDATA\_LOAD(m, p, k, d, t);**

12 **IN m**  
13 Pointer to the `pmix_pdata_t` structure into which the key and data are to be loaded  
14 (pointer to `pmix_pdata_t`)  
15 **IN p**  
16 Pointer to the `pmix_proc_t` structure containing the identifier of the process being  
17 referenced (pointer to `pmix_proc_t`)  
18 **IN k**  
19 String key to be loaded - must be less than or equal to `PMIX_MAX_KEYLEN` in length  
20 (handle)  
21 **IN d**  
22 Pointer to the data value to be loaded (handle)  
23 **IN t**  
24 Type of the provided data value (`pmix_data_type_t`)

### Advice to users

25 Key, process identifier, and data will all be copied into the `pmix_pdata_t` - thus, the source  
26 information can be modified or free'd without affecting the copied data once the macro has  
27 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_PDATA_XFER(d, s);
```

**IN** `d`

Pointer to the destination `pmix_pdata_t` (pointer to `pmix_pdata_t`)

**IN** `s`

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.

## 9.7.2 Lookup Callback Function

### Summary

The `pmix_lookup_cfunc_t` is used by `PMIx_Lookup_nb` to return data.

*PMIx v1.0*

```
typedef void (*pmix_lookup_cfunc_t)
    (pmix_status_t status,
     pmix_pdata_t data[], size_t ndata,
     void *cbdata);
```

**IN** `status`

Status associated with the operation (handle)

**IN** `data`

Array of data returned (`pmix_pdata_t`)

**IN** `ndata`

Number of elements in the `data` array (`size_t`)

**IN** `cbdata`

Callback data passed to original API call (memory reference)

## Description

A callback function for calls to `PMIx_Lookup_nb`. The function will be called upon completion of the `PMIx_Lookup_nb` API with the *status* indicating the success or failure of the request. Any 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.

Note that the `pmix_pdata_t` structures will be released upon return from the callback function, so the receiver must copy/protect the data prior to returning if it needs to be retained.

## 9.8 Retrieval rules for published data

The retrieval rules for published data primarily revolve around enforcing data access permissions and range constraints. The datastore shall search its stored information for each specified key according to the following precedence logic:

1. If the requester specified the range, then the search shall be constrained to data where the publishing process falls within the specified range.
2. If the key of the stored information does not match the specified key, then the search will continue.
3. If the requester's identifier does not fall within the range specified by the publisher, then the search will continue.
4. If the publisher specified access permissions, the effective UID and GID of the requester shall be checked against those permissions, with the datastore rejecting the match if the requester fails to meet the requirements.
5. If all of the above checks pass, then the value is added to the information that is to be returned.

The status returned by the datastore shall be set to:

- `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.

In the case where 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, the datastore must return the `PMIX_ERR_NO_PERMISSIONS` error.

## Advice to users

Note that duplicate keys are allowed to exist on different ranges, and that ranges do overlap each other. Thus, if duplicate keys are published on overlapping ranges, it is possible for the datastore to successfully find multiple responses for a given key should publisher and requester specify sufficiently broad ranges. In this situation, the choice of resolving the duplication is left to the datastore implementation - e.g., it may return the first value found in its search, or the value corresponding to the most limited range of the found values, or it may choose to simply return an error.

Users are advised to avoid this ambiguity by careful selection of key values and ranges - e.g., by creating range-specific keys where necessary.

## 9.9 PMIx\_Unpublish

### Summary

Unpublish data posted by this process using the given keys.

### Format

PMIx v1.0

```
pmix_status_t
PMIx_Unpublish(char **keys,
               const pmix_info_t info[], size_t ninfo);
```

- IN keys**  
NULL-terminated array of keys (array of strings)
- IN info**  
Array of info structures (array of handles)
- IN ninfo**  
Number of elements in the *info* array (integer)

Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### 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\_GRPID** attributes of the client process that is requesting the operation.



## 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.

### Description

Unpublish data posted by this process using the given *keys*. The function will block until the data has been removed by the server (i.e., it is safe to publish that key again within the specified range). A value of **NULL** for the *keys* parameter instructs the server to remove all data published by this process.

By default, the range is assumed to be **PMIX\_RANGE\_SESSION**. Changes to the range, and any additional directives, can be provided in the *info* array.

## 9.10 PMIx\_Unpublish\_nb

### Summary

Nonblocking version of **PMIx\_Unpublish**.

### Format

C

```
pmix_status_t
PMIx_Unpublish_nb(char **keys,
                  const pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

**IN keys**  
NULL-terminated array of keys (array of strings)

**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)

A successful return indicates that the request is being processed and the result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API. The callback function, `cbfunc`, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the `cbfunc` will *not* be called.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 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_GRPID` attributes of the client process that is requesting the operation.

### 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.

## Description

Non-blocking form of the `PMIx_Unpublish` function. The callback function will be executed once the server confirms removal of the specified data. The `info` array must be maintained until the callback is provided.

## CHAPTER 10

# Event Notification

---

1 This chapter defines the PMIx event notification system. These interfaces are designed to support  
2 the reporting of events to/from clients and servers, and between library layers within a single  
3 process.

## 4 10.1 Notification and Management

5 PMIx event notification provides an asynchronous out-of-band mechanism for communicating  
6 events between application processes and/or elements of the SMS. Its uses span a wide range  
7 including fault notification, coordination between multiple programming libraries within a single  
8 process, and workflow orchestration for non-synchronous programming models. Events can be  
9 divided into two distinct classes:

- 10 • *Job-specific events* directly relate to a job executing within the session, such as a debugger  
11 attachment, process failure within a related job, or events generated by an application process.  
12 Events in this category are to be immediately delivered to the PMIx server library for relay to the  
13 related local processes.
- 14 • *Environment events* indirectly relate to a job but do not specifically target the job itself. This  
15 category includes SMS-generated events such as Error Check and Correction (ECC) errors,  
16 temperature excursions, and other non-job conditions that might directly affect a session's  
17 resources, but would never include an event generated by an application process. Note that  
18 although these do potentially impact the session's jobs, they are not directly tied to those jobs.  
19 Thus, events in this category are to be delivered to the PMIx server library only upon request.

20 Both SMS elements and applications can register for events of either type.

### ▼ Advice to PMIx library implementers ▼

21 Race conditions can cause the registration to come after events of possible interest (e.g., a memory  
22 ECC event that occurs after start of execution but prior to registration, or an application process  
23 generating an event prior to another process registering to receive it). SMS vendors are *requested* to  
24 cache environment events for some time to mitigate this situation, but are not *required* to do so.  
25 However, PMIx implementers are *required* to cache all events received by the PMIx server library  
26 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 by providing the relevant event handler names, if the user specified an  
26 event handler name when registering the corresponding event. Thus, users can specify that a given  
27 handler be executed before or after another handler should both handlers appear in an event chain  
28 (the ordering is ignored if the other handler isn't included). Note that ordering does not imply  
29 immediate relationships. For example, multiple handlers registered to be serviced after event  
30 handler *A* will all be executed after *A*, but are not guaranteed to be executed in any particular order  
31 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. Deregistration  
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.

## 17 10.1.1 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).

## 23 10.1.2 PMIx\_Register\_event\_handler

### 24 Summary

25 Register an event handler.

## Format

C

```
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);
```

C

### 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_reg_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:

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned. The result of the registration operation shall be returned in the provided callback function along with the assigned event handler identifier.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_ERR\_EVENT\_REGISTRATION** indicating that the registration has failed for an undetermined reason.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

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.evfirst" (bool)

8 Invoke this event handler before any other handlers.

9 **PMIX\_EVENT\_HDLR\_LAST** "pmix.evlast" (bool)

10 Invoke this event handler after all other handlers have been called.

11 **PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY** "pmix.evfirstcat" (bool)

12 Invoke this event handler before any other handlers in this category.

13 **PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY** "pmix.evlastcat" (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.evrage" (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.evaffected" (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 **10.1.3 Event registration constants**

26 **PMIX\_ERR\_EVENT\_REGISTRATION** Error in event registration.



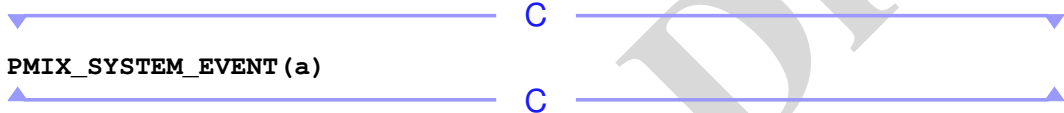
## 1 10.1.4 System events

- 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.

PMIx v2.2



14 **IN a**

15 Error constant to be checked (`pmix_status_t`)

16 Returns **true** if the provided values falls within the dedicated range of events for system event  
17 reporting.

## 18 10.1.5 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.

1 **PMIX\_EVENT\_HDLR\_APPEND** "pmix.evappend" (bool)  
 2 Append this handler to the precedence list within its category.

3 **PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrangle" (pmix\_data\_array\_t\*)  
 4 Array of **pmix\_proc\_t** defining range of event notification.

5 **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (pmix\_proc\_t)  
 6 The single process that was affected.

7 **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaffected" (pmix\_data\_array\_t\*)  
 8 Array of **pmix\_proc\_t** defining affected processes.

9 **PMIX\_EVENT\_NON\_DEFAULT** "pmix.evnondf" (bool)  
 10 Event is not to be delivered to default event handlers.

11 **PMIX\_EVENT\_RETURN\_OBJECT** "pmix.evobject" (void \*)  
 12 Object to be returned whenever the registered callback function **cbfunc** is invoked. The  
 13 object will only be returned to the process that registered it.

14 **PMIX\_EVENT\_DO\_NOT\_CACHE** "pmix.evnocache" (bool)  
 15 Instruct the PMIx server not to cache the event.

16 **PMIX\_EVENT\_PROXY** "pmix.evproxy" (pmix\_proc\_t\*)  
 17 PMIx server that sourced the event.

18 **PMIX\_EVENT\_TEXT\_MESSAGE** "pmix.evtext" (char\*)  
 19 Text message suitable for output by recipient - e.g., describing the cause of the event.

20 **PMIX\_EVENT\_TIMESTAMP** "pmix.evtstamp" (time\_t)  
 21 System time when the associated event occurred.

### 22 10.1.5.1 Fault tolerance event attributes

23 The following attributes may be used by the host environment when providing an event notification  
 24 as qualifiers indicating the action it intends to take in response to the event:

25 **PMIX\_EVENT\_TERMINATE\_SESSION** "pmix.evterm.sess" (bool)  
 26 The RM intends to terminate this session.

27 **PMIX\_EVENT\_TERMINATE\_JOB** "pmix.evterm.job" (bool)  
 28 The RM intends to terminate this job.

29 **PMIX\_EVENT\_TERMINATE\_NODE** "pmix.evterm.node" (bool)  
 30 The RM intends to terminate all processes on this node.

31 **PMIX\_EVENT\_TERMINATE\_PROC** "pmix.evterm.proc" (bool)  
 32 The RM intends to terminate just this process.

33 **PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evtimeout" (int)  
 34 The time in seconds before the RM will execute the indicated operation.

### 35 10.1.5.2 Hybrid programming event attributes

36 The following attributes may be used by programming models to coordinate their use of common  
 37 resources within a process in conjunction with the **PMIX\_OPENMP\_PARALLEL\_ENTERED** event:

38 **PMIX\_MODEL\_PHASE\_NAME** "pmix.mdl.phase" (char\*)  
 39 User-assigned name for a phase in the application execution (e.g., "cfd reduction").

40 **PMIX\_MODEL\_PHASE\_TYPE** "pmix.mdl.ptype" (char\*)  
 41 Type of phase being executed (e.g., "matrix multiply").

## 1 10.1.6 Notification Function

### 2 Summary

3 The `pmix_notification_fn_t` is called by PMIx to deliver notification of an event.

#### Advice to users

4 The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but  
5 different signature than the v2.0 Standard described below. The *ad hoc* v1.0 version was removed  
6 from the v2.0 Standard is not included in this document to avoid confusion.

PMIx v2.0

```
7 typedef void (*pmix_notification_fn_t)
8     (size_t evhdlr_registration_id,
9      pmix_status_t status,
10     const pmix_proc_t *source,
11     pmix_info_t info[], size_t ninfo,
12     pmix_info_t results[], size_t nresults,
13     pmix_event_notification_cbfunc_fn_t cbfunc,
14     void *cbdata);
```

15 **IN** `evhdlr_registration_id`  
16 Registration number of the handler being called (`size_t`)

17 **IN** `status`  
18 Status associated with the operation (`pmix_status_t`)

19 **IN** `source`  
20 Identifier of the process that generated the event (`pmix_proc_t`). If the source is the SMS,  
21 then the nspace will be empty and the rank will be `PMIX_RANK_UNDEF`

22 **IN** `info`  
23 Information describing the event (`pmix_info_t`). This argument will be `NULL` if no  
24 additional information was provided by the event generator.

25 **IN** `ninfo`  
26 Number of elements in the info array (`size_t`)

27 **IN** `results`  
28 Aggregated results from prior event handlers servicing this event (`pmix_info_t`). This  
29 argument will be `NULL` if this is the first handler servicing the event, or if no prior handlers  
30 provided results.

31 **IN** `nresults`  
32 Number of elements in the results array (`size_t`)

33 **IN** `cbfunc`  
34 `pmix_event_notification_cbfunc_fn_t` callback function to be executed upon  
35 completion of the handler's operation and prior to handler return (function reference).

## IN `cbdata`

Callback data to be passed to `cbfunc` (memory reference)

### Description

Note that different RMs may provide differing levels of support for event notification to application processes. Thus, the *info* array may be **NULL** or may contain detailed information of the event. It is the responsibility of the application to parse any provided info array for defined key-values if it so desires.

#### Advice to users

Possible uses of the *info* array include:

- for the host RM to alert the process as to planned actions, such as aborting the session, in response to the reported event
- provide a timeout for alternative action to occur, such as for the application to request an alternate response to the event

For example, the RM might alert the application to the failure of a node that resulted in termination of several processes, and indicate that the overall session will be aborted unless the application requests an alternative behavior in the next 5 seconds. The application then has time to respond with a checkpoint request, or a request to recover from the failure by obtaining replacement nodes and restarting from some earlier checkpoint.

Support for these options is left to the discretion of the host RM. Info keys are included in the common definitions above but may be augmented by environment vendors.

#### Advice to PMIx server hosts

On the server side, the notification function is used to inform the PMIx server library's host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the `pmix_server_notify_event_fn_t` function pointer, if provided.

## 10.1.7 `PMIx_Deregister_event_handler`

### Summary

Deregister an event handler.

## Format

C

```
pmix_status_t
PMIx_Deregister_event_handler(size_t evhdlr_ref,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata);
```

C

**IN** `evhdlr_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.

A successful return indicates that the request is being processed and the result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API. The callback function, `cbfunc`, is only called when `PMIX_SUCCESS` is returned.

- `PMIX_OPERATION_SUCCEEDED`, returned when the request was immediately processed successfully - the `cbfunc` will *not* be called.

The returned status code of `cbfunc` 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`).

## 10.1.8 PMIx\_Notify\_event

### Summary

Report an event for notification via any registered event handler.

## Format

C

```
pmix_status_t
PMIx_Notify_event (pmix_status_t status,
                  const pmix_proc_t *source,
                  pmix_data_range_t range,
                  pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

### IN status

Status code of the event ([pmix\\_status\\_t](#))

### IN source

Pointer to a [pmix\\_proc\\_t](#) identifying the original reporter of the event (handle)

### IN range

Range across which this notification shall be delivered ([pmix\\_data\\_range\\_t](#))

### IN info

Array of [pmix\\_info\\_t](#) structures containing any further info provided by the originator of the event (array of handles)

### IN ninfo

Number of elements in the *info* array ([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.

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned. Note that a successful call does *not* reflect the success or failure of delivering the event to any recipients.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, returned when the request was immediately processed successfully - the *cbfunc* will *not* be called.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Required Attributes

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

**PMIX\_EVENT\_NON\_DEFAULT** "pmix.evnondf" (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.evaffected" (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)

The RM intends to terminate just this process.

**PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evtimeout" (int)

The time in seconds before the RM will execute the indicated operation.

▲-----▲

## 1 Description

2 Report an event for notification via any registered event handler. This function can be called by any  
3 PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server  
4 calls this API to report events it detected itself so that the host SMS daemon distribute and handle  
5 them, and to pass events given to it by its host down to any attached client processes for processing.  
6 Examples might include notification of the failure of another process, detection of an impending  
7 node failure due to rising temperatures, or an intent to preempt the application. Events may be  
8 locally generated or come from anywhere in the system.

9 Host SMS daemons call the API to pass events down to its embedded PMIx server both for  
10 transmittal to local client processes and for the host's own internal processing where the host has  
11 registered its own event handlers. The PMIx server library is not allowed to echo any event given to  
12 it by its host via this API back to the host through the `pmix_server_notify_event_fn_t`  
13 server module function. The host is required to deliver the event to all PMIx servers where the  
14 targeted processes either are currently running, or (if they haven't started yet) might be running at  
15 some point in the future as the events are required to be cached by the PMIx server library.

16 Client application processes can call this function to notify the SMS and/or other application  
17 processes of an event it encountered. Note that processes are not constrained to report status values  
18 defined in the official PMIx standard — any integer value can be used. Thus, applications are free  
19 to define their own internal events and use the notification system for their own internal purposes.

▼-----▼

### Advice to users

20 The callback function will be called upon completion of the `notify_event` function's actions.  
21 At that time, any messages required for executing the operation (e.g., to send the notification to the  
22 local PMIx server) will have been queued, but may not yet have been transmitted. The caller is  
23 required to maintain the input data until the callback function has been executed — the sole purpose  
24 of the callback function is to indicate when the input data is no longer required.

▲-----▲



## 1 10.1.9 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.

```
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);
```

#### 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 10.1.9.1 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 11

# 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 11.1 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

## 1 11.2 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* ▼ C \_\_\_\_\_ ▼

6 **PMIX\_DATA\_BUFFER\_CREATE**(buffer);

▲ \_\_\_\_\_ 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* ▼ C \_\_\_\_\_ ▼

12 **PMIX\_DATA\_BUFFER\_RELEASE**(buffer);

▲ \_\_\_\_\_ 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* ▼ C \_\_\_\_\_ ▼

17 **PMIX\_DATA\_BUFFER\_CONSTRUCT**(buffer);

▲ \_\_\_\_\_ 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* ▼ C \_\_\_\_\_ ▼

22 **PMIX\_DATA\_BUFFER\_DESTRUCT**(buffer);

▲ \_\_\_\_\_ 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 **11.3 General Routines**

27 The following routines are provided to support internode transfers in heterogeneous environments.

28 **11.3.1 PMIx\_Data\_pack**

29 **Summary**

30 Pack one or more values of a specified type into a buffer, usually for transmission to another process.

## Format

C

```
pmix_status_t
PMIx_Data_pack(const pmix_proc_t *target,
               pmix_data_buffer_t *buffer,
               void *src, int32_t num_vals,
               pmix_data_type_t type);
```

C

### IN target

Pointer to a [pmix\\_proc\\_t](#) containing the nspace/rank of the process that will be unpacking the final buffer. A NULL value may be used to indicate that the target is based on the same PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

### IN buffer

Pointer to a [pmix\\_data\\_buffer\\_t](#) where the packed data is to be stored (handle)

### IN src

Pointer to a location where the data resides. Strings are to be passed as (char \*\*) — i.e., the caller must pass the address of the pointer to the string as the (void\*). This allows the caller to pass multiple strings in a single call. (memory reference)

### IN num\_vals

Number of elements pointed to by the *src* pointer. A string value is counted as a single value regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g., string arrays) should be contiguous, although the data pointed to need not be contiguous across array entries. ([int32\\_t](#))

### IN type

The type of the data to be packed ([pmix\\_data\\_type\\_t](#))

Returns PMIX\_SUCCESS or one of the following error codes when the condition described occurs:

[PMIX\\_ERR\\_BAD\\_PARAM](#) The provided buffer or src is **NULL**

[PMIX\\_ERR\\_UNKNOWN\\_DATA\\_TYPE](#) The specified data type is not known to this implementation

[PMIX\\_ERR\\_OUT\\_OF\\_RESOURCE](#) Not enough memory to support the operation

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Description

The pack function packs one or more values of a specified type into the specified buffer. The buffer must have already been initialized via the [PMIX\\_DATA\\_BUFFER\\_CREATE](#) or [PMIX\\_DATA\\_BUFFER\\_CONSTRUCT](#) macros — otherwise, [PMIx\\_Data\\_pack](#) will return an error. Providing an unsupported type flag will likewise be reported as an error.

Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may lose precision when unpacked by a non-homogeneous recipient. The [PMIx\\_Data\\_pack](#) function will do its best to deal with heterogeneity issues between the packer and unpacker in such cases.

1 Sending a number larger than can be handled by the recipient will return an error code (generated  
2 upon unpacking) — the error cannot be detected during packing.

3 The namespace of the intended recipient of the packed buffer (i.e., the process that will be  
4 unpacking it) is used solely to resolve any data type differences between PMI<sub>x</sub> versions. The  
5 recipient must, therefore, be known to the user prior to calling the pack function so that the PMI<sub>x</sub>  
6 library is aware of the version the recipient is using. Note that all processes in a given namespace  
7 are *required* to use the same PMI<sub>x</sub> version — thus, the caller must only know at least one process  
8 from the target’s namespace.

## 9 11.3.2 PMI<sub>x</sub>\_Data\_unpack

### 10 Summary

11 Unpack values from a `pmix_data_buffer_t`

### 12 Format

*PMI<sub>x</sub> v2.0*

C

13 `pmix_status_t`

```
14 PMIx_Data_unpack(const pmix_proc_t *source,  
15                  pmix_data_buffer_t *buffer, void *dest,  
16                  int32_t *max_num_values,  
17                  pmix_data_type_t type);
```

C

#### 19 IN `source`

20 Pointer to a `pmix_proc_t` structure containing the nspace/rank of the process that packed  
21 the provided buffer. A NULL value may be used to indicate that the source is based on the  
22 same PMI<sub>x</sub> version as the caller. Note that only the source’s nspace is relevant. (handle)

#### 23 IN `buffer`

24 A pointer to the buffer from which the value will be extracted. (handle)

#### 25 INOUT `dest`

26 A pointer to the memory location into which the data is to be stored. Note that these values  
27 will be stored contiguously in memory. For strings, this pointer must be to (`char**`) to provide  
28 a means of supporting multiple string operations. The unpack function will allocate memory  
29 for each string in the array - the caller must only provide adequate memory for the array of  
30 pointers. (`void*`)

#### 31 INOUT `max_num_values`

32 The number of values to be unpacked — upon completion, the parameter will be set to the  
33 actual number of values unpacked. In most cases, this should match the maximum number  
34 provided in the parameters — but in no case will it exceed the value of this parameter. Note  
35 that unpacking fewer values than are actually available will leave the buffer in an unpackable  
36 state — the function will return an error code to warn of this condition. (`int32_t`)

## IN type

The type of the data to be unpacked — must be one of the PMI<sub>x</sub> defined data types (`pmix_data_type_t`)

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

`PMIX_ERR_BAD_PARAM` The provided buffer or dest is `NULL`

`PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this implementation

`PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation

If none of the above return codes are appropriate, then an implementation must return either a general PMI<sub>x</sub> error code or an implementation defined error code as described in Section 3.1.1.

## Description

The `unpack` function unpacks the next value (or values) of a specified type from the given buffer.

The buffer must have already been initialized via an `PMIX_DATA_BUFFER_CREATE` or `PMIX_DATA_BUFFER_CONSTRUCT` call (and assumedly filled with some data) — otherwise, the `unpack_value` function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

Note that it is possible for the buffer to be corrupted and that PMI<sub>x</sub> will *think* there is a proper variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte field in a string array that so happens to have a value that matches the specified data type flag). Therefore, the data type error check is *not* completely safe.

Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer. It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the `unpack_ptr`.

Warning: The caller is responsible for providing adequate memory storage for the requested data. The user must provide a parameter indicating the maximum number of values that can be unpacked into the allocated memory. If more values exist in the buffer than can fit into the memory storage, then the function will unpack what it can fit into that location and return an error code indicating that the buffer was only partially unpacked.

Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may lose precision when unpacked by a non-homogeneous recipient. PMI<sub>x</sub> will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code generated upon unpacking — these errors cannot be detected during packing.

The namespace of the process that packed the buffer is used solely to resolve any data type differences between PMI<sub>x</sub> versions. The packer must, therefore, be known to the user prior to calling the `pack` function so that the PMI<sub>x</sub> 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 11.3.3 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 PMIx\_SUCCESS or one of the following error codes when the condition described occurs:

18 [PMIX\\_ERR\\_BAD\\_PARAM](#) The provided src or dest is **NULL**

19 [PMIX\\_ERR\\_UNKNOWN\\_DATA\\_TYPE](#) The specified data type is not known to this  
20 implementation

21 [PMIX\\_ERR\\_OUT\\_OF\\_RESOURCE](#) Not enough memory to support the operation

22 If none of the above return codes are appropriate, then an implementation must return either a  
23 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

#### 24 Description

25 Since registered data types can be complex structures, the system needs some way to know how to  
26 copy the data from one location to another (e.g., for storage in the registry). This function, which  
27 can call other copy functions to build up complex data types, defines the method for making a copy  
28 of the specified data type.

### 29 11.3.4 PMIx\_Data\_print

#### 30 Summary

31 Pretty-print a data value.



## Format

C

```
pmix_status_t
PMIx_Data_print(char **output, char *prefix,
                void *src, pmix_data_type_t type);
```

C

### IN output

The address of a pointer into which the address of the resulting output is to be stored.  
(char\*\*)

### IN prefix

String to be prepended to the resulting output (char\*)

### IN src

A pointer to the memory location of the data value to be printed (handle)

### IN type

The type of the data value to be printed — must be one of the PMIx defined data types.  
(pmix\_data\_type\_t)

Returns PMIX\_SUCCESS or one of the following error codes when the condition described occurs:

**PMIX\_ERR\_BAD\_PARAM** The provided data type is not recognized.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Description

Since registered data types can be complex structures, the system needs some way to know how to print them (i.e., convert them to a string representation). Primarily for debug purposes.

## 11.3.5 PMIx\_Data\_copy\_payload

### Summary

Copy a payload from one buffer to another

### Format

C

PMIx v2.0

```
pmix_status_t
PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
                       pmix_data_buffer_t *src);
```

C

### IN dest

Pointer to the destination [pmix\\_data\\_buffer\\_t](#) (handle)

### IN src

Pointer to the source [pmix\\_data\\_buffer\\_t](#) (handle)

Returns one of the following:

**PMIX\_SUCCESS** The data has been copied as requested

**PMIX\_ERR\_BAD\_PARAM** The src and dest `pmix_data_buffer_t` types do not match

**PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.

## Description

This function will append a copy of the payload in one buffer into another buffer. Note that this is *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

### 11.3.6 PMIx\_Data\_load

#### Summary

Load a buffer with the provided payload

#### Format

Provisional  
v4.1

`pmix_status_t`

```
PMIx_Data_load(pmix_data_buffer_t *dest,  
               pmix_byte_object_t *src);
```

**IN dest**

Pointer to the destination `pmix_data_buffer_t` (handle)

**IN src**

Pointer to the source `pmix_byte_object_t` (handle)

Returns one of the following:

**PMIX\_SUCCESS** The data has been loaded as requested

**PMIX\_ERR\_BAD\_PARAM** The `dest` structure pointer is **NULL**

**PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.

## Description

The load function allows the caller to transfer the contents of the `src` `pmix_byte_object_t` to the `dest` target buffer. If a payload already exists in the buffer, the function will "free" the existing data to release it, and then replace the data payload with the one provided by the caller.

#### Advice to users

The buffer must be allocated or constructed in advance - failing to do so will cause the load function to return an error code.

The caller is responsible for pre-packing the provided payload. For example, the load function cannot convert to network byte order any data contained in the provided payload.

## 1 11.3.7 PMIx\_Data\_unload

### 2 Summary

3 Unload a buffer into a byte object

### 4 Format

Provisional  
v4.1

```
pmix_status_t  
PMIx_Data_unload(pmix_data_buffer_t *src,  
                 pmix_byte_object_t *dest);
```

8 IN src

9 Pointer to the source `pmix_data_buffer_t` (handle)

10 IN dest

11 Pointer to the destination `pmix_byte_object_t` (handle)

12 Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

13 `PMIX_ERR_BAD_PARAM` The destination and/or source pointer is `NULL`

14 If none of the above return codes are appropriate, then an implementation must return either a  
15 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 16 Description

17 The unload function provides the caller with a pointer to the portion of the data payload within the  
18 buffer that has not yet been unpacked, along with the size of that region. Any portion of the payload  
19 that was previously unpacked using the `PMIx_Data_unpack` routine will be ignored. This  
20 allows the user to directly access the payload.

#### Advice to users

21 This is a destructive operation. While the payload returned in the destination

22 `pmix_byte_object_t` is undisturbed, the function will clear the `src`'s pointers to the payload.

23 Thus, the `src` and the payload are completely separated, leaving the caller able to free or destruct the  
24 `src`.

## 25 11.3.8 PMIx\_Data\_compress

### 26 Summary

27 Perform a lossless compression on the provided data

## Format

C

`bool`

```
PMIx_Data_compress(const uint8_t *inbytes, size_t size,  
                  uint8_t **outbytes, size_t *nbytes);
```

C

**IN** `inbytes`

Pointer to the source data (handle)

**IN** `size`

Number of bytes in the source data region (`size_t`)

**OUT** `outbytes`

Address where the pointer to the compressed data region is to be returned (handle)

**OUT** `nbytes`

Address where the number of bytes in the compressed data region is to be returned (handle)

Returns one of the following:

- **True** The data has been compressed as requested
- **False** The data has not been compressed

## Description

Compress the provided data block. Destination memory will be allocated if operation is successfully concluded. Caller is responsible for release of the allocated region. The input data block will remain unaltered.

Note: the compress function will return **False** if the operation would not result in a smaller data block.

## 11.3.9 PMIx\_Data\_decompress

### Summary

Decompress the provided data

### Format

*Provisional*  
v4.1

C

```
1 bool
2 PMIx_Data_decompress(const uint8_t *inbytes, size_t size,
3                       uint8_t **outbytes, size_t *nbytes,);
```

C

**OUT outbytes**

Address where the pointer to the decompressed data region is to be returned (handle)

**OUT nbytes**

Address where the number of bytes in the decompressed data region is to be returned (handle)

**IN inbytes**

Pointer to the source data (handle)

**IN size**

Number of bytes in the source data region (**size\_t**)

Returns one of the following:

- **True** The data has been decompressed as requested
- **False** The data has not been decompressed

**Description**

Decompress the provided data block. Destination memory will be allocated if operation is successfully concluded. Caller is responsible for release of the allocated region. The input data block will remain unaltered.

Only data compressed by the [PMIx\\_Data\\_compress](#) API can be decompressed by this function. Passing data that has not been compressed by [PMIx\\_Data\\_compress](#) will lead to unexpected and potentially catastrophic results.

## CHAPTER 12

# Process Management

---

1 This chapter defines functionality processes can use to abort processes, spawn processes, and  
2 determine the relative locality of local processes.

## 3 12.1 Abort

4 PMIx provides a dedicated API by which an application can request that specified processes be  
5 aborted by the system.

### 6 12.1.1 PMIx\_Abort

#### 7 Summary

8 Abort the specified processes

#### 9 Format

*PMIx v1.0*

```
10 pmix_status_t  
11 PMIx_Abort(int status, const char msg[],  
12           pmix_proc_t procs[], size_t nprocs)
```

#### 13 IN status

14 Error code to return to invoking environment (integer)

#### 15 IN msg

16 String message to be returned to user (string)

#### 17 IN procs

18 Array of `pmix_proc_t` structures (array of handles)

#### 19 IN nprocs

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

21 A successful return indicates that the requested processes are in a terminated state. Note that the  
22 function shall not return in this situation if the caller's own process was included in the request.

23 Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- **PMIX\_ERR\_PARAM\_VALUE\_NOT\_SUPPORTED** if the PMI<sub>x</sub> implementation and host environment support this API, but the request includes processes that the host environment cannot abort - e.g., if the request is to abort subsets of processes from a namespace, or processes outside of the caller's own namespace, and the host environment does not permit such operations. In this case, none of the specified processes will be terminated.

If none of the above return codes are appropriate, then an implementation must return either a general PMI<sub>x</sub> error code or an implementation defined error code as described in Section 3.1.1.

### Description

Request that the host resource manager print the provided message and abort the provided array of *procs*. A Unix or POSIX environment should handle the provided status as a return error code from the main program that launched the application. A **NULL** for the *procs* array indicates that all processes in the caller's namespace are to be aborted, including itself - this is the equivalent of passing a **pmix\_proc\_t** array element containing the caller's namespace and a rank value of **PMIX\_RANK\_WILDCARD**. While it is permitted for a caller to request abort of processes from namespaces other than its own, not all environments will support such requests. Passing a **NULL** *msg* parameter is allowed.

The function shall not return until the host environment has carried out the operation on the specified processes. If the caller is included in the array of targets, then the function will not return unless the host is unable to execute the operation.

### Advice to users

The response to this request is somewhat dependent on the specific RM and its configuration (e.g., some resource managers will not abort the application if the provided status is zero unless specifically configured to do so, some cannot abort subsets of processes in an application, and some may not permit termination of processes outside of the caller's own namespace), and thus lies outside the control of PMI<sub>x</sub> itself. However, the PMI<sub>x</sub> client library shall inform the RM of the request that the specified *procs* be aborted, regardless of the value of the provided status.

Note that race conditions caused by multiple processes calling **PMIx\_Abort** are left to the server implementation to resolve with regard to which status is returned and what messages (if any) are printed.

## 12.2 Process Creation

The **PMIx\_Spawn** commands spawn new processes and/or applications in the PMI<sub>x</sub> universe. This may include requests to extend the existing resource allocation or obtain a new one, depending upon provided and supported attributes.

## 1 12.2.1 PMIx\_Spawn

### 2 Summary

3 Spawn a new job.

### 4 Format

*PMIx v1.0*

C

```
5 pmix_status_t  
6 PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,  
7             const pmix_app_t apps[], size_t napps,  
8             char nspace[])
```

C

9 **IN** **job\_info**  
10 Array of info structures (array of handles)

11 **IN** **ninfo**  
12 Number of elements in the *job\_info* array (integer)

13 **IN** **apps**  
14 Array of **pmix\_app\_t** structures (array of handles)

15 **IN** **napps**  
16 Number of elements in the *apps* array (integer)

17 **OUT** **nspace**  
18 Namespace of the new job (string)

19 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### Required Attributes

20 PMIx libraries are not required to directly support any attributes for this function. However, any  
21 provided attributes must be passed to the host environment for processing.

22 Host environments are required to support the following attributes when present in either the  
23 *job\_info* or the *info* array of an element of the *apps* array:

24 **PMIX\_WDIR** "**pmix.wdir**" (**char\***)  
25 Working directory for spawned processes.

26 **PMIX\_SET\_SESSION\_CWD** "**pmix.ssn cwd**" (**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.



1 **PMIX\_HOSTFILE** "pmix.hostfile" (char\*)

2 Hostfile to use for spawned processes.

▲-----▲  
▼-----▼ **Optional Attributes** -----▼

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

4 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (char\*)

5 Hostfile containing hosts to add to existing allocation.

6 **PMIX\_ADD\_HOST** "pmix.addhost" (char\*)

7 Comma-delimited list of hosts to add to the allocation.

8 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (bool)

9 Preload executables onto nodes prior to executing launch procedure.

10 **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (char\*)

11 Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

12 **PMIX\_PERSONALITY** "pmix.pers" (char\*)

13 Name of personality corresponding to programming model used by application - supported  
14 values depend upon PMIx implementation.

15 **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (bool)

16 Display process mapping upon spawn.

17 **PMIX\_PPR** "pmix.ppr" (char\*)

18 Number of processes to spawn on each identified resource.

19 **PMIX\_MAPBY** "pmix.mapby" (char\*)

20 Process mapping policy - when accessed using **PMIx\_Get**, use the  
21 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
22 provided namespace. Supported values are launcher specific.

23 **PMIX\_RANKBY** "pmix.rankby" (char\*)

24 Process ranking policy - when accessed using **PMIx\_Get**, use the  
25 **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
26 provided namespace. Supported values are launcher specific.

27 **PMIX\_BINDTO** "pmix.bindto" (char\*)

28 Process binding policy - when accessed using **PMIx\_Get**, use the  
29 **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
30 provided namespace. Supported values are launcher specific.

31 **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)

32 Spawned process rank that is to receive any forwarded **stdin**.

33 **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\_CPUS\_PER\_PROC** "pmix.cpusperproc" (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.envar.set" (pmix_envar_t*)`  
5 Set the envar to the given value, overwriting any pre-existing one

6 `PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)`  
7 Unset the environment variable specified in the string.

8 `PMIX_ADD_ENVAR "pmix.envar.add" (pmix_envar_t*)`  
9 Add the environment variable, but do not overwrite any pre-existing one

10 `PMIX_PREPEND_ENVAR "pmix.envar.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.envar.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.envar.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          **PMIX\_COSPAWN\_APP** "pmix.cospawn" (bool)

17          Designated application is to be spawned as a disconnected job - i.e., the launcher shall not  
18          include the application in any of the job-level values (e.g., **PMIX\_RANK** within the job)  
19          provided to any other application process generated by the same spawn request. Typically  
20          used to cospawn debugger daemons alongside an application.

21          **PMIX\_SPAWN\_TOOL** "pmix.spwn.tool" (bool)

22          Indicate that the job being spawned is a tool.

23          **PMIX\_EVENT\_SILENT\_TERMINATION** "pmix.avsilentterm" (bool)

24          Do not generate an event when this job normally terminates.



## 25          **Description**

26          Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspc*  
27          parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the  
28          namespace returned. The *nspc* array must be at least of size one more than **PMIX\_MAX\_NSLEN**.

29          By default, the spawned processes will be PMIx "connected" to the parent process upon successful  
30          launch (see Section 12.3 for details). This includes that (a) the parent process will be given a copy  
31          of the new job's information so it can query job-level info without incurring any communication  
32          penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level  
33          info, and (c) both the parent process and members of the child job will receive notification of errors  
34          from processes in their combined assemblage.

## Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

## Advice to PMIx library implementers

Tools may utilize `PMIx_Spawn` to start intermediate launchers as described in Section 18.2.2. For times where the tool is not attached to a PMIx server, internal support for fork/exec of the specified applications would allow the tool to maintain a single code path for both the connected and disconnected cases. Inclusion of such support is recommended, but not required.

## 12.2.2 `PMIx_Spawn_nb`

### Summary

Nonblocking version of the `PMIx_Spawn` routine.

### Format

*PMIx v1.0*

```
pmix_status_t  
PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,  
              const pmix_app_t apps[], size_t napps,  
              pmix_spawn_cbfunc_t cbfunc, void *cbdata)
```

**IN** `job_info`  
Array of info structures (array of handles)

**IN** `ninfo`  
Number of elements in the `job_info` array (integer)

**IN** `apps`  
Array of `pmix_app_t` structures (array of handles)

**IN** `cbfunc`  
Callback function `pmix_spawn_cbfunc_t` (function reference)

**IN** `cbdata`  
Data to be passed to the callback function (memory reference)

A successful return indicates that the request is being processed and the result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API. The callback function, `cbfunc`, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 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.

Host environments are required to support the following attributes when present in either the *job\_info* or the *info* array of an element of the *apps* array:

**PMIX\_WDIR** "pmix.wdir" (char\*)

Working directory for spawned processes.

**PMIX\_SET\_SESSION\_CWD** "pmix.ssn cwd" (bool)

Set the current working directory to the session working directory assigned by the RM - can be assigned to the entire job (by including attribute in the *job\_info* array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.

**PMIX\_PREFIX** "pmix.prefix" (char\*)

Prefix to use for starting spawned processes - i.e., the directory where the executables can be found.

**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.

### Optional Attributes

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

**PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (char\*)

Hostfile containing hosts to add to existing allocation.

**PMIX\_ADD\_HOST** "pmix.addhost" (char\*)

Comma-delimited list of hosts to add to the allocation.

**PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (bool)

Preload executables onto nodes prior to executing launch procedure.

**PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (char\*)

Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

**PMIX\_PERSONALITY** "pmix.pers" (char\*)

1 Name of personality corresponding to programming model used by application - supported  
2 values depend upon PMIx implementation.

3 **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (bool)

4 Display process mapping upon spawn.

5 **PMIX\_PPR** "pmix.ppr" (char\*)

6 Number of processes to spawn on each identified resource.

7 **PMIX\_MAPBY** "pmix.mapby" (char\*)

8 Process mapping policy - when accessed using **PMIx\_Get**, use the  
9 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
10 provided namespace. Supported values are launcher specific.

11 **PMIX\_RANKBY** "pmix.rankby" (char\*)

12 Process ranking policy - when accessed using **PMIx\_Get**, use the  
13 **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
14 provided namespace. Supported values are launcher specific.

15 **PMIX\_BINDTO** "pmix.bindto" (char\*)

16 Process binding policy - when accessed using **PMIx\_Get**, use the  
17 **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
18 provided namespace. Supported values are launcher specific.

19 **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)

20 Spawned process rank that is to receive any forwarded **stdin**.

21 **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)

22 Tag **stdout/stderr** with the identity of the source process - can be assigned to the entire  
23 job (by including attribute in the *job\_info* array) or on a per-application basis in the *info*  
24 array for each **pmix\_app\_t**.

25 **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)

26 Timestamp output - can be assigned to the entire job (by including attribute in the *job\_info*  
27 array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.

28 **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)

29 Merge **stdout** and **stderr** streams - can be assigned to the entire job (by including  
30 attribute in the *job\_info* array) or on a per-application basis in the *info* array for each  
31 **pmix\_app\_t**.

32 **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)

33 Direct output (both **stdout** and **stderr**) into files of form "<filename>.rank" - can be  
34 assigned to the entire job (by including attribute in the *job\_info* array) or on a per-application  
35 basis in the *info* array for each **pmix\_app\_t**.

36 **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)

37 Mark the **argv** with the rank of the process.

1 **PMIX\_CPUS\_PER\_PROC** "pmix.cpusperproc" (uint32\_t)  
2     Number of PUs to assign to each rank - when accessed using **PMIx\_Get**, use the  
3 **PMIX\_RANK\_WILDCARD** value for the rank to discover the PUs/process assigned to the  
4 provided namespace.

5 **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)  
6     Do not place processes on the head node.

7 **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)  
8     Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a  
9 node.

10 **PMIX\_REPORT\_BINDINGS** "pmix.repbind" (bool)  
11     Report bindings of the individual processes.

12 **PMIX\_CPU\_LIST** "pmix.cpulist" (char\*)  
13     List of PUs to use for this job - when accessed using **PMIx\_Get**, use the  
14 **PMIX\_RANK\_WILDCARD** value for the rank to discover the PU list used for the provided  
15 namespace.

16 **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)  
17     Application supports recoverable operations.

18 **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)  
19     Application is continuous, all failed processes should be immediately restarted.

20 **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)  
21     Maximum number of times to restart a process - when accessed using **PMIx\_Get**, use the  
22 **PMIX\_RANK\_WILDCARD** value for the rank to discover the max restarts for the provided  
23 namespace.

24 **PMIX\_SET\_ENVAR** "pmix.envar.set" (pmix\_envar\_t\*)  
25     Set the envar to the given value, overwriting any pre-existing one

26 **PMIX\_UNSET\_ENVAR** "pmix.envar.unset" (char\*)  
27     Unset the environment variable specified in the string.

28 **PMIX\_ADD\_ENVAR** "pmix.envar.add" (pmix\_envar\_t\*)  
29     Add the environment variable, but do not overwrite any pre-existing one

30 **PMIX\_PREPEND\_ENVAR** "pmix.envar.prepnd" (pmix\_envar\_t\*)  
31     Prepend the given value to the specified environmental value using the given separator  
32 character, creating the variable if it doesn't already exist

33 **PMIX\_APPEND\_ENVAR** "pmix.envar.appnd" (pmix\_envar\_t\*)  
34     Append the given value to the specified environmental value using the given separator  
35 character, creating the variable if it doesn't already exist

36 **PMIX\_FIRST\_ENVAR** "pmix.envar.first" (pmix\_envar\_t\*)



1           Ensure the given value appears first in the specified envar using the separator character,  
2           creating the envar if it doesn't already exist

3     **PMIX\_ALLOC\_QUEUE** "pmix.alloc.queue" (char\*)

4           Name of the WLM queue to which the allocation request is to be directed, or the queue being  
5           referenced in a query.

6     **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (uint32\_t)

7           Total session time (in seconds) being requested in an allocation request.

8     **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (uint64\_t)

9           The number of nodes being requested in an allocation request.

10    **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)

11          Regular expression of the specific nodes being requested in an allocation request.

12    **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (uint64\_t)

13          Number of PUs being requested in an allocation request.

14    **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (char\*)

15          Regular expression of the number of PUs for each node being requested in an allocation  
16          request.

17    **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (char\*)

18          Regular expression of the specific PUs being requested in an allocation request.

19    **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (float)

20          Number of Megabytes[base2] of memory (per process) being requested in an allocation  
21          request.

22    **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (float)

23          Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
24          request.

25    **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (char\*)

26          Fabric quality of service level for the job being requested in an allocation request.

27    **PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (char\*)

28          Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

29    **PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (char\*)

30          ID string for the *fabric plane* to be used for the requested allocation.

31    **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (size\_t)

32          Number of endpoints to allocate per *process* in the job.

33    **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (size\_t)

34          Number of endpoints to allocate per *node* for the job.

35    **PMIX\_COSPAWN\_APP** "pmix.cospawn" (bool)

1 Designated application is to be spawned as a disconnected job - i.e., the launcher shall not  
2 include the application in any of the job-level values (e.g., **PMIX\_RANK** within the job)  
3 provided to any other application process generated by the same spawn request. Typically  
4 used to cospawn debugger daemons alongside an application.

5 **PMIX\_SPAWN\_TOOL** "pmix.spwn.tool" (bool)

6 Indicate that the job being spawned is a tool.

7 **PMIX\_EVENT\_SILENT\_TERMINATION** "pmix.evsilentterm" (bool)

8 Do not generate an event when this job normally terminates.



### 9 **Description**

10 Nonblocking version of the **PMIx\_Spawn** routine. The provided callback function will be  
11 executed upon successful start of *all* specified application processes.

#### 12 **Advice to users**

13 Behavior of individual resource managers may differ, but it is expected that failure of any  
14 application process to start will result in termination/cleanup of all processes in the newly spawned  
job and return of an error code to the caller.

## 15 **12.2.3 Spawn-specific constants**

16 In addition to the generic error constants, the following spawn-specific error constants may be  
17 returned by the spawn APIs:

18 **PMIX\_ERR\_JOB\_ALLOC\_FAILED** The job request could not be executed due to failure to  
19 obtain the specified allocation

20 **PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE** The specified application executable either  
21 could not be found, or lacks execution privileges.

22 **PMIX\_ERR\_JOB\_NO\_EXE\_SPECIFIED** The job request did not specify an executable.

23 **PMIX\_ERR\_JOB\_FAILED\_TO\_MAP** The launcher was unable to map the processes for the  
24 specified job request.

25 **PMIX\_ERR\_JOB\_FAILED\_TO\_LAUNCH** One or more processes in the job request failed to  
26 launch

## 1 12.2.4 Spawn attributes

2 Attributes used to describe [PMIx\\_Spawn](#) behavior - they are values passed to the [PMIx\\_Spawn](#)  
3 API and therefore are not accessed using the [PMIx\\_Get](#) APIs when used in that context. However,  
4 some of the attributes defined in this section can be provided by the host environment for other  
5 purposes - e.g., the host might provide the [PMIX\\_MAPBY](#) attribute in the job-related information so  
6 that an application can use [PMIx\\_Get](#) to discover the mapping used for determining process  
7 locations. Multi-use attributes and their respective access reference rank are denoted below.

8 **PMIX\_PERSONALITY** "pmix.pers" (char\*)

9 Name of personality corresponding to programming model used by application - supported  
10 values depend upon PMIx implementation.

11 **PMIX\_HOST** "pmix.host" (char\*)

12 Comma-delimited list of hosts to use for spawned processes.

13 **PMIX\_HOSTFILE** "pmix.hostfile" (char\*)

14 Hostfile to use for spawned processes.

15 **PMIX\_ADD\_HOST** "pmix.addhost" (char\*)

16 Comma-delimited list of hosts to add to the allocation.

17 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (char\*)

18 Hostfile containing hosts to add to existing allocation.

19 **PMIX\_PREFIX** "pmix.prefix" (char\*)

20 Prefix to use for starting spawned processes - i.e., the directory where the executables can be  
21 found.

22 **PMIX\_WDIR** "pmix.wdir" (char\*)

23 Working directory for spawned processes.

24 **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (bool)

25 Display process mapping upon spawn.

26 **PMIX\_PPR** "pmix.ppr" (char\*)

27 Number of processes to spawn on each identified resource.

28 **PMIX\_MAPBY** "pmix.mapby" (char\*)

29 Process mapping policy - when accessed using [PMIx\\_Get](#), use the  
30 [PMIX\\_RANK\\_WILDCARD](#) value for the rank to discover the mapping policy used for the  
31 provided namespace. Supported values are launcher specific.

32 **PMIX\_RANKBY** "pmix.rankby" (char\*)

33 Process ranking policy - when accessed using [PMIx\\_Get](#), use the  
34 [PMIX\\_RANK\\_WILDCARD](#) value for the rank to discover the ranking algorithm used for the  
35 provided namespace. Supported values are launcher specific.

36 **PMIX\_BINDTO** "pmix.bindto" (char\*)

37 Process binding policy - when accessed using [PMIx\\_Get](#), use the  
38 [PMIX\\_RANK\\_WILDCARD](#) value for the rank to discover the binding policy used for the  
39 provided namespace. Supported values are launcher specific.

40 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (bool)

41 Preload executables onto nodes prior to executing launch procedure.

42 **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.ssn cwd" (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\_CPUS\_PER\_PROC** "pmix.cputerproc" (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.repbinding" (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 13.4.3.

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 13.4.3. 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.envar.set`" (`pmix_envar_t*`)

16 Set the envar to the given value, overwriting any pre-existing one

17 **PMIX\_UNSET\_ENVAR** "`pmix.envar.unset`" (`char*`)

18 Unset the environment variable specified in the string.

19 **PMIX\_ADD\_ENVAR** "`pmix.envar.add`" (`pmix_envar_t*`)

20 Add the environment variable, but do not overwrite any pre-existing one

21 **PMIX\_PREPEND\_ENVAR** "`pmix.envar.prepnd`" (`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.envar.appnd`" (`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.envar.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

## 30 12.2.5 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*

C

```
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;
```

C

### 17 12.2.5.1 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

▼ `PMIX_APP_CREATE` C

3 **PMIX\_APP\_CREATE**(*m*, *n*)

▲ C

4 **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`)

8 **Free an app structure**

9 Release a `pmix_app_t` structure

*PMIx v4.0*

▼ C

10 **PMIX\_APP\_RELEASE**(*m*)

▲ C

11 **IN** *m*

Pointer to a `pmix_app_t` structure (handle)

13 **Free an app array**

14 Release an array of `pmix_app_t` structures

*PMIx v1.0*

▼ C

15 **PMIX\_APP\_FREE**(*m*, *n*)

▲ C

16 **IN** *m*

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

18 **IN** *n*

Number of structures in the array (`size_t`)

20 **Create the info array of application directives**

21 Create an array of `pmix_info_t` structures for passing application-level directives, updating the  
22 *ninfo* field of the `pmix_app_t` structure.

*PMIx v2.2*

▼ C

23 **PMIX\_APP\_INFO\_CREATE**(*m*, *n*)

▲ C

24 **IN** *m*

Pointer to the `pmix_app_t` structure (handle)

26 **IN** *n*

Number of directives to be allocated (`size_t`)

27



## 1 12.2.5.2 Spawn Callback Function

### 2 Summary

3 The `pmix_spawn_cbfunc_t` is used on the PMIx client side by `PMIx_Spawn_nb` and on the  
4 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_namespace_t nspace, void *cbdata);
```

8 **IN status**

9 Status associated with the operation (handle)

10 **IN nspace**

11 Namespace string (`pmix_namespace_t`)

12 **IN cbdata**

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

### 14 Description

15 The callback will be executed upon launch of the specified applications in `PMIx_Spawn_nb`, or  
16 upon failure to launch any of them.

17 The *status* of the callback will indicate whether or not the spawn succeeded. The *namespace* of the  
18 spawned processes will be returned, along with any provided callback data. Note that the returned  
19 *namespace* value will not be protected upon return from the callback function, so the receiver must  
20 copy it if it needs to be retained.

## 21 12.3 Connecting and Disconnecting Processes

22 This section defines functions to connect and disconnect processes in two or more separate PMIx  
23 namespaces. The PMIx definition of *connected* solely implies that the host environment should  
24 treat the failure of any process in the assemblage as a reportable event, taking action on the  
25 assemblage as if it were a single application. For example, if the environment defaults (in the  
26 absence of any application directives) to terminating an application upon failure of any process in  
27 that application, then the environment should terminate all processes in the connected assemblage  
28 upon failure of any member.

29 The host environment may choose to assign a new namespace to the connected assemblage and/or  
30 assign new ranks for its members for its own internal tracking purposes. However, it is not required  
31 to communicate such assignments to the participants (e.g., in response to an appropriate call to  
32 `PMIx_Query_info_nb`). The host environment is required to generate a  
33 `PMIX_ERR_PROC_TERM_WO_SYNC` event should any process in the assemblage terminate or  
34 call `PMIx_Finalize` without first *disconnecting* from the assemblage. If the job including the  
35 process is terminated as a result of that action, then the host environment is required to also  
36 generate the `PMIX_ERR_JOB_TERM_WO_SYNC` for all jobs that were terminated as a result.

### Advice to PMIx server hosts

1 The *connect* operation does not require the exchange of job-level information nor the inclusion of  
2 information posted by participating processes via `PMIx_Put`. Indeed, the callback function  
3 utilized in `pmix_server_connect_fn_t` cannot pass information back into the PMIx server  
4 library. However, host environments are advised that collecting such information at the  
5 participating daemons represents an optimization opportunity as participating processes are likely  
6 to request such information after the connect operation completes.

### Advice to users

7 Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation.  
8 While not explicitly prohibited, users are advised that a PMIx implementation or host environment  
9 may return an error in such cases.

10 Neither the PMIx implementation nor host environment are required to provide any tracking  
11 support for the assemblage. Thus, the application is responsible for maintaining the membership  
12 list of the assemblage.

## 13 12.3.1 PMIx\_Connect

### 14 Summary

15 Connect namespaces.

## Format

C

```
pmix_status_t
PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,
             const pmix_info_t info[], size_t ninfo)
```

C

**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)

Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### 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

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The function will return once all processes identified in *procs* have called either **PMIx\_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

A process can only engage in one connect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple connect 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 timeout constraints and other options available from the host RM.

### Advice to users

All processes engaged in a given **PMIx\_Connect** 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\_Connect** 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. 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.

## 12.3.2 PMIx\_Connect\_nb

### Summary

Nonblocking **PMIx\_Connect\_nb** routine.

## Format

C

```
pmix_status_t
PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,
               const pmix_info_t info[], size_t ninfo,
               pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

- 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)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 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

Nonblocking version of **PMIx\_Connect**. The callback function is called once all processes identified in *procs* have called either **PMIx\_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes. See the advice provided in the description for **PMIx\_Connect** for more information.

## 12.3.3 PMIx\_Disconnect

### Summary

Disconnect a previously connected set of processes.

### Format

PMIx v1.0

```
pmix_status_t
PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
                const pmix_info_t info[], size_t ninfo);
```

**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)

Returns PMIX\_SUCCESS or one of the following error codes when the condition described occurs:

- 1 • the **PMIX\_ERR\_INVALID\_OPERATION** error indicating that the specified set of *procs* was not  
2 previously *connected* via a call to **PMIx\_Connect** or its non-blocking form.

3 If none of the above return codes are appropriate, then an implementation must return either a  
4 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Required Attributes

5 PMIx libraries are not required to directly support any attributes for this function. However, any  
6 provided attributes must be passed to the host SMS daemon for processing.

### 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 Disconnect a previously connected set of processes. The function will return once all processes  
16 identified in *procs* have called either **PMIx\_Disconnect** or its non-blocking version, *and* the  
17 host environment has completed any required supporting operations.  
18

19 A process can only engage in one disconnect operation involving the identical *procs* array at a time.  
20 However, a process can be simultaneously engaged in multiple disconnect operations, each  
21 involving a different *procs* array.

22 As in the case of the **PMIx\_Fence** operation, the *info* array can be used to pass user-level  
23 directives regarding the algorithm to be used for any collective operation involved in the operation,  
24 timeout constraints, and other options available from the host RM.

### Advice to users

25 All processes engaged in a given **PMIx\_Disconnect** operation must provide the identical *procs*  
26 array as ordering of entries in the array and the method by which those processes are identified  
27 (e.g., use of **PMIX\_RANK\_WILDCARD** versus listing the individual processes) *may* impact the host  
28 environment's algorithm for uniquely identifying an operation.

## Advice to PMIx library implementers

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

## Advice to PMIx server hosts

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

### 8 12.3.4 PMIx\_Disconnect\_nb

#### 9 Summary

10 Nonblocking **PMIx\_Disconnect** routine.

#### 11 Format

*PMIx v1.0*

```
12 pmix_status_t  
13 PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,  
14                   const pmix_info_t info[], size_t ninfo,  
15                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

16 **IN procs**  
17 Array of proc structures (array of handles)

18 **IN nprocs**  
19 Number of elements in the *procs* array (integer)

20 **IN info**  
21 Array of info structures (array of handles)

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

24 **IN cbfunc**  
25 Callback function **pmix\_op\_cbfunc\_t** (function reference)

26 **IN cbdata**  
27 Data to be passed to the callback function (memory reference)



1 A successful return indicates that the request is being processed and the result will be returned in  
2 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
3 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

4 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- 5 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
6 returned *success* - the *cbfunc* will *not* be called

7 If none of the above return codes are appropriate, then an implementation must return either a  
8 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Required Attributes

9 PMIx libraries are not required to directly support any attributes for this function. However, any  
10 provided attributes must be passed to the host SMS daemon for processing.

### Optional Attributes

11 The following attributes are optional for PMIx implementations:

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

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

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

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

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

## Description

19 Nonblocking **PMIx\_Disconnect** routine. The callback function is called either:

- 21 • to return the **PMIX\_ERR\_INVALID\_OPERATION** error indicating that the specified set of  
22 *procs* was not previously *connected* via a call to **PMIx\_Connect** or its non-blocking form;
- 23 • to return a PMIx error constant indicating that the operation failed; or
- 24 • once all processes identified in *procs* have called either **PMIx\_Disconnect\_nb** or its  
25 blocking version, *and* the host environment has completed any required supporting operations.

26 See the advice provided in the description for **PMIx\_Disconnect** for more information.

## 1 12.4 Process Locality

2 The relative locality of processes is often used to optimize their interactions with the hardware and  
3 other processes. PMIx provides a means by which the host environment can communicate the  
4 locality of a given process using the `PMIx_server_generate_locality_string` to  
5 generate an abstracted representation of that value. This provides a human-readable format and  
6 allows the client to parse the locality string with a method of its choice that may differ from the one  
7 used by the server that generated it.

8 There are times, however, when relative locality and other PMIx-provided information doesn't  
9 include some element required by the application. In these instances, the application may need  
10 access to the full description of the local hardware topology. PMIx does not itself generate such  
11 descriptions - there are multiple third-party libraries that fulfill that role. Instead, PMIx offers an  
12 abstraction method by which users can obtain a pointer to the description. This transparently  
13 enables support for different methods of sharing the topology between the host environment (which  
14 may well have already generated it prior to local start of application processes) and the clients - e.g.,  
15 through passing of a shared memory region.

### 16 12.4.1 PMIx\_Load\_topology

#### 17 Summary

18 Load the local hardware topology description

#### 19 Format

*PMIx v4.0*

C

20 `pmix_status_t`

21 `PMIx_Load_topology(pmix_topology_t *topo);`

C

#### 22 INOUT topo

23 Address of a `pmix_topology_t` structure where the topology information is to be loaded  
24 (handle)

25 A successful return indicates that the *topo* was successfully loaded.

26 Returns `PMIX_SUCCESS` or a negative value indicating the error.

#### 27 Description

28 Obtain a pointer to the topology description of the local node. If the *source* field of the provided  
29 `pmix_topology_t` is set, then the PMIx library must return a description from the specified  
30 implementation or else indicate that the implementation is not available by returning the  
31 `PMIX_ERR_NOT_SUPPORTED` error constant.

32 The returned pointer may point to a shared memory region or an actual instance of the topology  
33 description. In either case, the description shall be treated as a "read-only" object - attempts to  
34 modify the object are likely to fail and return an error. The PMIx library is responsible for  
35 performing any required cleanup when the client library finalizes.

## Advice to users

It is the responsibility of the user to ensure that the *topo* argument is properly initialized prior to calling this API, and to check the returned *source* to verify that the returned topology description is compatible with the user's code.

### 12.4.2 PMIx\_Get\_relative\_locality

#### Summary

Get the relative locality of two local processes given their locality strings.

#### Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Get_relative_locality(const char *locality1,
                          const char *locality2,
                          pmix_locality_t *locality);
```

C

#### IN locality1

String returned by the [PMIx\\_server\\_generate\\_locality\\_string](#) API (handle)

#### IN locality2

String returned by the [PMIx\\_server\\_generate\\_locality\\_string](#) API (handle)

#### INOUT locality

Location where the relative locality bitmask is to be constructed (memory reference)

A successful return indicates that the *locality* was successfully loaded.

Returns [PMIX\\_SUCCESS](#) or a negative value indicating the error.

#### Description

Parse the locality strings of two processes (as returned by [PMIx\\_Get](#) using the [PMIX\\_LOCALITY\\_STRING](#) key) and set the appropriate [pmix\\_locality\\_t](#) locality bits in the provided memory location.

#### 12.4.2.1 Topology description

The [pmix\\_topology\\_t](#) structure contains a (case-insensitive) string identifying the source of the topology (e.g., "hwloc") and a pointer to the corresponding implementation-specific topology description.

PMIx v4.0

C

```
typedef struct pmix_topology {
    char *source;
    void *topology;
} pmix_topoology_t;
```

C

## 1 12.4.2.2 Topology support macros

2 The following macros support the `pmix_topology_t` structure.

### 3 Initialize the topology structure

4 Initialize the `pmix_topology_t` fields to `NULL`

*PMIx v4.0*

▼ `PMIX_TOPOLOGY_CONSTRUCT` C

5 `PMIX_TOPOLOGY_CONSTRUCT` (m)

▲ `PMIX_TOPOLOGY_CONSTRUCT` C

6 **IN** m

7 Pointer to the structure to be initialized (pointer to `pmix_topology_t`)

### 8 Destruct the topology structure

9 Destruct the `pmix_topology_t` fields

*PMIx v4.0*

▼ `PMIX_TOPOLOGY_DESTRUCT` C

10 `PMIX_TOPOLOGY_DESTRUCT` (m)

▲ `PMIX_TOPOLOGY_DESTRUCT` C

11 **IN** m

12 Pointer to the structure to be destructed (pointer to `pmix_topology_t`)

### 13 Create a topology array

14 Allocate and initialize a `pmix_topology_t` array.

*PMIx v4.0*

▼ `PMIX_TOPOLOGY_CREATE` C

15 `PMIX_TOPOLOGY_CREATE` (m, n)

▲ `PMIX_TOPOLOGY_CREATE` C

16 **INOUT** m

17 Address where the pointer to the array of `pmix_topology_t` structures shall be stored  
18 (handle)

19 **IN** n

20 Number of structures to be allocated (size\_t)

### 21 Release a topology array

22 Release a `pmix_topology_t` array.

*PMIx v4.0*

▼ `PMIX_TOPOLOGY_FREE` C

23 `PMIX_TOPOLOGY_FREE` (m, n)

▲ `PMIX_TOPOLOGY_FREE` C

24 **INOUT** m

25 Address of the array of `pmix_topology_t` structures to be released (handle)

26 **IN** n

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

### 1 12.4.2.3 Relative locality of two processes

2 The `pmix_locality_t` datatype is a `uint16_t` bitmask that defines the relative locality of  
3 two processes on a node. The following constants represent specific bits in the mask and can be  
4 used to test a locality value using standard bit-test methods.

- 5 **PMIX\_LOCALITY\_UNKNOWN** All bits are set to zero, indicating that the relative locality of the  
6 two processes is unknown
- 7 **PMIX\_LOCALITY\_NONLOCAL** The two processes do not share any common locations
- 8 **PMIX\_LOCALITY\_SHARE\_HWTHREAD** The two processes share at least one hardware thread
- 9 **PMIX\_LOCALITY\_SHARE\_CORE** The two processes share at least one core
- 10 **PMIX\_LOCALITY\_SHARE\_L1CACHE** The two processes share at least an L1 cache
- 11 **PMIX\_LOCALITY\_SHARE\_L2CACHE** The two processes share at least an L2 cache
- 12 **PMIX\_LOCALITY\_SHARE\_L3CACHE** The two processes share at least an L3 cache
- 13 **PMIX\_LOCALITY\_SHARE\_PACKAGE** The two processes share at least a package
- 14 **PMIX\_LOCALITY\_SHARE\_NUMA** The two processes share at least one Non-Uniform  
15 Memory Access (NUMA) region
- 16 **PMIX\_LOCALITY\_SHARE\_NODE** The two processes are executing on the same node

17 Implementers and vendors may choose to extend these definitions as needed to describe a particular  
18 system.

### 19 12.4.2.4 Locality keys

20 **PMIX\_LOCALITY\_STRING** "pmix.locstr" (`char*`)

21 String describing a process's bound location - referenced using the process's rank. The string  
22 is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon. The  
23 remainder of the string represents the corresponding locality as expressed by the underlying  
24 implementation. The entire string must be passed to `PMIx_Get_relative_locality`  
25 for processing. Note that hosts are only required to provide locality strings for local client  
26 processes - thus, a call to `PMIx_Get` for the locality string of a process that returns  
27 `PMIX_ERR_NOT_FOUND` indicates that the process is not executing on the same node.

## 28 12.4.3 PMIx\_Parse\_cpuset\_string

### 29 Summary

30 Parse the PU binding bitmap from its string representation.

### 31 Format

*PMIx v4.0*

C

32 `pmix_status_t`  
33 `PMIx_Parse_cpuset_string(const char *cpuset_string,`  
34 `pmix_cpuset_t *cpuset);`

**IN** `cpuset_string`

String returned by the `PMIx_server_generate_cpuset_string` API (handle)

**INOUT** `cpuset`

Address of an object where the bitmap is to be stored (memory reference)

A successful return indicates that the `cpuset` was successfully loaded.

Returns `PMIX_SUCCESS` or a negative value indicating the error.

**Description**

Parse the string representation of the binding bitmap (as returned by `PMIx_Get` using the `PMIX_CPUBIND` key) and set the appropriate PU binding location information in the provided memory location.

**12.4.4** `PMIx_Get_cpuset`**Summary**

Get the PU binding bitmap of the current process.

**Format**

*PMIx v4.0*

`pmix_status_t`

```
PMIx_Get_cpuset (pmix_cpuset_t *cpuset, pmix_bind_envelope_t ref);
```

**INOUT** `cpuset`

Address of an object where the bitmap is to be stored (memory reference)

**IN** `ref`

The binding envelope to be considered when formulating the bitmap  
(`pmix_bind_envelope_t`)

A successful return indicates that the `cpuset` was successfully loaded.

Returns `PMIX_SUCCESS` or a negative value indicating the error.

**Description**

Obtain and set the appropriate PU binding location information in the provided memory location based on the specified binding envelope.

**12.4.4.1** `Binding envelope`

*PMIx v4.0*

The `pmix_bind_envelope_t` data type defines the envelope of threads within a possibly multi-threaded process that are to be considered when getting the `cpuset` associated with the process. Valid values include:

**PMIX\_CPUBIND\_PROCESS** Use the location of all threads in the possibly multi-threaded process.

**PMIX\_CPUBIND\_THREAD** Use only the location of the thread calling the API.

## 1 12.4.5 PMIx\_Compute\_distances

### 2 Summary

3 Compute distances from specified process location to local devices.

### 4 *PMIx v4.0* Format

C

```
5 pmix_status_t
6 PMIx_Compute_distances(pmix_topology_t *topo,
7                        pmix_cpuset_t *cpuset,
8                        pmix_info_t info[], size_t ninfo[],
9                        pmix_device_distance_t *distances[],
10                       size_t *ndist);
```

C

#### 11 IN topo

12 Pointer to the topology description of the node where the process is located (**NULL** indicates  
13 the local node) ([pmix\\_topology\\_t](#))

#### 14 IN cpuset

15 Pointer to the location of the process ([pmix\\_cpuset\\_t](#))

#### 16 IN info

17 Array of [pmix\\_info\\_t](#) describing the devices whose distance is to be computed (handle)

#### 18 IN ninfo

19 Number of elements in *info* (integer)

#### 20 INOUT distances

21 Pointer to an address where the array of [pmix\\_device\\_distance\\_t](#) structures  
22 containing the distances from the caller to the specified devices is to be returned (handle)

#### 23 INOUT ndist

24 Pointer to an address where the number of elements in the *distances* array is to be returned  
25 (handle)

26 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### 27 Description

28 Both the minimum and maximum distance fields in the elements of the array shall be filled with the  
29 respective distances between the current process location and the types of devices or specific device  
30 identified in the *info* directives. In the absence of directives, distances to all supported device types  
31 shall be returned.

### Advice to users

32 A process whose threads are not all bound to the same location may return inconsistent results from  
33 calls to this API by different threads if the **PMIX\_CPUBIND\_THREAD** binding envelope was used  
34 when generating the *cpuset*.

## 1 12.4.6 PMIx\_Compute\_distances\_nb

### 2 Summary

3 Compute distances from specified process location to local devices.

### 4 Format

PMIx v4.0

C

```
5 pmix_status_t
6 PMIx_Compute_distances_nb(pmix_topology_t *topo,
7                           pmix_cpuset_t *cpuset,
8                           pmix_info_t info[], size_t ninfo[],
9                           pmix_device_dist_cbfunc_t cbfunc,
10                          void *cbdata);
```

C

#### 11 IN topo

12 Pointer to the topology description of the node where the process is located (**NULL** indicates  
13 the local node) ([pmix\\_topology\\_t](#))

#### 14 IN cpuset

15 Pointer to the location of the process ([pmix\\_cpuset\\_t](#))

#### 16 IN info

17 Array of [pmix\\_info\\_t](#) describing the devices whose distance is to be computed (handle)

#### 18 IN ninfo

19 Number of elements in *info* (integer)

#### 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 A successful return indicates that the request is being processed and the result will be returned in  
25 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
26 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

27 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- 28 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed  
29 successfully - the *cbfunc* will *not* be called.

30 If none of the above return codes are appropriate, then an implementation must return either a  
31 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 32 Description

33 Non-blocking form of the [PMIx\\_Compute\\_distances](#) API.



## 1 12.4.7 Device Distance Callback Function

### 2 Summary

3 The `pmix_device_dist_cbfnc_t` is used to return an array of device distances.

```
4 typedef void (*pmix_device_dist_cbfnc_t)
5     (pmix_status_t status,
6      pmix_device_distance_t *dist,
7      size_t ndist,
8      void *cbdata,
9      pmix_release_cbfnc_t release_fn,
10     void *release_cbdata);
```

#### 11 IN status

12 Status associated with the operation (`pmix_status_t`)

#### 13 IN dist

14 Array of `pmix_device_distance_t` returned by the operation (pointer)

#### 15 IN ndist

16 Number of elements in the *dist* array (`size_t`)

#### 17 IN cbdata

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

#### 19 IN release\_fn

20 Function to be called when done with the *dist* data (function pointer)

#### 21 IN release\_cbdata

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

### 23 Description

24 The *status* indicates if requested data was found or not. The array of  
25 `pmix_device_distance_t` will contain the distance information.

## 26 12.4.8 Device type

27 The `pmix_device_type_t` is a `uint64_t` bitmask for identifying the type(s) whose  
28 distances are being requested, or the type of a specific device being referenced (e.g., in a  
29 `pmix_device_distance_t` object).

*PMIx v1.0*

```
1 typedef uint16_t pmix_device_type_t;
```

2 The following constants can be used to set a variable of the type `pmix_device_type_t`.

3 **PMIX\_DEVTYPE\_UNKNOWN** The device is of an unknown type - will not be included in  
4 returned device distances.

5 **PMIX\_DEVTYPE\_BLOCK** Operating system block device, or non-volatile memory device  
6 (e.g., "sda" or "dax2.0" on Linux).

7 **PMIX\_DEVTYPE\_GPU** Operating system Graphics Processing Unit (GPU) device (e.g.,  
8 "card0" for a Linux Direct Rendering Manager (DRM) device).

9 **PMIX\_DEVTYPE\_NETWORK** Operating system network device (e.g., the "eth0" interface on  
10 Linux).

11 **PMIX\_DEVTYPE\_OPENFABRICS** Operating system OpenFabrics device (e.g., an "mlx4\_0"  
12 InfiniBand Host Channel Adapter (HCA), or "hfi1\_0" Omni-Path interface on Linux).

13 **PMIX\_DEVTYPE\_DMA** Operating system Direct Memory Access (DMA) engine device (e.g.,  
14 the "dma0chan0" DMA channel on Linux).

15 **PMIX\_DEVTYPE\_COPROC** Operating system co-processor device (e.g., "mic0" for a Xeon Phi  
16 on Linux, "opencl0d0" for an OpenCL device, or "cuda0" for a Compute Unified Device  
17 Architecture (CUDA) device).

## 18 12.4.9 Device Distance Structure

19 The `pmix_device_distance_t` structure contains the minimum and maximum relative  
20 distance from the caller to a given device.

*PMIx v4.0*

```
21 typedef struct pmix_device_distance {  
22     char *uuid;  
23     char *osname;  
24     pmix_device_type_t type;  
25     uint16_t mindist;  
26     uint16_t maxdist;  
27 } pmix_device_distance_t;
```

28 The *uuid* is a string identifier guaranteed to be unique within the cluster and is typically assembled  
29 from discovered device attributes (e.g., the Internet Protocol (IP) address of the device). The  
30 *osname* is the local operating system name of the device and is only unique to that node.

31 The two distance fields provide the minimum and maximum relative distance to the device from the  
32 specified location of the process, expressed as a 16-bit integer value where a smaller number  
33 indicates that this device is closer to the process than a device with a larger distance value. Note

1 that relative distance values are not necessarily correlated to a physical property - e.g., a device at  
2 twice the distance from another device does not necessarily have twice the latency for  
3 communication with it.

4 Relative distances only apply to similar devices and cannot be used to compare devices of different  
5 types. Both minimum and maximum distances are provided to support cases where the process may  
6 be bound to more than one location, and the locations are at different distances from the device.

7 A relative distance value of `UINT16_MAX` indicates that the distance from the process to the  
8 device could not be provided. This may be due to lack of available information (e.g., the PMIx  
9 library not having access to device locations) or other factors.

## 10 12.4.10 Device distance support macros

11 The following macros are provided to support the `pmix_device_distance_t` structure.

### 12 Initialize the device distance structure

13 Initialize the `pmix_device_distance_t` fields.

*PMIx v4.0*

▼ C ————— ▼

14 **PMIX\_DEVICE\_DIST\_CONSTRUCT** (m)

▲ C ————— ▲

15 **IN** m

16 Pointer to the structure to be initialized (pointer to `pmix_device_distance_t`)

### 17 Destruct the device distance structure

18 Destruct the `pmix_device_distance_t` fields.

*PMIx v4.0*

▼ C ————— ▼

19 **PMIX\_DEVICE\_DIST\_DESTRUCT** (m)

▲ C ————— ▲

20 **IN** m

21 Pointer to the structure to be destructed (pointer to `pmix_device_distance_t`)

### 22 Create an device distance array

23 Allocate and initialize a `pmix_device_distance_t` array.

*PMIx v4.0*

▼ C ————— ▼

24 **PMIX\_DEVICE\_DIST\_CREATE** (m, n)

▲ C ————— ▲

25 **INOUT** m

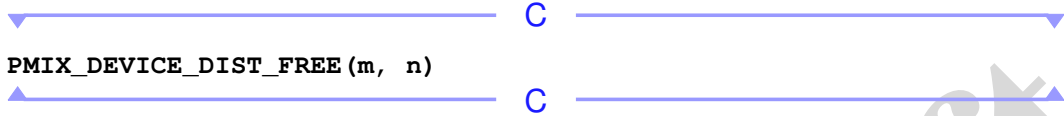
26 Address where the pointer to the array of `pmix_device_distance_t` structures shall be  
27 stored (handle)

28 **IN** n

29 Number of structures to be allocated (`size_t`)

1 **Release an device distance array**

2 Release an array of `pmix_device_distance_t` structures.



8 **12.4.11 Device distance attributes**

9 The following attributes can be used to retrieve device distances from the PMIx data store. Note  
10 that distances stored by the host environment are based on the process location at the time of start  
11 of execution and may not reflect changes to location imposed by the process itself.

12 **PMIX\_DEVICE\_DISTANCES** "pmix.dev.dist" (`pmix_data_array_t`)

13 Return an array of `pmix_device_distance_t` containing the minimum and maximum  
14 distances of the given process location to all devices of the specified type on the local node.

15 **PMIX\_DEVICE\_TYPE** "pmix.dev.type" (`pmix_device_type_t`)

16 Bitmask specifying the type(s) of device(s) whose information is being requested. Only used  
17 as a directive/qualifier.

18 **PMIX\_DEVICE\_ID** "pmix.dev.id" (`string`)

19 System-wide Universally Unique Identifier (UUID) or node-local Operating System (OS)  
20 name of a particular device.

## CHAPTER 13

# 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.

## 13.1 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.

### 13.1.1 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**  
 2       Allocation directive ([pmix\\_alloc\\_directive\\_t](#))  
 3       **IN info**  
 4       Array of [pmix\\_info\\_t](#) structures (array of handles)  
 5       **IN ninfo**  
 6       Number of elements in the *info* array (integer)  
 7       **INOUT results**  
 8       Address where a pointer to an array of [pmix\\_info\\_t](#) containing the results of the request  
 9       can be returned (memory reference)  
 10       **INOUT nresults**  
 11       Address where the number of elements in *results* can be returned (handle)

Returns [PMIX\\_SUCCESS](#) or a negative value indicating the error.

### Required Attributes

13       PMIx libraries are not required to directly support any attributes for this function. However, any  
 14       provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
 15       *required* to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process making  
 16       the request.

17       Host environments that implement support for this operation are required to support the following  
 18       attributes:

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\\_NUM\\_NODES](#) "pmix.alloc.nnodes" (uint64\_t)**  
 23       The number of nodes being requested in an allocation request.

24       **[PMIX\\_ALLOC\\_NUM\\_CPUS](#) "pmix.alloc.ncpus" (uint64\_t)**  
 25       Number of PUs being requested in an allocation request.

26       **[PMIX\\_ALLOC\\_TIME](#) "pmix.alloc.time" (uint32\_t)**  
 27       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 **13.1.2 PMIx\_Allocation\_request\_nb**

### 26 **Summary**

27 Request an allocation operation from the host resource manager.



## Format

C

```
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);
```

C

### IN directive

Allocation directive ([pmix\\_alloc\\_directive\\_t](#))

### IN info

Array of [pmix\\_info\\_t](#) structures (array of handles)

### IN ninfo

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

### IN cbfunc

Callback function [pmix\\_info\\_cbfunc\\_t](#) (function reference)

### IN cbdata

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

Returns one of the following:

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when [PMIX\\_SUCCESS](#) is returned.

Returns [PMIX\\_SUCCESS](#) or one of the following error codes when the condition described occurs:

- [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section [3.1.1](#).

## 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](#))

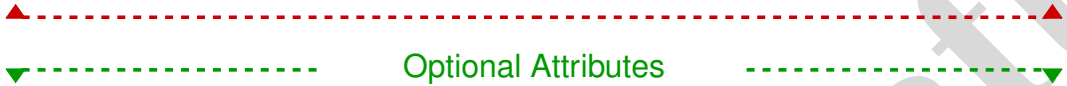
1 The number of nodes being requested in an allocation request.

2 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (uint64\_t)

3 Number of PUs being requested in an allocation request.

4 **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (uint32\_t)

5 Total session time (in seconds) being requested in an allocation request.



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

7 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)

8 Regular expression of the specific nodes being requested in an allocation request.

9 **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (char\*)

10 Regular expression of the number of PUs for each node being requested in an allocation  
11 request.

12 **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (char\*)

13 Regular expression of the specific PUs being requested in an allocation request.

14 **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (float)

15 Number of Megabytes[base2] of memory (per process) being requested in an allocation  
16 request.

17 **PMIX\_ALLOC\_FABRIC** "pmix.alloc.net" (array)

18 Array of **pmix\_info\_t** describing requested fabric resources. This must include at least:  
19 **PMIX\_ALLOC\_FABRIC\_ID**, **PMIX\_ALLOC\_FABRIC\_TYPE**, and  
20 **PMIX\_ALLOC\_FABRIC\_ENDPTS**, plus whatever other descriptors are desired.

21 **PMIX\_ALLOC\_FABRIC\_ID** "pmix.alloc.netid" (char\*)

22 The key to be used when accessing this requested fabric allocation. The fabric allocation  
23 will be returned/stored as a **pmix\_data\_array\_t** of **pmix\_info\_t** whose first  
24 element is composed of this key and the allocated resource description. The type of the  
25 included value depends upon the fabric support. For example, a TCP allocation might  
26 consist of a comma-delimited string of socket ranges such as "32000-32100,  
27 33005,38123-38146". Additional array entries will consist of any provided resource  
28 request directives, along with their assigned values. Examples include:

29 **PMIX\_ALLOC\_FABRIC\_TYPE** - the type of resources provided;

30 **PMIX\_ALLOC\_FABRIC\_PLANE** - if applicable, what plane the resources were assigned  
31 from; **PMIX\_ALLOC\_FABRIC\_QOS** - the assigned QoS; **PMIX\_ALLOC\_BANDWIDTH** -  
32 the allocated bandwidth; **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** - a security key for the  
33 requested fabric allocation. NOTE: the array contents may differ from those requested,  
34 especially if **PMIX\_INFO\_REQD** was not set in the request.

35 **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (float)

1 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
2 request.

3 **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (char\*)

4 Fabric quality of service level for the job being requested in an allocation request.

5 **PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (char\*)

6 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

7 **PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (char\*)

8 ID string for the *fabric plane* to be used for the requested allocation.

9 **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (size\_t)

10 Number of endpoints to allocate per *process* in the job.

11 **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (size\_t)

12 Number of endpoints to allocate per *node* for the job.

13 **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** "pmix.alloc.nsec" (pmix\_byte\_object\_t)

14 Request that the allocation include a fabric security key for the spawned job.



## 15 Description

16 Non-blocking form of the [PMIx\\_Allocation\\_request](#) API.

### 17 13.1.3 Job Allocation attributes

18 Attributes used to describe the job allocation - these are values passed to and/or returned by the  
19 [PMIx\\_Allocation\\_request\\_nb](#) and [PMIx\\_Allocation\\_request](#) APIs and are not  
20 accessed using the [PMIx\\_Get](#) API.

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 **PMIX\_ALLOC\_ID** "pmix.alloc.id" (char\*)

25 A string identifier (provided by the host environment) for the resulting allocation which can  
26 later be used to reference the allocated resources in, for example, a call to [PMIx\\_Spawn](#).

27 **PMIX\_ALLOC\_QUEUE** "pmix.alloc.queue" (char\*)

28 Name of the WLM queue to which the allocation request is to be directed, or the queue being  
29 referenced in a query.

30 **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (uint64\_t)

31 The number of nodes being requested in an allocation request.

32 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)

33 Regular expression of the specific nodes being requested in an allocation request.

34 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (uint64\_t)

35 Number of PUs being requested in an allocation request.

1 **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (char\*)  
2 Regular expression of the number of PUs for each node being requested in an allocation  
3 request.

4 **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (char\*)  
5 Regular expression of the specific PUs being requested in an allocation request.

6 **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (float)  
7 Number of Megabytes[base2] of memory (per process) being requested in an allocation  
8 request.

9 **PMIX\_ALLOC\_FABRIC** "pmix.alloc.net" (array)  
10 Array of [pmix\\_info\\_t](#) describing requested fabric resources. This must include at least:  
11 [PMIX\\_ALLOC\\_FABRIC\\_ID](#), [PMIX\\_ALLOC\\_FABRIC\\_TYPE](#), and  
12 [PMIX\\_ALLOC\\_FABRIC\\_ENDPTS](#), plus whatever other descriptors are desired.

13 **PMIX\_ALLOC\_FABRIC\_ID** "pmix.alloc.netid" (char\*)  
14 The key to be used when accessing this requested fabric allocation. The fabric allocation  
15 will be returned/stored as a [pmix\\_data\\_array\\_t](#) of [pmix\\_info\\_t](#) whose first  
16 element is composed of this key and the allocated resource description. The type of the  
17 included value depends upon the fabric support. For example, a TCP allocation might  
18 consist of a comma-delimited string of socket ranges such as "32000–32100,  
19 33005, 38123–38146". Additional array entries will consist of any provided resource  
20 request directives, along with their assigned values. Examples include:  
21 [PMIX\\_ALLOC\\_FABRIC\\_TYPE](#) - the type of resources provided;  
22 [PMIX\\_ALLOC\\_FABRIC\\_PLANE](#) - if applicable, what plane the resources were assigned  
23 from; [PMIX\\_ALLOC\\_FABRIC\\_QOS](#) - the assigned QoS; [PMIX\\_ALLOC\\_BANDWIDTH](#) -  
24 the allocated bandwidth; [PMIX\\_ALLOC\\_FABRIC\\_SEC\\_KEY](#) - a security key for the  
25 requested fabric allocation. NOTE: the array contents may differ from those requested,  
26 especially if [PMIX\\_INFO\\_REQD](#) was not set in the request.

27 **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (float)  
28 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
29 request.

30 **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (char\*)  
31 Fabric quality of service level for the job being requested in an allocation request.

32 **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (uint32\_t)  
33 Total session time (in seconds) being requested in an allocation request.

34 **PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (char\*)  
35 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

36 **PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (char\*)  
37 ID string for the *fabric plane* to be used for the requested allocation.

38 **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (size\_t)  
39 Number of endpoints to allocate per *process* in the job.

40 **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (size\_t)  
41 Number of endpoints to allocate per *node* for the job.

42 **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** "pmix.alloc.nsec" (pmix\_byte\_object\_t)  
43 Request that the allocation include a fabric security key for the spawned job.

## 1 13.1.4 Job Allocation Directives

2 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 13.2 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.

## 1 13.2.1 PMIx\_Job\_control

### 2 Summary

3 Request a job control action.

### 4 Format

*PMIx v3.0*

C

```
5 pmix_status_t
6 PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
7                 const pmix_info_t directives[], size_t ndirs,
8                 pmix_info_t *results[], size_t *nresults);
```

C

#### 9 IN targets

10 Array of proc structures (array of handles)

#### 11 IN ntargets

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 `PMIX_SUCCESS` or a negative value indicating the error.

### Required Attributes

23 PMIx libraries are not required to directly support any attributes for this function. However, any  
24 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
25 *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process making  
26 the request.

27 Host environments that implement support for this operation are required to support the following  
28 attributes:

29 `PMIX_JOB_CTRL_ID` "pmix.jctrl.id" (char\*)

30 Provide a string identifier for this request. The user can provide an identifier for the  
31 requested operation, thus allowing them to later request status of the operation or to  
32 terminate it. The host, therefore, shall track it with the request for future reference.

33 `PMIX_JOB_CTRL_PAUSE` "pmix.jctrl.pause" (bool)

34 Pause the specified processes.

35 `PMIX_JOB_CTRL_RESUME` "pmix.jctrl.resume" (bool)

1 Resume (“un-pause”) the specified processes.

2 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (bool)

3 Forcibly terminate the specified processes and cleanup.

4 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)

5 Send given signal to specified processes.

6 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)

7 Politely terminate the specified processes.

8 **PMIX\_REGISTER\_CLEANUP** "pmix.reg.cleanup" (char\*)

9 Comma-delimited list of files to be removed upon process termination.

10 **PMIX\_REGISTER\_CLEANUP\_DIR** "pmix.reg.cleanupdir" (char\*)

11 Comma-delimited list of directories to be removed upon process termination.

12 **PMIX\_CLEANUP\_RECURSIVE** "pmix.clnup.recurse" (bool)

13 Recursively cleanup all subdirectories under the specified one(s).

14 **PMIX\_CLEANUP\_EMPTY** "pmix.clnup.empty" (bool)

15 Only remove empty subdirectories.

16 **PMIX\_CLEANUP\_IGNORE** "pmix.clnup.ignore" (char\*)

17 Comma-delimited list of filenames that are not to be removed.

18 **PMIX\_CLEANUP\_LEAVE\_TOPDIR** "pmix.clnup.lvtop" (bool)

19 When recursively cleaning subdirectories, do not remove the top-level directory (the one  
20 given in the cleanup request).



### Optional Attributes

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

22 **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (char\*)

23 Cancel the specified request - the provided request ID must match the

24 **PMIX\_JOB\_CTRL\_ID** provided to a previous call to **PMIx\_Job\_control**. An ID of

25 **NULL** implies cancel all requests from this requestor.

26 **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (char\*)

27 Restart the specified processes using the given checkpoint ID.

28 **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (char\*)

29 Checkpoint the specified processes and assign the given ID to it.

30 **PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (bool)

31 Use event notification to trigger a process checkpoint.

32 **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (int)

33 Use the given signal to trigger a process checkpoint.

1 **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (int)

2 Time in seconds to wait for a checkpoint to complete.

3 **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**

4 "pmix.jctrl.ckmethod" (pmix\_data\_array\_t)

5 Array of **pmix\_info\_t** declaring each method and value supported by this application.

6 **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (char\*)

7 Regular expression identifying nodes that are to be provisioned.

8 **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvning" (char\*)

9 Name of the image that is to be provisioned.

10 **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

11 Indicate that the job can be pre-empted.



## 12 Description

13 Request a job control action. The *targets* array identifies the processes to which the requested job  
14 control action is to be applied. All *clones* of an identified process are to have the requested action  
15 applied to them. A **NULL** value can be used to indicate all processes in the caller's namespace. The  
16 use of **PMIX\_RANK\_WILDCARD** can also be used to indicate that all processes in the given  
17 namespace are to be included.

18 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The returned  
19 *status* indicates whether or not the request was granted, and information as to the reason for any  
20 denial of the request shall be returned in the *results* array.

## 21 13.2.2 PMIx\_Job\_control\_nb

### 22 Summary

23 Request a job control action.

### 24 Format

PMIx v2.0

C

25 **pmix\_status\_t**

26 **PMIx\_Job\_control\_nb**(const pmix\_proc\_t targets[], size\_t ntargets,  
27 const pmix\_info\_t directives[], size\_t ndirs,  
28 pmix\_info\_cbfunc\_t cbfunc, void \*cbdata);



1 **IN targets**  
 2 Array of proc structures (array of handles)  
 3 **IN ntargets**  
 4 Number of elements in the *targets* array (integer)  
 5 **IN directives**  
 6 Array of info structures (array of handles)  
 7 **IN ndirs**  
 8 Number of elements in the *directives* 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 A successful return indicates that the request is being processed and the result will be returned in  
 14 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
 15 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

16 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- 17 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
 18 returned *success* - the *cbfunc* will *not* be called

19 If none of the above return codes are appropriate, then an implementation must return either a  
 20 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 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, and the PMIx library is  
 23 *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process making  
 24 the request.

25 Host environments that implement support for this operation are required to support the following  
 26 attributes:

27 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (char\*)

28 Provide a string identifier for this request. The user can provide an identifier for the  
 29 requested operation, thus allowing them to later request status of the operation or to  
 30 terminate it. The host, therefore, shall track it with the request for future reference.

31 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (bool)

32 Pause the specified processes.

33 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (bool)

34 Resume ("un-pause") the specified processes.

35 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (bool)

1 Forcibly terminate the specified processes and cleanup.

2 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)

3 Send given signal to specified processes.

4 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)

5 Politely terminate the specified processes.

6 **PMIX\_REGISTER\_CLEANUP** "pmix.reg.cleanup" (char\*)

7 Comma-delimited list of files to be removed upon process termination.

8 **PMIX\_REGISTER\_CLEANUP\_DIR** "pmix.reg.cleanupdir" (char\*)

9 Comma-delimited list of directories to be removed upon process termination.

10 **PMIX\_CLEANUP\_RECURSIVE** "pmix.clnup.recurse" (bool)

11 Recursively cleanup all subdirectories under the specified one(s).

12 **PMIX\_CLEANUP\_EMPTY** "pmix.clnup.empty" (bool)

13 Only remove empty subdirectories.

14 **PMIX\_CLEANUP\_IGNORE** "pmix.clnup.ignore" (char\*)

15 Comma-delimited list of filenames that are not to be removed.

16 **PMIX\_CLEANUP\_LEAVE\_TOPDIR** "pmix.clnup.lvtop" (bool)

17 When recursively cleaning subdirectories, do not remove the top-level directory (the one  
18 given in the cleanup request).

▲-----▲  
▼-----▼ **Optional Attributes** -----▼

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

20 **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (char\*)

21 Cancel the specified request - the provided request ID must match the

22 **PMIX\_JOB\_CTRL\_ID** provided to a previous call to **PMIx\_Job\_control**. An ID of  
23 **NULL** implies cancel all requests from this requestor.

24 **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (char\*)

25 Restart the specified processes using the given checkpoint ID.

26 **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (char\*)

27 Checkpoint the specified processes and assign the given ID to it.

28 **PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (bool)

29 Use event notification to trigger a process checkpoint.

30 **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (int)

31 Use the given signal to trigger a process checkpoint.

32 **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (int)

33 Time in seconds to wait for a checkpoint to complete.

1 **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
 2 "pmix.jctrl.ckmethod" (pmix\_data\_array\_t)  
 3 Array of pmix\_info\_t declaring each method and value supported by this application.  
 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.pvning" (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 Description

11 Non-blocking form of the **PMIx\_Job\_control** API. The *targets* array identifies the processes to  
 12 which the requested job control action is to be applied. All *clones* of an identified process are to  
 13 have the requested action applied to them. A **NULL** value can be used to indicate all processes in  
 14 the caller's namespace. The use of **PMIX\_RANK\_WILDCARD** can also be used to indicate that all  
 15 processes in the given namespace are to be included.

16 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The callback  
 17 function provides a *status* to indicate whether or not the request was granted, and to provide some  
 18 information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
 19 **pmix\_info\_t** structures.

### 20 13.2.3 Job control constants

21 The following constants are specifically defined for return by the job control APIs:

22 **PMIX\_ERR\_CONFLICTING\_CLEANUP\_DIRECTIVES** Conflicting directives given for  
 23 job/process cleanup.

### 24 13.2.4 Job control events

25 The following job control events may be available for registration, depending upon implementation  
 26 and host environment support:

27 **PMIX\_JCTRL\_CHECKPOINT** Monitored by PMIx client to trigger a checkpoint operation.  
 28 **PMIX\_JCTRL\_CHECKPOINT\_COMPLETE** Sent by a PMIx client and monitored by a PMIx  
 29 server to notify that requested checkpoint operation has completed.  
 30 **PMIX\_JCTRL\_PREEMPT\_ALERT** Monitored by a PMIx client to detect that an RM intends to  
 31 preempt the job.  
 32 **PMIX\_ERR\_PROC\_RESTART** Error in process restart.  
 33 **PMIX\_ERR\_PROC\_CHECKPOINT** Error in process checkpoint.  
 34 **PMIX\_ERR\_PROC\_MIGRATE** Error in process migration.

## 1 13.2.5 Job control attributes

2 Attributes used to request control operations on an executing application - these are values passed  
3 to the job control APIs and are not accessed using the [PMIx\\_Get](#) API.

4 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (char\*)  
5 Provide a string identifier for this request. The user can provide an identifier for the  
6 requested operation, thus allowing them to later request status of the operation or to  
7 terminate it. The host, therefore, shall track it with the request for future reference.

8 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (bool)  
9 Pause the specified processes.

10 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (bool)  
11 Resume ("un-pause") the specified processes.

12 **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (char\*)  
13 Cancel the specified request - the provided request ID must match the  
14 [PMIX\\_JOB\\_CTRL\\_ID](#) provided to a previous call to [PMIx\\_Job\\_control](#). An ID of  
15 NULL implies cancel all requests from this requestor.

16 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (bool)  
17 Forcibly terminate the specified processes and cleanup.

18 **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (char\*)  
19 Restart the specified processes using the given checkpoint ID.

20 **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (char\*)  
21 Checkpoint the specified processes and assign the given ID to it.

22 **PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (bool)  
23 Use event notification to trigger a process checkpoint.

24 **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (int)  
25 Use the given signal to trigger a process checkpoint.

26 **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (int)  
27 Time in seconds to wait for a checkpoint to complete.

28 **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
29 "pmix.jctrl.ckmethod" (pmix\_data\_array\_t)  
30 Array of [pmix\\_info\\_t](#) declaring each method and value supported by this application.

31 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)  
32 Send given signal to specified processes.

33 **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (char\*)  
34 Regular expression identifying nodes that are to be provisioned.

35 **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvnimg" (char\*)  
36 Name of the image that is to be provisioned.

37 **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)  
38 Indicate that the job can be pre-empted.

39 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)  
40 Politely terminate the specified processes.

41 **PMIX\_REGISTER\_CLEANUP** "pmix.reg.cleanup" (char\*)  
42 Comma-delimited list of files to be removed upon process termination.

1 **PMIX\_REGISTER\_CLEANUP\_DIR** "pmix.reg.cleanupdir" (char\*)  
 2 Comma-delimited list of directories to be removed upon process termination.  
 3 **PMIX\_CLEANUP\_RECURSIVE** "pmix.clnup.recurse" (bool)  
 4 Recursively cleanup all subdirectories under the specified one(s).  
 5 **PMIX\_CLEANUP\_EMPTY** "pmix.clnup.empty" (bool)  
 6 Only remove empty subdirectories.  
 7 **PMIX\_CLEANUP\_IGNORE** "pmix.clnup.ignore" (char\*)  
 8 Comma-delimited list of filenames that are not to be removed.  
 9 **PMIX\_CLEANUP\_LEAVE\_TOPDIR** "pmix.clnup.lvtop" (bool)  
 10 When recursively cleaning subdirectories, do not remove the top-level directory (the one  
 11 given in the cleanup request).

## 12 13.3 Process and Job Monitoring

13 In addition to external faults, a common problem encountered in HPC applications is a failure to  
 14 make progress due to some internal conflict in the computation. These situations can result in a  
 15 significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the  
 16 job. Various watchdog methods have been developed for detecting this situation, including  
 17 requiring a periodic “heartbeat” from the application and monitoring a specified file for changes in  
 18 size and/or modification time.

19 The following APIs allow applications to request monitoring, directing what is to be monitored, the  
 20 frequency of the associated check, whether or not the application is to be notified (via the event  
 21 notification subsystem) of stall detection, and other characteristics of the operation.

### 22 13.3.1 PMIx\_Process\_monitor

#### 23 Summary

24 Request that application processes be monitored.

#### 25 *PMIx v3.0* Format

```
26 pmix_status_t
27 PMIx_Process_monitor(const pmix_info_t *monitor,
28                     pmix_status_t error,
29                     const pmix_info_t directives[], size_t ndirs,
30                     pmix_info_t *results[], size_t *nresults);
```

31 **IN** **monitor**  
 32 info (handle)  
 33 **IN** **error**  
 34 status (integer)

1 **IN directives**

2 Array of info structures (array of handles)

3 **IN ndirs**

4 Number of elements in the *directives* array (integer)

5 **INOUT results**

6 Address where a pointer to an array of `pmix_info_t` containing the results of the request  
7 can be returned (memory reference)

8 **INOUT nresults**

9 Address where the number of elements in *results* can be returned (handle)

10 A successful return indicates that the results have been placed in the *results* array.

11 Returns `PMIX_SUCCESS` or a negative value indicating the error.

▼----- Optional Attributes -----▼

12 The following attributes may be implemented by a PMIx library or by the host environment. If  
13 supported by the PMIx server library, then the library must not pass the supported attributes to the  
14 host environment. All attributes not directly supported by the server library must be passed to the  
15 host environment if it supports this operation, and the library is *required* to add the  
16 `PMIX_USERID` and the `PMIX_GRPID` attributes of the requesting process:

17 `PMIX_MONITOR_ID` "pmix.monitor.id" (char\*)

18 Provide a string identifier for this request.

19 `PMIX_MONITOR_CANCEL` "pmix.monitor.cancel" (char\*)

20 Identifier to be canceled (`NULL` means cancel all monitoring for this process).

21 `PMIX_MONITOR_APP_CONTROL` "pmix.monitor.appctrl" (bool)

22 The application desires to control the response to a monitoring event - i.e., the application is  
23 requesting that the host environment not take immediate action in response to the event (e.g.,  
24 terminating the job).

25 `PMIX_MONITOR_HEARTBEAT` "pmix.monitor.mbeat" (void)

26 Register to have the PMIx server monitor the requestor for heartbeats.

27 `PMIX_MONITOR_HEARTBEAT_TIME` "pmix.monitor.btime" (uint32\_t)

28 Time in seconds before declaring heartbeat missed.

29 `PMIX_MONITOR_HEARTBEAT_DROPS` "pmix.monitor.bdrop" (uint32\_t)

30 Number of heartbeats that can be missed before generating the event.

31 `PMIX_MONITOR_FILE` "pmix.monitor.fmon" (char\*)

32 Register to monitor file for signs of life.

33 `PMIX_MONITOR_FILE_SIZE` "pmix.monitor.fsize" (bool)

34 Monitor size of given file is growing to determine if the application is running.

35 `PMIX_MONITOR_FILE_ACCESS` "pmix.monitor.faccess" (char\*)

36 Monitor time since last access of given file to determine if the application is running.

1 **PMIX\_MONITOR\_FILE\_MODIFY** "pmix.monitor.fmod" (char\*)  
2 Monitor time since last modified of given file to determine if the application is running.  
3 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)  
4 Time in seconds between checking the file.  
5 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)  
6 Number of file checks that can be missed before generating the event.  
7 **PMIX\_SEND\_HEARTBEAT** "pmix.monitor.beat" (void)  
8 Send heartbeat to local PMIx server.

---

## Description

Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not become “wedged”. When a monitor detects the specified alarm condition, it will generate an event notification using the provided error code and passing along any available relevant information. It is up to the caller to register a corresponding event handler.

The *monitor* argument is an attribute indicating the type of monitor being requested. For example, **PMIX\_MONITOR\_FILE** to indicate that the requestor is asking that a file be monitored.

The *error* argument is the status code to be used when generating an event notification alerting that the monitor has been triggered. The range of the notification defaults to **PMIX\_RANGE\_NAMESPACE**. This can be changed by providing a **PMIX\_RANGE** directive.

The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency of checking to be done

The returned *status* indicates whether or not the request was granted, and information as to the reason for any denial of the request shall be returned in the *results* array.

## 13.3.2 PMIx\_Process\_monitor\_nb

### Summary

Request that application processes be monitored.

## Format

```
pmix_status_t
PMIx_Process_monitor_nb(const pmix_info_t *monitor,
                        pmix_status_t error,
                        const pmix_info_t directives[],
                        size_t ndirs,
                        pmix_info_cbfunc_t cbfunc, void *cbdata);
```

- IN monitor**  
info (handle)
- IN error**  
status (integer)
- IN directives**  
Array of info structures (array of handles)
- IN ndirs**  
Number of elements in the *directives* array (integer)
- IN cbfunc**  
Callback function [pmix\\_info\\_cbfunc\\_t](#) (function reference)
- IN cbdata**  
Data to be passed to the callback function (memory reference)

Returns one of the following:

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Optional Attributes

The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the requesting process:

**PMIX\_MONITOR\_ID** "pmix.monitor.id" (char\*)  
Provide a string identifier for this request.



1 **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (char\*)  
 2 Identifier to be canceled (NULL means cancel all monitoring for this process).

3 **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (bool)  
 4 The application desires to control the response to a monitoring event - i.e., the application is  
 5 requesting that the host environment not take immediate action in response to the event (e.g.,  
 6 terminating the job).

7 **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (void)  
 8 Register to have the PMIx server monitor the requestor for heartbeats.

9 **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (uint32\_t)  
 10 Time in seconds before declaring heartbeat missed.

11 **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (uint32\_t)  
 12 Number of heartbeats that can be missed before generating the event.

13 **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (char\*)  
 14 Register to monitor file for signs of life.

15 **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (bool)  
 16 Monitor size of given file is growing to determine if the application is running.

17 **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (char\*)  
 18 Monitor time since last access of given file to determine if the application is running.

19 **PMIX\_MONITOR\_FILE\_MODIFY** "pmix.monitor.fmod" (char\*)  
 20 Monitor time since last modified of given file to determine if the application is running.

21 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)  
 22 Time in seconds between checking the file.

23 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)  
 24 Number of file checks that can be missed before generating the event.

25 **PMIX\_SEND\_HEARTBEAT** "pmix.monitor.beat" (void)  
 26 Send heartbeat to local PMIx server.

▲-----▲

## 27 Description

28 Non-blocking form of the **PMIx\_Process\_monitor** API. The *cbfunc* function provides a  
 29 *status* to indicate whether or not the request was granted, and to provide some information as to the  
 30 reason for any denial in the **pmix\_info\_cbfunc\_t** array of **pmix\_info\_t** structures.

### 31 13.3.3 PMIx\_Heartbeat

#### 32 Summary

33 Send a heartbeat to the PMIx server library

## Format

```
PMIx_Heartbeat ();
```

## Description

A simplified macro wrapping `PMIx_Process_monitor_nb` that sends a heartbeat to the PMIx server library.

### 13.3.4 Monitoring events

The following monitoring events may be available for registration, depending upon implementation and host environment support:

**PMIX\_MONITOR\_HEARTBEAT\_ALERT** Heartbeat failed to arrive within specified window.

The process that triggered this alert will be identified in the event.

**PMIX\_MONITOR\_FILE\_ALERT** File failed its monitoring detection criteria. The file that triggered this alert will be identified in the event.

### 13.3.5 Monitoring attributes

Attributes used to control monitoring of an executing application- these are values passed to the `PMIx_Process_monitor_nb` API and are not accessed using the `PMIx_Get` API.

**PMIX\_MONITOR\_ID** "pmix.monitor.id" (char\*)

Provide a string identifier for this request.

**PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (char\*)

Identifier to be canceled (NULL means cancel all monitoring for this process).

**PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (bool)

The application desires to control the response to a monitoring event - i.e., the application is requesting that the host environment not take immediate action in response to the event (e.g., terminating the job).

**PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (void)

Register to have the PMIx server monitor the requestor for heartbeats.

**PMIX\_SEND\_HEARTBEAT** "pmix.monitor.beat" (void)

Send heartbeat to local PMIx server.

**PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (uint32\_t)

Time in seconds before declaring heartbeat missed.

**PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (uint32\_t)

Number of heartbeats that can be missed before generating the event.

**PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (char\*)

Register to monitor file for signs of life.

**PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (bool)

Monitor size of given file is growing to determine if the application is running.

```

1 PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
2     Monitor time since last access of given file to determine if the application is running.
3 PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
4     Monitor time since last modified of given file to determine if the application is running.
5 PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
6     Time in seconds between checking the file.
7 PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
8     Number of file checks that can be missed before generating the event.

```

## 9 13.4 Logging

10 The logging interface supports posting information by applications and SMS elements to persistent  
11 storage. This function is *not* intended for output of computational results, but rather for reporting  
12 status and saving state information such as inserting computation progress reports into the  
13 application's SMS job log or error reports to the local syslog.

### 14 13.4.1 PMIx\_Log

#### 15 Summary

16 Log data to a data service.

#### 17 Format

*PMIx v3.0*

```

18 pmix_status_t
19 PMIx_Log(const pmix_info_t data[], size_t ndata,
20         const pmix_info_t directives[], size_t ndirs);

```

21 **IN data**  
22 Array of info structures (array of handles)

23 **IN ndata**  
24 Number of elements in the *data* array (**size\_t**)

25 **IN directives**  
26 Array of info structures (array of handles)

27 **IN ndirs**  
28 Number of elements in the *directives* array (**size\_t**)

29 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

## Required Attributes

1 If the PMIx library does not itself perform this operation, then it is required to pass any attributes  
2 provided by the client to the host environment for processing. In addition, it must include the  
3 following attributes in the passed *info* array:

4 **PMIX\_USERID** "pmix.euid" (uint32\_t)

5 Effective user ID of the connecting process.

6 **PMIX\_GRPID** "pmix.egid" (uint32\_t)

7 Effective group ID of the connecting process.

8 Host environments or PMIx libraries that implement support for this operation are required to  
9 support the following attributes:

10 **PMIX\_LOG\_STDERR** "pmix.log.stderr" (char\*)

11 Log string to **stderr**.

12 **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (char\*)

13 Log string to **stdout**.

14 **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (char\*)

15 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
16 otherwise to local syslog.

17 **PMIX\_LOG\_LOCAL\_SYSLOG** "pmix.log.lsys" (char\*)

18 Log data to local syslog. Defaults to **ERROR** priority.

19 **PMIX\_LOG\_GLOBAL\_SYSLOG** "pmix.log.gsys" (char\*)

20 Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.

21 **PMIX\_LOG\_SYSLOG\_PRI** "pmix.log.syspri" (int)

22 Syslog priority level.

23 **PMIX\_LOG\_ONCE** "pmix.log.once" (bool)

24 Only log this once with whichever channel can first support it, taking the channels in priority  
25 order.

## Optional Attributes

26 The following attributes are optional for host environments or PMIx libraries that support this  
27 operation:

28 **PMIX\_LOG\_SOURCE** "pmix.log.source" (pmix\_proc\_t\*)

29 ID of source of the log request.

30 **PMIX\_LOG\_TIMESTAMP** "pmix.log.tstamp" (time\_t)

31 Timestamp for log report.

32 **PMIX\_LOG\_GENERATE\_TIMESTAMP** "pmix.log.gtstamp" (bool)

1           Generate timestamp for log.

2     **PMIX\_LOG\_TAG\_OUTPUT** "pmix.log.tag" (bool)

3           Label the output stream with the channel name (e.g., "stdout").

4     **PMIX\_LOG\_TIMESTAMP\_OUTPUT** "pmix.log.tsout" (bool)

5           Print timestamp in output string.

6     **PMIX\_LOG\_XML\_OUTPUT** "pmix.log.xml" (bool)

7           Print the output stream in eXtensible Markup Language (XML) format.

8     **PMIX\_LOG\_EMAIL** "pmix.log.email" (pmix\_data\_array\_t)

9           Log via email based on **pmix\_info\_t** containing directives.

10    **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (char\*)

11           Comma-delimited list of email addresses that are to receive the message.

12    **PMIX\_LOG\_EMAIL\_SENDER\_ADDR** "pmix.log.emfaddr" (char\*)

13           Return email address of sender.

14    **PMIX\_LOG\_EMAIL\_SERVER** "pmix.log.esrvr" (char\*)

15           Hostname (or IP address) of SMTP server.

16    **PMIX\_LOG\_EMAIL\_SRVR\_PORT** "pmix.log.esrvrprt" (int32\_t)

17           Port the email server is listening to.

18    **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (char\*)

19           Subject line for email.

20    **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (char\*)

21           Message to be included in email.

22    **PMIX\_LOG\_JOB\_RECORD** "pmix.log.jrec" (bool)

23           Log the provided information to the host environment's job record.

24    **PMIX\_LOG\_GLOBAL\_DATASTORE** "pmix.log.gstore" (bool)

25           Store the log data in a global data store (e.g., database).



## 26     **Description**

27     Log data subject to the services offered by the host environment. The data to be logged is provided  
28     in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel.

## 29     **Advice to users**

30     It is strongly recommended that the **PMIx\_Log** API not be used by applications for streaming data  
31     as it is not a "performant" transport and can perturb the application since it involves the local PMIx  
32     server and host SMS daemon. Note that a return of **PMIX\_SUCCESS** only denotes that the data  
33     was successfully handed to the appropriate system call (for local channels) or the host environment  
   and does not indicate receipt at the final destination.

---

## 1 13.4.2 PMI<sub>x</sub>\_Log\_nb

### 2 Summary

3 Log data to a data service.

### 4 Format

*PMI<sub>x</sub> v2.0*

C

```
5 pmix_status_t  
6 PMIx_Log_nb(const pmix_info_t data[], size_t ndata,  
7             const pmix_info_t directives[], size_t ndirs,  
8             pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

#### 9 IN data

10 Array of info structures (array of handles)

#### 11 IN ndata

12 Number of elements in the *data* array (**size\_t**)

#### 13 IN directives

14 Array of info structures (array of handles)

#### 15 IN ndirs

16 Number of elements in the *directives* array (**size\_t**)

#### 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 Return codes are one of the following:

22 A successful return indicates that the request is being processed and the result will be returned in  
23 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
24 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

25 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

26 **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
27 returned *success* - the *cbfunc* will *not* be called

28 **PMIX\_ERR\_BAD\_PARAM** The logging request contains at least one incorrect entry that prevents  
29 it from being processed. The callback function will not be called.

30 If none of the above return codes are appropriate, then an implementation must return either a  
31 general PMI<sub>x</sub> error code or an implementation defined error code as described in Section 3.1.1.

## Required Attributes

1 If the PMIx library does not itself perform this operation, then it is required to pass any attributes  
2 provided by the client to the host environment for processing. In addition, it must include the  
3 following attributes in the passed *info* array:

4 **PMIX\_USERID** "pmix.euid" (uint32\_t)

5 Effective user ID of the connecting process.

6 **PMIX\_GRPID** "pmix.egid" (uint32\_t)

7 Effective group ID of the connecting process.

8 Host environments or PMIx libraries that implement support for this operation are required to  
9 support the following attributes:

10 **PMIX\_LOG\_STDERR** "pmix.log.stderr" (char\*)

11 Log string to **stderr**.

12 **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (char\*)

13 Log string to **stdout**.

14 **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (char\*)

15 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
16 otherwise to local syslog.

17 **PMIX\_LOG\_LOCAL\_SYSLOG** "pmix.log.lsys" (char\*)

18 Log data to local syslog. Defaults to **ERROR** priority.

19 **PMIX\_LOG\_GLOBAL\_SYSLOG** "pmix.log.gsys" (char\*)

20 Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.

21 **PMIX\_LOG\_SYSLOG\_PRI** "pmix.log.syspri" (int)

22 Syslog priority level.

23 **PMIX\_LOG\_ONCE** "pmix.log.once" (bool)

24 Only log this once with whichever channel can first support it, taking the channels in priority  
25 order.

## Optional Attributes

26 The following attributes are optional for host environments or PMIx libraries that support this  
27 operation:

28 **PMIX\_LOG\_SOURCE** "pmix.log.source" (pmix\_proc\_t\*)

29 ID of source of the log request.

30 **PMIX\_LOG\_TIMESTAMP** "pmix.log.tstamp" (time\_t)

31 Timestamp for log report.

32 **PMIX\_LOG\_GENERATE\_TIMESTAMP** "pmix.log.gtstamp" (bool)

1           Generate timestamp for log.

2     **PMIX\_LOG\_TAG\_OUTPUT** "pmix.log.tag" (bool)

3           Label the output stream with the channel name (e.g., "stdout").

4     **PMIX\_LOG\_TIMESTAMP\_OUTPUT** "pmix.log.tsout" (bool)

5           Print timestamp in output string.

6     **PMIX\_LOG\_XML\_OUTPUT** "pmix.log.xml" (bool)

7           Print the output stream in XML format.

8     **PMIX\_LOG\_EMAIL** "pmix.log.email" (pmix\_data\_array\_t)

9           Log via email based on **pmix\_info\_t** containing directives.

10    **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (char\*)

11           Comma-delimited list of email addresses that are to receive the message.

12    **PMIX\_LOG\_EMAIL\_SENDER\_ADDR** "pmix.log.emfaddr" (char\*)

13           Return email address of sender.

14    **PMIX\_LOG\_EMAIL\_SERVER** "pmix.log.esrvr" (char\*)

15           Hostname (or IP address) of SMTP server.

16    **PMIX\_LOG\_EMAIL\_SRVR\_PORT** "pmix.log.esrvrprt" (int32\_t)

17           Port the email server is listening to.

18    **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (char\*)

19           Subject line for email.

20    **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (char\*)

21           Message to be included in email.

22    **PMIX\_LOG\_JOB\_RECORD** "pmix.log.jrec" (bool)

23           Log the provided information to the host environment's job record.

24    **PMIX\_LOG\_GLOBAL\_DATASTORE** "pmix.log.gstore" (bool)

25           Store the log data in a global data store (e.g., database).



## 26     **Description**

27     Log data subject to the services offered by the host environment. The data to be logged is provided  
28     in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The  
29     callback function will be executed when the log operation has been completed. The *data* and  
30     *directives* arrays must be maintained until the callback is provided.



## Advice to users

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.

### 13.4.3 Log attributes

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.

`PMIX_LOG_SOURCE` "pmix.log.source" (`pmix_proc_t*`)

ID of source of the log request.

`PMIX_LOG_STDERR` "pmix.log.stderr" (`char*`)

Log string to `stderr`.

`PMIX_LOG_STDOUT` "pmix.log.stdout" (`char*`)

Log string to `stdout`.

`PMIX_LOG_SYSLOG` "pmix.log.syslog" (`char*`)

Log data to syslog. Defaults to `ERROR` priority. Will log to global syslog if available, otherwise to local syslog.

`PMIX_LOG_LOCAL_SYSLOG` "pmix.log.lsys" (`char*`)

Log data to local syslog. Defaults to `ERROR` priority.

`PMIX_LOG_GLOBAL_SYSLOG` "pmix.log.gsys" (`char*`)

Forward data to system “gateway” and log msg to that syslog Defaults to `ERROR` priority.

`PMIX_LOG_SYSLOG_PRI` "pmix.log.syspri" (`int`)

Syslog priority level.

`PMIX_LOG_TIMESTAMP` "pmix.log.tstamp" (`time_t`)

Timestamp for log report.

`PMIX_LOG_GENERATE_TIMESTAMP` "pmix.log.gtstamp" (`bool`)

Generate timestamp for log.

`PMIX_LOG_TAG_OUTPUT` "pmix.log.tag" (`bool`)

Label the output stream with the channel name (e.g., “stdout”).

`PMIX_LOG_TIMESTAMP_OUTPUT` "pmix.log.tsout" (`bool`)

Print timestamp in output string.

`PMIX_LOG_XML_OUTPUT` "pmix.log.xml" (`bool`)

Print the output stream in XML format.

`PMIX_LOG_ONCE` "pmix.log.once" (`bool`)

Only log this once with whichever channel can first support it, taking the channels in priority order.

`PMIX_LOG_MSG` "pmix.log.msg" (`pmix_byte_object_t`)

1 Message blob to be sent somewhere.

2 **PMIX\_LOG\_EMAIL** "pmix.log.email" (pmix\_data\_array\_t)

3 Log via email based on pmix\_info\_t containing directives.

4 **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (char\*)

5 Comma-delimited list of email addresses that are to receive the message.

6 **PMIX\_LOG\_EMAIL\_SENDER\_ADDR** "pmix.log.emfaddr" (char\*)

7 Return email address of sender.

8 **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (char\*)

9 Subject line for email.

10 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (char\*)

11 Message to be included in email.

12 **PMIX\_LOG\_EMAIL\_SERVER** "pmix.log.esrvr" (char\*)

13 Hostname (or IP address) of SMTP server.

14 **PMIX\_LOG\_EMAIL\_SRVR\_PORT** "pmix.log.esrvrprt" (int32\_t)

15 Port the email server is listening to.

16 **PMIX\_LOG\_GLOBAL\_DATASTORE** "pmix.log.gstore" (bool)

17 Store the log data in a global data store (e.g., database).

18 **PMIX\_LOG\_JOB\_RECORD** "pmix.log.jrec" (bool)

19 Log the provided information to the host environment's job record.

## CHAPTER 14

# 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.

## 4 14.1 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 • *Process sets* have no implied relationship between their members - i.e., a process in a process set  
24 has no concept of a "pset rank" as it would in a *process group*.
- 25 • *Process set* identifiers are set by the host environment or by the user at time of application  
26 submission for execution - there are no PMIx APIs provided by which an application can define a  
27 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.

## 14.1.1 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**).

## 1 14.1.2 Process Set Attributes

2 Several attributes are provided for querying the system regarding process sets using the  
3 [PMIx\\_Query\\_info](#) APIs.

4 **PMIX\_QUERY\_NUM\_PSETS** "pmix.qry.psetnum" (size\_t)

5 Return the number of process sets defined in the specified range (defaults to  
6 [PMIX\\_RANGE\\_SESSION](#)).

7 **PMIX\_QUERY\_PSET\_NAMES** "pmix.qry.psets" (pmix\_data\_array\_t\*)

8 Return a [pmix\\_data\\_array\\_t](#) containing an array of strings of the process set names  
9 defined in the specified range (defaults to [PMIX\\_RANGE\\_SESSION](#)).

10 **PMIX\_QUERY\_PSET\_MEMBERSHIP** "pmix.qry.pmems" (pmix\_data\_array\_t\*)

11 Return an array of [pmix\\_proc\\_t](#) containing the members of the specified process set.

12 The [PMIX\\_PROCESS\\_SET\\_DEFINE](#) event shall include the name of the newly defined process  
13 set and its members: **PMIX\_PSET\_NAME** "pmix.pset.nm" (char\*)

14 The name of the newly defined process set.

15 **PMIX\_PSET\_MEMBERS** "pmix.pset.mems" (pmix\_data\_array\_t\*)

16 An array of [pmix\\_proc\\_t](#) containing the members of the newly defined process set.

17 In addition, a process can request (via [PMIx\\_Get](#)) the process sets to which a given process  
18 (including itself) belongs:

19 **PMIX\_PSET\_NAMES** "pmix.pset.nms" (pmix\_data\_array\_t\*)

20 Returns an array of **char\*** string names of the process sets in which the given process is a  
21 member.

## 22 14.2 Process Groups

23 *PMIx Groups* are defined as a collection of processes desiring a common, unique identifier for  
24 operational purposes such as passing events or participating in [PMIx fence](#) operations. As with  
25 processes that assemble via [PMIx\\_Connect](#), each member of the group is provided with both the  
26 job-level information of any other namespace represented in the group, and the contact information  
27 for all group members.

28 However, members of *PMIx Groups* are *loosely coupled* as opposed to *tightly connected* when  
29 constructed via [PMIx\\_Connect](#). Thus, *groups* differ from [PMIx\\_Connect](#) assemblages in  
30 several key areas, as detailed in the following sections.

### 31 14.2.1 Relation to the host environment

32 Calls to *PMIx Group* APIs are first processed within the local *PMIx server*. When constructed, the  
33 server creates a tracker that associates the specified processes with the user-provided group  
34 identifier, and assigns a new *group rank* based on their relative position in the array of processes  
35 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”

## 17 14.2.2 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 14.2.3 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 14.2.4 Process Group Events

25 *PMIx v4.0*

26 Asynchronous process group operations rely heavily on PMIx events. The following events have  
27 been defined for that purpose.

28 **PMIX\_GROUP\_INVITED** The process has been invited to join a PMIx Group - the identifier of  
29 the group and the ID's of other invited (or already joined) members will be included in the  
30 notification.

31 **PMIX\_GROUP\_LEFT** A process has asynchronously left a PMIx Group - the process identifier  
32 of the departing process will be included in the notification.

33 **PMIX\_GROUP\_MEMBER\_FAILED** A member of a PMIx Group has abnormally terminated  
34 (i.e., without formally leaving the group prior to termination) - the process identifier of the  
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.

## 25 14.2.5 Process Group Attributes

26 *PMIx v4.0*

Attributes for querying the system regarding process groups include:

27 **PMIX\_QUERY\_NUM\_GROUPS** "pmix.qry.pgrpnum" (**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** "pmix.qry.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 "pmix.qry.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 14.2.6 PMIx\_Group\_construct

#### 4 Summary

5 Construct a PMIx process group.

#### 6 *PMIx v4.0* Format

C

```
7 pmix_status_t  
8 PMIx_Group_construct(const char grp[],  
9                     const pmix_proc_t procs[], size_t nprocs,  
10                    const pmix_info_t directives[],  
11                    size_t ndirs,  
12                    pmix_info_t **results,  
13                    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 **PMIX\_SUCCESS** or a negative value indicating the error.

## Required Attributes

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 support 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

Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less than or equal to **PMIX\_MAX\_NSLEN**. Only characters accepted by standard string comparison functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with 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.

If the **PMIX\_GROUP\_NOTIFY\_TERMINATION** attribute is provided and has a value of **true**, then either the construct leader (if **PMIX\_GROUP\_LEADER** is provided) or all participants who register for the **PMIX\_GROUP\_MEMBER\_FAILED** event will receive events whenever a process fails or terminates prior to calling **PMIx\_Group\_construct** – i.e. if a *group leader* is declared, *only* that process will receive the event. In the absence of a declared leader, *all* specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the **PMIX\_GROUP\_MEMBERSHIP** attribute) as part of the *results* array returned by this API.

Failure of the declared leader at any time will cause a **PMIX\_GROUP\_LEADER\_FAILED** event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX\_GROUP\_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX\_GROUP\_LEADER\_SELECTED** event identifying the new leader. If no leader was selected, then the **pmix\_info\_t** provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns **PMIX\_GROUP\_CONSTRUCT\_ABORT** from either the **PMIX\_GROUP\_MEMBER\_FAILED** or the **PMIX\_GROUP\_LEADER\_FAILED** event handler will cause the construct process to abort, returning from the call with a **PMIX\_GROUP\_CONSTRUCT\_ABORT** status.

If the **PMIX\_GROUP\_NOTIFY\_TERMINATION** attribute is not provided or has a value of **false**, then the **PMIx\_Group\_construct** operation will simply return an error whenever a proposed group member fails or terminates prior to calling **PMIx\_Group\_construct**.

Providing the **PMIX\_GROUP\_OPTIONAL** attribute with a value of **true** directs the PMIx library to consider participation by any specified group member as non-required - thus, the operation will 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.

## 19 14.2.7 `PMIx_Group_construct_nb`

### 20 Summary

21 Non-blocking form of `PMIx_Group_construct`.

## Format

C

```
pmix_status_t
PMIx_Group_construct_nb(const char grp[],
                       const pmix_proc_t procs[], size_t nprocs,
                       const pmix_info_t directives[],
                       size_t ndirs,
                       pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

- IN grp**  
NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group identifier (string)
- IN procs**  
Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the member processes (array of handles)
- IN nprocs**  
Number of elements in the *procs* array (**size\_t**)
- IN directives**  
Array of **pmix\_info\_t** structures (array of handles)
- IN ndirs**  
Number of elements in the *directives* array (**size\_t**)
- IN cbfunc**  
Callback function **pmix\_info\_cbfunc\_t** (function reference)
- IN cbdata**  
Data to be passed to the callback function (memory reference)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

**PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed successfully - the *cbfunc* will *not* be called.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

If executed, the status returned in the provided callback function will be one of the following 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.

- 1 • **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM  
2 does not.
- 3 • a non-zero PMIx error constant indicating a reason for the request's failure.

### Required Attributes

4 PMIx libraries that choose not to support this operation *must* return  
5 **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called.

6 The following attributes are *required* to be supported by all PMIx libraries that support this  
7 operation:

8 **PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (bool)

9 This process is the leader of the group.

10 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

11 Participation is optional - do not return an error if any of the specified processes terminate  
12 without having joined. The default is **false**.

13 **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

14 Group operation only involves local processes. PMIx implementations are *required* to  
15 automatically scan an array of group members for local vs remote processes - if only local  
16 processes are detected, the implementation need not execute a global collective for the  
17 operation unless a context ID has been requested from the host environment. This can result  
18 in significant time savings. This attribute can be used to optimize the operation by indicating  
19 whether or not only local processes are represented, thus allowing the implementation to  
20 bypass the scan.

21 **PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)

22 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective  
23 operation.

24 Host environments that support this operation are *required* to provide the following attributes:

25 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

26 Requests that the RM assign a new context identifier to the newly created group. The  
27 identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
28 specified in the request. Thus, the value serves as a means of identifying the group within  
29 that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

30 **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

31 Notify remaining members when another member terminates without first leaving the group.  
32

## 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\_construct** operation. The callback function will be called once all group members have called either **PMIx\_Group\_construct** or **PMIx\_Group\_construct\_nb**.

## 14.2.8 PMIx\_Group\_destruct

### Summary

Destruct a PMIx process group.

### Format

PMIx v4.0

```
pmix_status_t
PMIx_Group_destruct(const char grp[],
                   const pmix_info_t directives[],
                   size_t ndirs);
```

#### IN grp

NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the identifier of the group to be destructed (string)

#### IN directives

Array of **pmix\_info\_t** structures (array of handles)

#### IN ndirs

Number of elements in the *directives* array (**size\_t**)

Returns **PMIX\_SUCCESS** or a negative value indicating the error.

## 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.

## 14.2.9 PMIx\_Group\_destruct\_nb

### Summary

Non-blocking form of **PMIx\_Group\_destruct**.

## Format

C

```
pmix_status_t
PMIx_Group_destruct_nb(const char grp[],
                      const pmix_info_t directives[],
                      size_t ndirs,
                      pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

### IN grp

NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the identifier of the group to be destructed (string)

### IN directives

Array of `pmix_info_t` structures (array of handles)

### IN ndirs

Number of elements in the *directives* 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)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

`PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed successfully - the *cbfunc* will *not* be called.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

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

- `PMIX_SUCCESS` The operation was successfully completed.
- `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. 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

Non-blocking version of the **PMIx\_Group\_destruct** operation. The callback function will be called once all members of the group have executed either **PMIx\_Group\_destruct** or **PMIx\_Group\_destruct\_nb**.

## 14.2.10 PMIx\_Group\_invite

### Summary

Asynchronously construct a PMIx process group.

### Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_invite(const char grp[],
                  const pmix_proc_t procs[], size_t nprocs,
                  const pmix_info_t directives[], size_t ndirs,
                  pmix_info_t **results, size_t *nresult);
```

C

- IN grp**  
NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group identifier (string)
- IN procs**  
Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the processes to be invited (array of handles)
- IN nprocs**  
Number of elements in the *procs* array (**size\_t**)
- IN directives**  
Array of **pmix\_info\_t** structures (array of handles)
- IN ndirs**  
Number of elements in the *directives* array (**size\_t**)

1 **INOUT results**

2 Pointer to a location where the array of `pmix_info_t` describing the results of the  
3 operation is to be returned (pointer to handle)

4 **INOUT nresults**

5 Pointer to a `size_t` location where the number of elements in *results* is to be returned  
6 (memory reference)

7 Returns `PMIX_SUCCESS` or a negative value indicating the error.

8 **Required Attributes**

9 The following attributes are *required* to be supported by all PMIx libraries that support this  
10 operation:

11 **PMIX\_GROUP\_OPTIONAL** "`pmix.grp.opt`" (`bool`)

12 Participation is optional - do not return an error if any of the specified processes terminate  
13 without having joined. The default is `false`.

14 **PMIX\_GROUP\_FT\_COLLECTIVE** "`pmix.grp.ftcoll`" (`bool`)

15 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective  
16 operation.

17 Host environments that support this operation are *required* to provide the following attributes:

18 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "`pmix.grp.actxid`" (`bool`)

19 Requests that the RM assign a new context identifier to the newly created group. The  
20 identifier is an unsigned, `size_t` value that the RM guarantees to be unique across the range  
21 specified in the request. Thus, the value serves as a means of identifying the group within  
22 that range. If no range is specified, then the request defaults to `PMIX_RANGE_SESSION`.

23 **PMIX\_GROUP\_NOTIFY\_TERMINATION** "`pmix.grp.notterm`" (`bool`)

24 Notify remaining members when another member terminates without first leaving the group.

25 **Optional Attributes**

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

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  
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 application to either invite a replacement, declare “abort”, or choose to remove the declining process from the final group. The inviting process should also register to receive `PMIX_GROUP_INVITE_FAILED` events whenever a process fails or terminates prior to responding to the invitation. Actions taken by the inviting process in response to these events must be communicated at the end of the event handler by returning the corresponding result so that the PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level information of each other’s namespaces plus any information posted via `PMIx_Put` by the other members.

The inviting process is automatically considered the leader of the asynchronous group construction procedure and will receive all failure or termination events for invited members prior to completion. The inviting process is required to provide a `PMIX_GROUP_CONSTRUCT_COMPLETE` event once the group has been fully assembled - this event is used by the PMIx library as a trigger to release participants from their call to `PMIx_Group_join` and provides information (e.g., the final group membership) to be returned in the *results* array.

Failure of the inviting process at any time will cause a `PMIX_GROUP_LEADER_FAILED` event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the `PMIX_GROUP_LEADER` attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a `PMIX_GROUP_LEADER_SELECTED` event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

## Advice to users

Applications are not allowed to use the group in any operations until group construction is complete. This is required in order to ensure consistent knowledge of group membership across all participants.

### 14.2.11 PMIx\_Group\_invite\_nb

#### Summary

Non-blocking form of `PMIx_Group_invite`.

#### Format

*PMIx v4.0*

C

```
pmix_status_t
PMIx_Group_invite_nb(const char grp[],
                    const pmix_proc_t procs[], size_t nprocs,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

- IN grp**  
NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)
- IN procs**  
Array of `pmix_proc_t` structures containing the PMIx identifiers of the processes to be invited (array of handles)
- IN nprocs**  
Number of elements in the *procs* array (`size_t`)
- IN directives**  
Array of `pmix_info_t` structures (array of handles)
- IN ndirs**  
Number of elements in the *directives* array (`size_t`)
- IN cbfunc**  
Callback function `pmix_info_cbfunc_t` (function reference)
- IN cbdata**  
Data to be passed to the callback function (memory reference)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

1 **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed  
2 successfully - the *cbfunc* will *not* be called.

3 If none of the above return codes are appropriate, then an implementation must return either a  
4 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

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

- 7 • **PMIX\_SUCCESS** The operation succeeded and all specified members participated.
- 8 • **PMIX\_ERR\_PARTIAL\_SUCCESS** The operation succeeded but not all specified members  
9 participated - the final group membership is included in the callback function.
- 10 • **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM  
11 does not.
- 12 • a non-zero PMIx error constant indicating a reason for the request's failure.

▼----- Required Attributes -----▼

13 The following attributes are *required* to be supported by all PMIx libraries that support this  
14 operation:

15 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

16 Participation is optional - do not return an error if any of the specified processes terminate  
17 without having joined. The default is **false**.

18 **PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)

19 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective  
20 operation.

21 Host environments that support this operation are *required* to provide the following attributes:

22 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

23 Requests that the RM assign a new context identifier to the newly created group. The  
24 identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
25 specified in the request. Thus, the value serves as a means of identifying the group within  
26 that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

27 **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

28 Notify remaining members when another member terminates without first leaving the group.  
29

▲-----▲

## 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**.

## 14.2.12 PMIx\_Group\_join

### Summary

Accept an invitation to join a PMIx process group.

### Format

PMIx v4.0

```
pmix_status_t
PMIx_Group_join(const char grp[],
                const pmix_proc_t *leader,
                pmix_group_opt_t opt,
                const pmix_info_t directives[], size_t ndirs,
                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)



1 **INOUT `nresults`**

2 Pointer to a `size_t` location where the number of elements in `results` is to be returned  
3 (memory reference)

4 Returns `PMIX_SUCCESS` or a negative value indicating the error.

▼----- Required Attributes -----▼

5 There are no identified required attributes for implementers.



▼----- Optional Attributes -----▼

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

7 `PMIX_TIMEOUT` "`pmix.timeout`" (`int`)

8 Time in seconds before the specified operation should time out (zero indicating infinite) and  
9 return the `PMIX_ERR_TIMEOUT` error. Care should be taken to avoid race conditions  
10 caused by multiple layers (client, server, and host) simultaneously timing the operation.



11 **Description**

12 Respond to an invitation to join a group that is being asynchronously constructed. The process must  
13 have registered for the `PMIX_GROUP_INVITED` event in order to be notified of the invitation.  
14 When called, the event information will include the `pmix_proc_t` identifier of the process that  
15 generated the invitation along with the identifier of the group being constructed. When ready to  
16 respond, the process provides a response using either form of `PMIx_Group_join`.

▼----- Advice to users -----▼

17 Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the  
18 blocking form of this call unless it first “thread shifts” out of the handler and into its own thread  
19 context. Likewise, while it is safe to call the non-blocking form of the API from the event handler,  
20 the process *must not* block in the handler while waiting for the callback function to be called.

1 Calling this function causes the inviting process (aka the *group leader*) to be notified that the  
2 process has either accepted or declined the request. The blocking form of the API will return once  
3 the group has been completely constructed or the group's construction has failed (as described  
4 below) – likewise, the callback function of the non-blocking form will be executed upon the same  
5 conditions.

6 Failure of the leader during the call to `PMIx_Group_join` will cause a  
7 `PMIX_GROUP_LEADER_FAILED` event to be delivered to all invited participants so they can  
8 optionally declare a new leader. A new leader is identified by providing the  
9 `PMIX_GROUP_LEADER` attribute in the results array in the return of the event handler. Only one  
10 process is allowed to return that attribute, declaring itself as the new leader. Results of the leader  
11 selection will be communicated to all participants via a `PMIX_GROUP_LEADER_SELECTED`  
12 event identifying the new leader. If no leader was selected, then the status code provided in the  
13 event handler will provide an error value so the participants can take appropriate action.

14 Any participant that returns `PMIX_GROUP_CONSTRUCT_ABORT` from the leader failed event  
15 handler will cause all participants to receive an event notifying them of that status. Similarly, the  
16 leader may elect to abort the procedure by either returning `PMIX_GROUP_CONSTRUCT_ABORT`  
17 from the handler assigned to the `PMIX_GROUP_INVITE_ACCEPTED` or  
18 `PMIX_GROUP_INVITE_DECLINED` codes, or by generating an event for the abort code. Abort  
19 events will be sent to all invited participants.

### 20 14.2.13 `PMIx_Group_join_nb`

#### 21 Summary

22 Non-blocking form of `PMIx_Group_join`

#### 23 Format

*PMIx v4.0*

C

24 `pmix_status_t`

```
25 PMIx_Group_join_nb(const char grp[],  
26 const pmix_proc_t *leader,  
27 pmix_group_opt_t opt,  
28 const pmix_info_t directives[], size_t ndirs,  
29 pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

30 **IN** `grp`

31 `NULL`-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group  
32 identifier (string)

33 **IN** `leader`

34 Process that generated the invitation (handle)

1 **IN** **opt**  
 2 Accept or decline flag (`pmix_group_opt_t`)  
 3 **IN** **directives**  
 4 Array of `pmix_info_t` structures (array of handles)  
 5 **IN** **ndirs**  
 6 Number of elements in the *directives* array (`size_t`)  
 7 **IN** **cbfunc**  
 8 Callback function `pmix_info_cbfunc_t` (function reference)  
 9 **IN** **cbdata**  
 10 Data to be passed to the callback function (memory reference)

11 A successful return indicates that the request is being processed and the result will be returned in  
 12 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
 13 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

14 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

15 **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed  
 16 successfully - the *cbfunc* will *not* be called.

17 If none of the above return codes are appropriate, then an implementation must return either a  
 18 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

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

- 21 • **PMIX\_SUCCESS** The operation succeeded and group membership is in the callback function  
 22 parameters.
- 23 • **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM  
 24 does not.
- 25 • a non-zero PMIx error constant indicating a reason for the request's failure.

▼----- Required Attributes -----▼

26 There are no identified required attributes for implementers.

▲-----

▼----- Optional Attributes -----▼

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

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

▲-----

## Description

Non-blocking version of the `PMIx_Group_join` 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`.

### 14.2.13.1 Group accept/decline directives

*PMIx v4.0*

The `pmix_group_opt_t` type is a `uint8_t` value used with the `PMIx_Group_join` API to indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

`PMIX_GROUP_DECLINE` Decline the invitation.

`PMIX_GROUP_ACCEPT` Accept the invitation.

### 14.2.14 `PMIx_Group_leave`

#### Summary

Leave a `PMIx` process group.

#### Format

*PMIx v4.0*

```
pmix_status_t
PMIx_Group_leave(const char grp[],
                 const pmix_info_t directives[],
                 size_t ndirs);
```

#### IN `grp`

`NULL`-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

#### IN `directives`

Array of `pmix_info_t` structures (array of handles)

#### IN `ndirs`

Number of elements in the *directives* array (`size_t`)

Returns `PMIX_SUCCESS` or a negative value indicating the error.

#### Required Attributes

There are no identified required attributes for implementers.

## Description

Calls to `PMIx_Group_leave` (or its non-blocking form) will cause a `PMIX_GROUP_LEFT` event to be generated notifying all members of the group of the caller's departure. The function will return (or the non-blocking function will execute the specified callback function) once the event has been locally generated and is not indicative of remote receipt.

### Advice to users

The `PMIx_Group_leave` API is intended solely for asynchronous departures of individual processes from a group as it is not a scalable operation – i.e., when a process determines it should no longer be a part of a defined group, but the remainder of the group retains a valid reason to 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.

## 14.2.15 `PMIx_Group_leave_nb`

### Summary

Non-blocking form of `PMIx_Group_leave`.

### Format

*PMIx v4.0*

```
pmix_status_t
PMIx_Group_leave_nb(const char grp[],
                   const pmix_info_t directives[],
                   size_t ndirs,
                   pmix_op_cbfunc_t cbfunc,
                   void *cbdata);
```

- IN** `grp`  
NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)
- IN** `directives`  
Array of `pmix_info_t` structures (array of handles)
- IN** `ndirs`  
Number of elements in the `directives` 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)

1 A successful return indicates that the request is being processed and the result will be returned in  
2 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
3 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

4 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

5 **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed  
6 successfully - the *cbfunc* will *not* be called.

7 If none of the above return codes are appropriate, then an implementation must return either a  
8 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

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

- 11 • **PMIX\_SUCCESS** The operation succeeded - i.e., the **PMIX\_GROUP\_LEFT** event was generated.
- 12 • **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx library supports this operation, the host RM  
13 does not.
- 14 • a non-zero PMIx error constant indicating a reason for the request's failure.

#### Required Attributes

15 There are no identified required attributes for implementers.

#### Description

16 Non-blocking version of the **PMIx\_Group\_leave** operation. The callback function will be  
17 called once the event has been locally generated and is not indicative of remote receipt.  
18

# Fabric Support Definitions

---

1 As the drive for performance continues, interest has grown in scheduling algorithms that take into  
2 account network locality of the allocated resources and in optimizing collective communication  
3 patterns by structuring them to follow fabric topology. In addition, concerns over the time required  
4 to initiate execution of parallel applications and enable communication across them have grown as  
5 the size of those applications extends into the hundreds of thousands of individual processes  
6 spanning tens of thousands of nodes.

7 PMIx supports the communication part of these efforts by defining data types and attributes by  
8 which fabric endpoints and coordinates for processes and devices can be obtained from the host  
9 environment. When used in conjunction with other PMIx methods described in Chapter 17, this  
10 results in the ability of a process to obtain the fabric endpoint and coordinate of all other processes  
11 without incurring additional overhead associated with a global exchange of that information. This  
12 includes:

- 13 • Defining several interfaces specifically intended to support WLMs by providing access to  
14 information of potential use to scheduling algorithms - e.g., information on communication costs  
15 between different points on the fabric.
- 16 • Supporting hierarchical collective operations by providing the fabric coordinates for all devices  
17 on participating nodes as well as a list of the peers sharing each fabric switch. This enables one,  
18 for example, to aggregate the contribution from all processes on a node, then again across all  
19 nodes on a common switch, and finally across all switches based on detailed knowledge of the  
20 fabric location of each participant.
- 21 • Enabling the "*instant on*" paradigm to mitigate the scalable launch problem by providing each  
22 process with a rich set of information about the environment and the application, including  
23 everything required for communication between peers within the application, at time of process  
24 start of execution.

25 Meeting these needs in the case where only a single fabric device exists on each node is relatively  
26 straightforward - PMIx and the host environment provide a single endpoint for each process plus a  
27 coordinate for the device on each node, and there is no uncertainty regarding the endpoint each  
28 process will use. Extending this to the multiple device per node case is more difficult as the choice  
29 of endpoint by any given process cannot be known in advance, and questions arise regarding  
30 reachability between devices on different nodes. Resolving these ambiguities without requiring a  
31 global operation requires that PMIx provide both (a) an endpoint for each application process on  
32 each of its local devices; and (b) the fabric coordinates of all remote and local devices on  
33 participating nodes. It also requires that each process open all of its assigned endpoints as the  
34 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 Modex* 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_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_DEVICE_ID` "`pmix.dev.id`" (`string`)  
4 System-wide UUID or node-local OS name of a particular 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.

- 1 • **PMIX\_FABRIC\_DEVICE\_MTU** "pmix.fabdev.mtu" (**size\_t**)  
2 The maximum transfer unit of link level frames or packets, in bytes.
- 3 • **PMIX\_FABRIC\_DEVICE\_SPEED** "pmix.fabdev.speed" (**size\_t**)  
4 The active link data rate, given in bits per second.
- 5 • **PMIX\_FABRIC\_DEVICE\_STATE** "pmix.fabdev.state" (**pmix\_link\_state\_t**)  
6 The last available physical port state for the specified device. Possible values are  
7 **PMIX\_LINK\_STATE\_UNKNOWN**, **PMIX\_LINK\_DOWN**, and **PMIX\_LINK\_UP**, to  
8 indicate if the port state is unknown or not applicable (unknown), inactive (down), or  
9 active (up).
- 10 • **PMIX\_FABRIC\_DEVICE\_TYPE** "pmix.fabdev.type" (**string**)  
11 Specifies the type of fabric interface currently active on the device, such as Ethernet or  
12 InfiniBand.

13 The remainder of this chapter details the events, data types, attributes, and APIs associated with  
14 fabric-related operations.

## 15 15.1 Fabric Support Events

16 The following events are defined for use in fabric-related operations.

17 **PMIX\_FABRIC\_UPDATE\_PENDING** The PMIx server library has been alerted to a change in  
18 the fabric that requires updating of one or more registered **pmix\_fabric\_t** objects.

19 **PMIX\_FABRIC\_UPDATED** The PMIx server library has completed updating the entries of all  
20 affected **pmix\_fabric\_t** objects registered with the library. Access to the entries of those  
21 objects may now resume.

22 **PMIX\_FABRIC\_UPDATE\_ENDPOINTS** Endpoint assignments have been updated, usually in  
23 response to migration or restart of a process. Clients should use **PMIx\_Get** to update any  
24 internally cached connections.

## 25 15.2 Fabric Support Datatypes

26 Several datatype definitions have been created to support fabric-related operations and information.

### 27 15.2.1 Fabric Endpoint Structure

28 The **pmix\_endpoint\_t** structure contains an assigned endpoint for a given fabric device.

PMIx v4.0

```

1 typedef struct pmix_endpoint {
2     char *uuid;
3     char *osname;
4     pmix_byte_object_t endpt;
5 } pmix_endpoint_t;

```

The *uuid* field contains the UUID of the fabric device, the *osname* is the local operating system's name for the device, and the *endpt* field contains a fabric vendor-specific object identifying the communication endpoint assigned to the process.

## 15.2.2 Fabric endpoint support macros

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

### Initialize the endpoint structure

Initialize the `pmix_endpoint_t` fields.

*PMIx v4.0*

```

13 PMIX_ENDPOINT_CONSTRUCT (m)

```

**IN** m

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

### Destruct the endpoint structure

Destruct the `pmix_endpoint_t` fields.

*PMIx v4.0*

```

18 PMIX_ENDPOINT_DESTRUCT (m)

```

**IN** m

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

### Create an endpoint array

Allocate and initialize a `pmix_endpoint_t` array.

*PMIx v4.0*

```

23 PMIX_ENDPOINT_CREATE (m, n)

```

**INOUT** m

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

**IN** n

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*

▼  ▼

3 **PMIX\_ENDPOINT\_FREE**(m, n)

▲  ▲

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`)

## 8 15.2.3 Fabric Coordinate Structure

9 The `pmix_coord_t` structure describes the fabric coordinates of a specified device in a given  
10 view.

*PMIx v4.0*

▼  ▼

```
11 typedef struct pmix_coord {  
12     pmix_coord_view_t view;  
13     uint32_t *coord;  
14     size_t dims;  
15 } pmix_coord_t;
```

▲  ▲

16 All coordinate values shall be expressed as unsigned integers due to their units being defined in  
17 fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity  
18 and not relative communication distance.

▼ **Advice to PMIx library implementers** ▼

19 Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the  
20 coordinate data is to be stored within the PMIx library. Implementers are free to store the  
21 coordinate in whatever format they choose.

▲  ▲

22 A fabric coordinate is associated with a given fabric device and must be unique within a given view.  
23 Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates  
24 are logically grouped within the *node* realm (as described in Section 6.1) and can be retrieved per  
25 the rules detailed in Section 6.1.5.

## 26 15.2.4 Fabric coordinate support macros

27 The following macros are provided to support the `pmix_coord_t` structure.

1 **Initialize the coord structure**

2 Initialize the `pmix_coord_t` fields.



3 **PMIX\_COORD\_CONSTRUCT (m)**



4 **IN m**

5 Pointer to the structure to be initialized (pointer to `pmix_coord_t`)

6 **Destruct the coord structure**

7 Destruct the `pmix_coord_t` fields.

*PMIx v4.0*



8 **PMIX\_COORD\_DESTRUCT (m)**



9 **IN m**

10 Pointer to the structure to be destructed (pointer to `pmix_coord_t`)

11 **Create a coord array**

12 Allocate and initialize a `pmix_coord_t` array.

*PMIx v4.0*



13 **PMIX\_COORD\_CREATE (m, n)**



14 **INOUT m**

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

16 **IN n**

17 Number of structures to be allocated (`size_t`)

18 **Release a coord array**

19 Release an array of `pmix_coord_t` structures.

*PMIx v4.0*



20 **PMIX\_COORD\_FREE (m, n)**



21 **IN m**

22 Pointer to the array of `pmix_coord_t` structures (handle)

23 **IN n**

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

## 1 15.2.5 Fabric Geometry Structure

2 The `pmix_geometry_t` structure describes the fabric coordinates of a specified device.

```
3 typedef struct pmix_geometry {  
4     size_t fabric;  
5     char *uuid;  
6     char *osname;  
7     pmix_coord_t *coordinates;  
8     size_t ncoords;  
9 } pmix_geometry_t;
```

10 All coordinate values shall be expressed as unsigned integers due to their units being defined in  
11 fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity  
12 and not relative communication distance.

### Advice to PMIx library implementers

13 Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the  
14 coordinate data is to be stored within the PMIx library. Implementers are free to store the  
15 coordinate in whatever format they choose.

16 A fabric coordinate is associated with a given fabric device and must be unique within a given view.  
17 Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates  
18 are logically grouped within the *node* realm (as described in Section 6.1) and can be retrieved per  
19 the rules detailed in Section 6.1.5.

## 20 15.2.6 Fabric geometry support macros

21 The following macros are provided to support the `pmix_geometry_t` structure.

### Initialize the geometry structure

22 Initialize the `pmix_geometry_t` fields.

23 *PMIx v4.0*

```
24 #define PMIX_GEOMETRY_CONSTRUCT(m)
```

25 **IN** *m*

26 Pointer to the structure to be initialized (pointer to `pmix_geometry_t`)

1 **Destruct the geometry structure**

2 Destruct the `pmix_geometry_t` fields.



3 **PMIX\_GEOMETRY\_DESTRUCT (m)**



4 **IN m**

5 Pointer to the structure to be destructed (pointer to `pmix_geometry_t`)

6 **Create a geometry array**

7 Allocate and initialize a `pmix_geometry_t` array.

*PMIx v4.0*



8 **PMIX\_GEOMETRY\_CREATE (m, n)**



9 **INOUT m**

10 Address where the pointer to the array of `pmix_geometry_t` structures shall be stored  
11 (handle)

12 **IN n**

13 Number of structures to be allocated (`size_t`)

14 **Release a geometry array**

15 Release an array of `pmix_geometry_t` structures.

*PMIx v4.0*



16 **PMIX\_GEOMETRY\_FREE (m, n)**



17 **IN m**

18 Pointer to the array of `pmix_geometry_t` structures (handle)

19 **IN n**

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

21 **15.2.7 Fabric Coordinate Views**

*PMIx v4.0*



```
22 typedef uint8_t pmix_coord_view_t;  
23 #define PMIX_COORD_VIEW_UNDEF 0x00  
24 #define PMIX_COORD_LOGICAL_VIEW 0x01  
25 #define PMIX_COORD_PHYSICAL_VIEW 0x02
```

1 Fabric coordinates can be reported based on different *views* according to user preference at the time  
2 of request. The following views have been defined:

3 **PMIX\_COORD\_VIEW\_UNDEF** The coordinate view has not been defined.

4 **PMIX\_COORD\_LOGICAL\_VIEW** The coordinates are provided in a *logical* view, typically  
5 given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by  
6 the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains,  
7 and other similar factors employed by that fabric.

8 **PMIX\_COORD\_PHYSICAL\_VIEW** The coordinates are provided in a *physical* view based on  
9 the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative  
10 position of that interface on the specific fabric cabling.

11 If the requester does not specify a view, coordinates shall default to the *logical* view.

## 12 15.2.8 Fabric Link State

13 The `pmix_link_state_t` is a `uint32_t` type for fabric link states.

14 *PMIx v4.0*

```
14 typedef uint8_t pmix_link_state_t;
```

15 The following constants can be used to set a variable of the type `pmix_link_state_t`. All  
16 definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state  
17 values start at zero.

18 **PMIX\_LINK\_STATE\_UNKNOWN** The port state is unknown or not applicable.

19 **PMIX\_LINK\_DOWN** The port is inactive.

20 **PMIX\_LINK\_UP** The port is active.

## 21 15.2.9 Fabric Operation Constants

22 *PMIx v4.0*

23 The `pmix_fabric_operation_t` data type is an enumerated type for specifying fabric  
operations used in the PMIx server module's `pmix_server_fabric_fn_t` API.

24 **PMIX\_FABRIC\_REQUEST\_INFO** Request information on a specific fabric - if the fabric isn't  
25 specified as per `PMIx Fabric register`, then return information on the default fabric of  
26 the overall system. Information to be returned is described in `pmix_fabric_t`.

27 **PMIX\_FABRIC\_UPDATE\_INFO** Update information on a specific fabric - the index of the  
28 fabric (`PMIX_FABRIC_INDEX`) to be updated must be provided.



## 1 15.2.10 Fabric registration structure

2 The `pmix_fabric_t` structure is used by a WLM to interact with fabric-related PMIx interfaces,  
3 and to provide information about the fabric for use in scheduling algorithms or other purposes.

```
4 typedef struct pmix_fabric_s {  
5     char *name;  
6     size_t index;  
7     pmix_info_t *info;  
8     size_t ninfo;  
9     void *module;  
10 } pmix_fabric_t;
```

11 Note that in this structure:

- 12 • *name* is an optional user-supplied string name identifying the fabric being referenced by this  
13 struct. If provided, the field must be a **NULL**-terminated string composed of standard  
14 alphanumeric values supported by common utilities such as *strcmp*;
- 15 • *index* is a PMIx-provided number identifying this object;
- 16 • *info* is an array of `pmix_info_t` containing information (provided by the PMIx library) about  
17 the fabric;
- 18 • *ninfo* is the number of elements in the *info* array;
- 19 • *module* points to an opaque object reserved for use by the PMIx server library.

20 Note that only the *name* field is provided by the user - all other fields are provided by the PMIx  
21 library and must not be modified by the user. The *info* array contains a varying amount of  
22 information depending upon both the PMIx implementation and information available from the  
23 fabric vendor. At a minimum, it must contain (ordering is arbitrary):

### Required Attributes

24 **PMIX\_FABRIC\_VENDOR** "`pmix.fab.vndr`" (**string**)

25 Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

26 **PMIX\_FABRIC\_IDENTIFIER** "`pmix.fab.id`" (**string**)

27 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

28 **PMIX\_FABRIC\_NUM\_DEVICES** "`pmix.fab.nverts`" (**size\_t**)

29 Total number of fabric devices in the overall system - corresponds to the number of rows or  
30 columns in the cost matrix.

31 and may optionally contain one or more of the following:

## Optional Attributes

1 **PMIX\_FABRIC\_COST\_MATRIX** "pmix.fab.cm" (pointer)

2 Pointer to a two-dimensional square array of point-to-point relative communication costs  
3 expressed as `uint16_t` values.

4 **PMIX\_FABRIC\_GROUPS** "pmix.fab.grps" (string)

5 A string delineating the group membership of nodes in the overall system, where each fabric  
6 group consists of the group number followed by a colon and a comma-delimited list of nodes  
7 in that group, with the groups delimited by semi-colons (e.g.,  
8 `0:node000,node002,node004,node006;1:node001,node003,`  
9 `node005,node007`)

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\_PLANE** "pmix.fab.plane" (string)

16 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
17 for information, specifies the plane whose information is to be returned. When used directly  
18 as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
19 planes in the overall system.

20 **PMIX\_FABRIC\_SHAPE** "pmix.fab.shape" (`pmix_data_array_t*`)

21 The size of each dimension in the specified fabric plane/view, returned in a  
22 `pmix_data_array_t` containing an array of `uint32_t` values. The size is defined as  
23 the number of elements present in that dimension - e.g., the number of devices in one  
24 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of  
25 each plane in the overall system will be returned in a `pmix_data_array_t` array where  
26 each element is itself a two-element array containing the **PMIX\_FABRIC\_PLANE** followed  
27 by that plane's fabric shape. Default is to provide the shape in *logical* view.

28 **PMIX\_FABRIC\_SHAPE\_STRING** "pmix.fab.shapestr" (string)

29 Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the  
30 shape of each plane in the overall system will be returned in a `pmix_data_array_t` array  
31 where each element is itself a two-element array containing the **PMIX\_FABRIC\_PLANE**  
32 followed by that plane's fabric shape string. Default is to provide the shape in *logical* view.

33 While unusual due to scaling issues, implementations may include an array of

34 **PMIX\_FABRIC\_DEVICE** elements describing the device information for each device in the  
35 overall system. Each element shall contain a `pmix_data_array_t` of `pmix_info_t` values  
36 describing the device. Each array may contain one or more of the following (ordering is arbitrary):

37 **PMIX\_FABRIC\_DEVICE\_NAME** "pmix.fabdev.nm" (string)

1 The operating system name associated with the device. This may be a logical fabric interface  
2 name (e.g. "eth0" or "eno1") or an absolute filename.

3 **PMIX\_FABRIC\_DEVICE\_VENDOR** "pmix.fabdev.vndr" (string)

4 Indicates the name of the vendor that distributes the device.

5 **PMIX\_DEVICE\_ID** "pmix.dev.id" (string)

6 System-wide UUID or node-local OS name of a particular device.

7 **PMIX\_HOSTNAME** "pmix.hname" (char\*)

8 Name of the host, as returned by the `gethostname` utility or its equivalent.

9 **PMIX\_FABRIC\_DEVICE\_DRIVER** "pmix.fabdev.driver" (string)

10 The name of the driver associated with the device.

11 **PMIX\_FABRIC\_DEVICE\_FIRMWARE** "pmix.fabdev.fmwr" (string)

12 The device's firmware version.

13 **PMIX\_FABRIC\_DEVICE\_ADDRESS** "pmix.fabdev.addr" (string)

14 The primary link-level address associated with the device, such as a MAC address. If  
15 multiple addresses are available, only one will be reported.

16 **PMIX\_FABRIC\_DEVICE\_MTU** "pmix.fabdev.mtu" (size\_t)

17 The maximum transfer unit of link level frames or packets, in bytes.

18 **PMIX\_FABRIC\_DEVICE\_SPEED** "pmix.fabdev.speed" (size\_t)

19 The active link data rate, given in bits per second.

20 **PMIX\_FABRIC\_DEVICE\_STATE** "pmix.fabdev.state" (pmix\_link\_state\_t)

21 The last available physical port state for the specified device. Possible values are  
22 **PMIX\_LINK\_STATE\_UNKNOWN**, **PMIX\_LINK\_DOWN**, and **PMIX\_LINK\_UP**, to indicate  
23 if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

24 **PMIX\_FABRIC\_DEVICE\_TYPE** "pmix.fabdev.type" (string)

25 Specifies the type of fabric interface currently active on the device, such as Ethernet or  
26 InfiniBand.

27 **PMIX\_FABRIC\_DEVICE\_BUS\_TYPE** "pmix.fabdev.btyp" (string)

28 The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

29 **PMIX\_FABRIC\_DEVICE\_PCI\_DEVID** "pmix.fabdev.pcidevid" (string)

30 A node-level unique identifier for a PCI device. Provided only if the device is located on a  
31 PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of  
32 the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in  
33 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The  
34 combination of node identifier (**PMIX\_HOSTNAME** or **PMIX\_NODEID**) and  
35 **PMIX\_FABRIC\_DEVICE\_PCI\_DEVID** shall be unique within the overall system.

▲-----▲

## 1 15.2.10.1 Initialize the fabric structure

2 Initialize the `pmix_fabric_t` fields.

PMIx v4.0

3 **PMIX\_FABRIC\_CONSTRUCT** (m)

4 **IN** m

5 Pointer to the structure to be initialized (pointer to `pmix_fabric_t`)

## 6 15.3 Fabric Support Attributes

7 The following attribute is used by the PMIx server library supporting the system's WLM to indicate  
8 that it wants access to the fabric support functions:

9 **PMIX\_SERVER\_SCHEDULER** "`pmix.srv.sched`" (**bool**)

10 Server is supporting system scheduler and desires access to appropriate WLM-supporting  
11 features. Indicates that the library is to be initialized for scheduler support.

12 The following attributes may be returned in response to fabric-specific APIs or queries (e.g.,  
13 `PMIx_Get` or `PMIx_Query_info`). These attributes are not related to a specific *data realm* (as  
14 described in Section 6.1) - the `PMIx_Get` function shall therefore ignore the value in its *proc*  
15 process identifier argument when retrieving these values.

16 **PMIX\_FABRIC\_COST\_MATRIX** "`pmix.fab.cm`" (**pointer**)

17 Pointer to a two-dimensional square array of point-to-point relative communication costs  
18 expressed as `uint16_t` values.

19 **PMIX\_FABRIC\_GROUPS** "`pmix.fab.grps`" (**string**)

20 A string delineating the group membership of nodes in the overall system, where each fabric  
21 group consists of the group number followed by a colon and a comma-delimited list of nodes  
22 in that group, with the groups delimited by semi-colons (e.g.,  
23 `0:node000,node002,node004,node006;1:node001,node003,`  
24 `node005,node007`)

25 **PMIX\_FABRIC\_PLANE** "`pmix.fab.plane`" (**string**)

26 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
27 for information, specifies the plane whose information is to be returned. When used directly  
28 as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
29 planes in the overall system.

30 **PMIX\_FABRIC\_SWITCH** "`pmix.fab.switch`" (**string**)

31 ID string of a fabric switch. When used as a modifier in a request for information, specifies  
32 the switch whose information is to be returned. When used directly as a key in a request,  
33 returns a `pmix_data_array_t` of string identifiers for all fabric switches in the overall  
34 system.

1 The following attributes may be returned in response to queries (e.g., **PMIx\_Get** or  
2 **PMIx\_Query\_info**). A qualifier (e.g., **PMIX\_FABRIC\_INDEX**) identifying the fabric whose  
3 value is being referenced must be provided for queries on systems supporting more than one fabric  
4 when values for the non-default fabric are requested. These attributes are not related to a specific  
5 *data realm* (as described in Section 6.1) - the **PMIx\_Get** function shall therefore ignore the value  
6 in its *proc* process identifier argument when retrieving these values.

7 **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (**string**)

8 Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

9 **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (**string**)

10 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

11 **PMIX\_FABRIC\_INDEX** "pmix.fab.idx" (**size\_t**)

12 The index of the fabric as returned in **pmix\_fabric\_t**.

13 **PMIX\_FABRIC\_NUM\_DEVICES** "pmix.fab.nverts" (**size\_t**)

14 Total number of fabric devices in the overall system - corresponds to the number of rows or  
15 columns in the cost matrix.

16 **PMIX\_FABRIC\_DIMS** "pmix.fab.dims" (**uint32\_t**)

17 Number of dimensions in the specified fabric plane/view. If no plane is specified in a  
18 request, then the dimensions of all planes in the overall system will be returned as a  
19 **pmix\_data\_array\_t** containing an array of **uint32\_t** values. Default is to provide  
20 dimensions in *logical* view.

21 **PMIX\_FABRIC\_SHAPE** "pmix.fab.shape" (**pmix\_data\_array\_t\***)

22 The size of each dimension in the specified fabric plane/view, returned in a  
23 **pmix\_data\_array\_t** containing an array of **uint32\_t** values. The size is defined as  
24 the number of elements present in that dimension - e.g., the number of devices in one  
25 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of  
26 each plane in the overall system will be returned in a **pmix\_data\_array\_t** array where  
27 each element is itself a two-element array containing the **PMIX\_FABRIC\_PLANE** followed  
28 by that plane's fabric shape. Default is to provide the shape in *logical* view.

29 **PMIX\_FABRIC\_SHAPE\_STRING** "pmix.fab.shapestr" (**string**)

30 Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the  
31 shape of each plane in the overall system will be returned in a **pmix\_data\_array\_t** array  
32 where each element is itself a two-element array containing the **PMIX\_FABRIC\_PLANE**  
33 followed by that plane's fabric shape string. Default is to provide the shape in *logical* view.

34 The following attributes are related to the *node realm* (as described in Section 6.1.5) and are  
35 retrieved according to those rules.

36 **PMIX\_FABRIC\_DEVICES** "pmix.fab.devs" (**pmix\_data\_array\_t**)

37 Array of **pmix\_info\_t** containing information for all devices on the specified node. Each  
38 element of the array will contain a **PMIX\_FABRIC\_DEVICE** entry, which in turn will  
39 contain an array of information on a given device.

40 **PMIX\_FABRIC\_COORDINATES** "pmix.fab.coords" (**pmix\_data\_array\_t**)

1 Array of `pmix_geometry_t` fabric coordinates for devices on the specified node. The  
2 array will contain the coordinates of all devices on the node, including values for all  
3 supported coordinate views. The information for devices on the local node shall be provided  
4 if the node is not specified in the request.

5 **PMIX\_FABRIC\_DEVICE** "`pmix.fabdev`" (`pmix_data_array_t`)

6 An array of `pmix_info_t` describing a particular fabric device using one or more of the  
7 attributes defined below. The first element in the array shall be the `PMIX_DEVICE_ID` of  
8 the device.

9 **PMIX\_FABRIC\_DEVICE\_INDEX** "`pmix.fabdev.idx`" (`uint32_t`)

10 Index of the device within an associated communication cost matrix.

11 **PMIX\_FABRIC\_DEVICE\_NAME** "`pmix.fabdev.nm`" (`string`)

12 The operating system name associated with the device. This may be a logical fabric interface  
13 name (e.g. "eth0" or "eno1") or an absolute filename.

14 **PMIX\_FABRIC\_DEVICE\_VENDOR** "`pmix.fabdev.vndr`" (`string`)

15 Indicates the name of the vendor that distributes the device.

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\_VENDORID** "`pmix.fabdev.vendid`" (`string`)

19 This is a vendor-provided identifier for the device or product.

20 **PMIX\_FABRIC\_DEVICE\_DRIVER** "`pmix.fabdev.driver`" (`string`)

21 The name of the driver associated with the device.

22 **PMIX\_FABRIC\_DEVICE\_FIRMWARE** "`pmix.fabdev.fmwr`" (`string`)

23 The device's firmware version.

24 **PMIX\_FABRIC\_DEVICE\_ADDRESS** "`pmix.fabdev.addr`" (`string`)

25 The primary link-level address associated with the device, such as a MAC address. If  
26 multiple addresses are available, only one will be reported.

27 **PMIX\_FABRIC\_DEVICE\_COORDINATES** "`pmix.fab.coord`" (`pmix_geometry_t`)

28 The `pmix_geometry_t` fabric coordinates for the device, including values for all  
29 supported coordinate views.

30 **PMIX\_FABRIC\_DEVICE\_MTU** "`pmix.fabdev.mtu`" (`size_t`)

31 The maximum transfer unit of link level frames or packets, in bytes.

32 **PMIX\_FABRIC\_DEVICE\_SPEED** "`pmix.fabdev.speed`" (`size_t`)

33 The active link data rate, given in bits per second.

34 **PMIX\_FABRIC\_DEVICE\_STATE** "`pmix.fabdev.state`" (`pmix_link_state_t`)

35 The last available physical port state for the specified device. Possible values are  
36 `PMIX_LINK_STATE_UNKNOWN`, `PMIX_LINK_DOWN`, and `PMIX_LINK_UP`, to indicate  
37 if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

38 **PMIX\_FABRIC\_DEVICE\_TYPE** "`pmix.fabdev.type`" (`string`)

39 Specifies the type of fabric interface currently active on the device, such as Ethernet or  
40 InfiniBand.

41 **PMIX\_FABRIC\_DEVICE\_PCI\_DEVID** "`pmix.fabdev.pcidevid`" (`string`)

42 A node-level unique identifier for a PCI device. Provided only if the device is located on a  
43 PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of

1 the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in  
2 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The  
3 combination of node identifier ([PMIX\\_HOSTNAME](#) or [PMIX\\_NODEID](#)) and  
4 [PMIX\\_FABRIC\\_DEVICE\\_PCI\\_DEVID](#) shall be unique within the overall system.

5 The following attributes are related to the *process realm* (as described in Section 6.1.4) and are  
6 retrieved according to those rules.

7 **PMIX\_FABRIC\_ENDPT** "[pmix.fab.endpt](#)" ([pmix\\_data\\_array\\_t](#))

8 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given  
9 process (e.g., in the case where multiple devices are associated with a package to which the  
10 process is bound), the returned values will be provided in a [pmix\\_data\\_array\\_t](#) of  
11 [pmix\\_endpoint\\_t](#) elements.

12 The following attributes are related to the *job realm* (as described in Section 6.1.2) and are retrieved  
13 according to those rules. Note that distances to fabric devices are retrieved using the  
14 [PMIX\\_DEVICE\\_DISTANCES](#) key with the appropriate [pmix\\_device\\_type\\_t](#) qualifier.

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\\_DEVICE\\_ID](#) of the local fabric device, the  
20 [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 15.4 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.



## 1 15.4.1 PMIx\_Fabric\_register

### 2 Summary

3 Register for access to fabric-related information.

### 4 Format

*PMIx v4.0*

```
pmix_status_t
PMIx_Fabric_register(pmix_fabric_t *fabric,
                    const pmix_info_t directives[],
                    size_t ndirs);
```

### 9 INOUT fabric

10 address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at  
11 will - PMIx does not utilize this field (handle)

### 12 IN directives

13 an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**,  
14 then the highest priority available fabric will be used (array of handles)

### 15 IN ndirs

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

17 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### Required Attributes

18 The following directives are required to be supported by all PMIx libraries to aid users in  
19 identifying the fabric whose data is being sought:

#### 20 **PMIX\_FABRIC\_PLANE** "pmix.fab.plane" (string)

21 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
22 for information, specifies the plane whose information is to be returned. When used directly  
23 as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
24 planes in the overall system.

#### 25 **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (string)

26 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

#### 27 **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (string)

28 Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.



## Description

Register for access to fabric-related information, including the communication cost matrix. This call must be made prior to requesting information from a fabric. The caller may request access to a particular fabric using the vendor, type, or identifier, or to a specific *fabric plane* via the `PMIX_FABRIC_PLANE` attribute - otherwise, information for the default fabric will be returned. Upon successful completion of the call, information will have been filled into the fields of the provided *fabric* structure.

For performance reasons, the PMIx library does not provide thread protection for accessing the information in the `pmix_fabric_t` structure. Instead, the PMIx implementation shall provide two methods for coordinating updates to the provided fabric information:

- Users may periodically poll for updates using the `PMIx_Fabric_update` API
- Users may register for `PMIX_FABRIC_UPDATE_PENDING` events indicating that an update to the cost matrix is pending. When received, users are required to terminate or pause any actions involving access to the cost matrix before returning from the event. Completion of the `PMIX_FABRIC_UPDATE_PENDING` event handler indicates to the PMIx library that the fabric object's entries are available for updating. This may include releasing and re-allocating memory as the number of vertices may have changed (e.g., due to addition or removal of one or more devices). When the update has been completed, the PMIx library will generate a `PMIX_FABRIC_UPDATED` event indicating that it is safe to begin using the updated fabric object(s).

There is no requirement that the caller exclusively use either one of these options. For example, the user may choose to both register for fabric update events, but poll for an update prior to some critical operation.

## 15.4.2 `PMIx_Fabric_register_nb`

### Summary

Register for access to fabric-related information.

### Format

*PMIx v4.0*

`pmix_status_t`

```
PMIx_Fabric_register_nb(pmix_fabric_t *fabric,  
                        const pmix_info_t directives[],  
                        size_t ndirs,  
                        pmix_op_cbfunc_t cbfunc, void *cbdata);
```

### 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)

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       A successful return indicates that the request is being processed and the result will be returned in  
 11       the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
 12       from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

13       Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- 14       • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
 15       returned *success* - the *cbfunc* will not be called

16       If none of the above return codes are appropriate, then an implementation must return either a  
 17       general PMIx error code or an implementation defined error code as described in Section 3.1.1.

18       **Description**

19       Non-blocking form of **PMIx\_Fabric\_register**. The caller is not allowed to access the  
 20       provided `pmix_fabric_t` until the callback function has been executed, at which time the fabric  
 21       information will have been loaded into the provided structure.

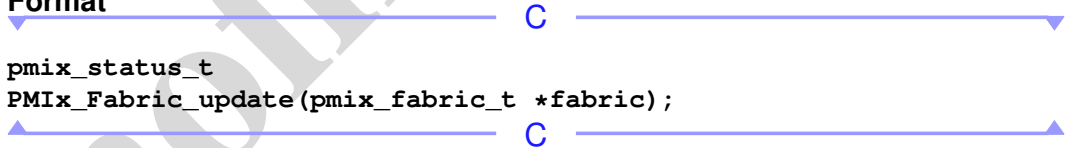
22       **15.4.3 PMIx\_Fabric\_update**

23       **Summary**

24       Update fabric-related information.

25       **Format**

*PMIx v4.0*



```
pmix_status_t
PMIx_Fabric_update(pmix_fabric_t *fabric);
```

28       **INOUT fabric**

29       address of a `pmix_fabric_t` (backed by storage) (handle)

30       Returns **PMIX\_SUCCESS** or a negative value indicating the error.

31       **Description**

32       Update fabric-related information. This call can be made at any time to request an update of the  
 33       fabric information contained in the provided `pmix_fabric_t` object. The caller is not allowed to  
 34       access the provided `pmix_fabric_t` until the call has returned. Upon successful return, the  
 35       information fields in the *fabric* structure will have been updated.

## 1 15.4.4 PMIx\_Fabric\_update\_nb

### 2 Summary

3 Update fabric-related information.

### 4 Format

PMIx v4.0

C

```
5 pmix_status_t  
6 PMIx_Fabric_update_nb(pmix_fabric_t *fabric,  
7 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

### 8 INOUT fabric

9 address of a [pmix\\_fabric\\_t](#) (handle)

### 10 IN cbfunc

11 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

### 12 IN cbdata

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

14 A successful return indicates that the request is being processed and the result will be returned in  
15 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
16 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

17 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- 18 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
19 returned *success* - the *cbfunc* will not be called

20 If none of the above return codes are appropriate, then an implementation must return either a  
21 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 22 Description

23 Non-blocking form of [PMIx\\_Fabric\\_update](#). The caller is not allowed to access the provided  
24 [pmix\\_fabric\\_t](#) until the callback function has been executed, at which time the fields in the  
25 provided *fabric* structure will have been updated.

## 26 15.4.5 PMIx\_Fabric\_deregister

### 27 Summary

28 Deregister a fabric object.

## Format

C

`pmix_status_t`

```
PMIx_Fabric_deregister(pmix_fabric_t *fabric);
```

C

**IN fabric**

address of a `pmix_fabric_t` (handle)

Returns `PMIX_SUCCESS` or a negative value indicating the error.

## 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.

### 15.4.6 PMIx\_Fabric\_deregister\_nb

## Summary

Deregister a fabric object.

## Format

PMIx v4.0

C

```
pmix_status_t PMIx_Fabric_deregister_nb(pmix_fabric_t *fabric,  
                                         pmix_op_cbfunc_t cbfunc,  
                                         void *cbdata);
```

C

**IN 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)

A successful return indicates that the request is being processed and the result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API. The callback function, `cbfunc`, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the `cbfunc` will not be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Description

Non-blocking form of `PMIx_Fabric_deregister`. Provided `fabric` must not be accessed until after callback function has been executed.

## CHAPTER 16

# 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 OS  
6 methods to determine the effective UID/GID of the process requesting the connection. PMIx  
7 implementations shall not rely on any values reported by the client process itself. The effective  
8 UID/GID reported by the OS is compared to the values provided by the host during registration - if  
9 the values fail to match, the PMIx server is required to drop the connection request. This ensures  
10 that the PMIx server does not allow connection from a client that doesn't at least meet some  
11 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\\_connected2\\_fn\\_t](#) interface, if provided. The host may perform  
17 any additional checks and operations before responding with either [PMIX\\_SUCCESS](#) to indicate  
18 that 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 16.1 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 16.1.1 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 A successful return indicates that the credential has been returned in the provided  
20 `pmix_byte_object_t`.

21 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### Required Attributes

22 There are no required attributes for this API. Note that implementations may choose to internally  
23 execute integration for some security environments (e.g., directly contacting a *munge* server).

24 Implementations that support the operation but cannot directly process the client's request must  
25 pass any attributes that are provided by the client to the host environment for processing. In  
26 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
27 library to the host environment:

28 `PMIX_USERID` "pmix.euid" (`uint32_t`)

29 Effective user ID of the connecting process.

30 `PMIX_GRPID` "pmix.egid" (`uint32_t`)

31 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.

## 16.1.2 PMIx\_Get\_credential\_nb

### Summary

Request a credential from the PMIx server library or the host environment.

### Format

PMIx v3.0

```
pmix_status_t
PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,
                      pmix_credential_cbfunc_t cbfunc,
                      void *cbdata);
```

#### 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)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed successfully - the *cbfunc* will *not* be called.

1 If none of the above return codes are appropriate, then an implementation must return either a  
2 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Required Attributes

3 There are no required attributes for this API. Note that implementations may choose to internally  
4 execute integration for some security environments (e.g., directly contacting a *munge* server).

5 Implementations that support the operation but cannot directly process the client's request must  
6 pass any attributes that are provided by the client to the host environment for processing. In  
7 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
8 library to the host environment:

9 **PMIX\_USERID** "pmix.euid" (uint32\_t)

10 Effective user ID of the connecting process.

11 **PMIX\_GRPID** "pmix.egid" (uint32\_t)

12 Effective group ID of the connecting process.

### Optional Attributes

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

18 Request a credential from the PMIx server library or the host environment. This version of the API  
19 is generally preferred in scenarios where the host environment may have to contact a remote  
20 credential service. Thus, provision is made for the system to return additional information (e.g., the  
21 identity of the issuing agent) outside of the credential itself and visible to the application.  
22

## 16.1.3 Credential Attributes

24 The following attributes are defined to support credential operations:

25 **PMIX\_CRED\_TYPE** "pmix.sec.ctype" (char\*)

26 When passed in **PMIx\_Get\_credential**, a prioritized, comma-delimited list of desired  
27 credential types for use in environments where multiple authentication mechanisms may be  
28 available. When returned in a callback function, a string identifier of the credential type.

29 **PMIX\_CRYPTO\_KEY** "pmix.sec.key" (pmix\_byte\_object\_t)

30 Blob containing crypto key.



## 1 16.2 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 16.2.1 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*

C

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);
```

C

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 A successful return indicates that the credential was valid and any information it contained was  
25 successfully processed. Details of the result will be returned in the *results* array.

26 Returns `PMIX_SUCCESS` or a negative value indicating the error.

#### Required Attributes

27 There are no required attributes for this API. Note that implementations may choose to internally  
28 execute integration for some security environments (e.g., directly contacting a *munge* server).

29 Implementations that support the operation but cannot directly process the client's request must  
30 pass any attributes that are provided by the client to the host environment for processing. In  
31 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
32 library to the host environment:

33 `PMIX_USERID` "pmix.euid" (`uint32_t`)

1 Effective user ID of the connecting process.

2 **PMIX\_GRPID** "pmix.egid" (uint32\_t)

3 Effective group ID of the connecting process.



### 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.



### 9 Description

10 Request validation of a credential by the PMIx server library or the host environment.

## 11 16.2.2 PMIx\_Validate\_credential\_nb

### 12 Summary

13 Request validation of a credential by the PMIx server library or the host environment. Provision is  
14 made for the system to return additional information regarding possible authorization limitations  
15 beyond simple authentication.

## Format

C

```
pmix_status_t
PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
                           const pmix_info_t info[], size_t ninfo,
                           pmix_validation_cbfunc_t cbfunc,
                           void *cbdata);
```

C

### IN cred

Pointer to `pmix_byte_object_t` containing the credential (handle)

### 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 result (`pmix_validation_cbfunc_t` function reference)

### IN cbdata

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

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when `PMIX_SUCCESS` is returned.

Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

`PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed successfully - the *cbfunc* will *not* be called.

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Required Attributes

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

`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.

## 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 validation of a credential by the PMIx server library or the host environment. This version of the API is generally preferred in scenarios where the host environment may have to contact a remote credential service. Provision is made for the system to return additional information (e.g., possible authorization limitations) beyond simple authentication.

## CHAPTER 17

# Server-Specific Interfaces

---

1 The process that hosts the PMIx server library interacts with that library in two distinct manners.  
2 First, PMIx provides a set of APIs by which the host can request specific services from its library.  
3 This includes:

- 4 • collecting inventory to support scheduling algorithms,
- 5 • providing subsystems with an opportunity to precondition their resources for optimized  
6 application support,
- 7 • generating regular expressions,
- 8 • registering information to be passed to client processes, and
- 9 • requesting information on behalf of a remote process.

10 Note that the host always has access to all PMIx client APIs - the functions listed below are in  
11 addition to those available to a PMIx client.

12 Second, the host can provide a set of callback functions by which the PMIx server library can pass  
13 requests upward for servicing by the host. These include notifications of client connection and  
14 finalize, as well as requests by clients for information and/or services that the PMIx server library  
15 does not itself provide.

## 17.1 Server Initialization and Finalization

17 Initialization and finalization routines for PMIx servers.

### 17.1.1 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);
```

**INOUT module**

**pmix\_server\_module\_t** structure (handle)  
**IN info**  
 Array of **pmix\_info\_t** structures (array of handles)  
**IN ninfo**  
 Number of elements in the *info* array (**size\_t**)

Returns **PMIX\_SUCCESS** or a negative value indicating the error.

----- Required Attributes -----

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

**PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (**char\***)

Name of the namespace to use for this PMIx server.

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

Rank of this PMIx server.

**PMIX\_SERVER\_TMPDIR** "pmix.srvr.tmpdir" (**char\***)

Top-level temporary directory for all client processes connected to this server, and where the PMIx server will place its tool rendezvous point and contact information.

**PMIX\_SYSTEM\_TMPDIR** "pmix.sys.tmpdir" (**char\***)

Temporary directory for this system, and where a PMIx server that declares itself to be a system-level server will place a tool rendezvous point and contact information.

**PMIX\_SERVER\_TOOL\_SUPPORT** "pmix.srvr.tool" (**bool**)

The host RM wants to declare itself as willing to accept tool connection requests.

**PMIX\_SERVER\_SYSTEM\_SUPPORT** "pmix.srvr.sys" (**bool**)

The host RM wants to declare itself as being the local system server for PMIx connection requests.

**PMIX\_SERVER\_SESSION\_SUPPORT** "pmix.srvr.sess" (**bool**)

The host RM wants to declare itself as being the local session server for PMIx connection requests.

**PMIX\_SERVER\_GATEWAY** "pmix.srv.gway" (**bool**)

Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes (e.g., logging to email).

**PMIX\_SERVER\_SCHEDULER** "pmix.srv.sched" (**bool**)

Server is supporting system scheduler and desires access to appropriate WLM-supporting features. Indicates that the library is to be initialized for scheduler support.

## Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

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

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

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

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

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

Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

**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. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

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

Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

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

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

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

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

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

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

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

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

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

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

**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\_EXTERNAL\_PROGRESS** "pmix.evext" (bool)

5 The host shall progress the PMIx library via calls to **PMIx\_Progress**

6 **PMIX\_EVENT\_BASE** "pmix.evbase" (void\*)

7 Pointer to an **event\_base** to use in place of the internal progress thread. All PMIx library  
8 events are to be assigned to the provided event base. The event base *must* be compatible with  
9 the event library used by the PMIx implementation - e.g., either both the host and PMIx  
10 library must use libevent, or both must use libev. Cross-matches are unlikely to work and  
11 should be avoided - it is the responsibility of the host to ensure that the PMIx  
12 implementation supports (and was built with) the appropriate event library.

13 **PMIX\_TOPOLOGY2** "pmix.topo2" (pmix\_topology\_t)

14 Provide a pointer to an implementation-specific description of the local node topology.

15 **PMIX\_SERVER\_SHARE\_TOPOLOGY** "pmix.srvr.share" (bool)

16 The PMIx server is to share its copy of the local node topology (whether given to it or  
17 self-discovered) with any clients. The PMIx server will perform the necessary actions to  
18 scalably expose the description to the local clients. This includes creating any required  
19 shared memory backing stores and/ or XML representations, plus ensuring that all necessary  
20 key-value pairs for clients to access the description are included in the job-level information  
21 provided to each client. All required files are to be installed under the effective  
22 **PMIX\_SERVER\_TMPDIR** directory. The PMIx server library is responsible for cleaning up  
23 any artifacts (e.g., shared memory backing files or cached key-value pairs) at library finalize.

24 **PMIX\_SERVER\_ENABLE\_MONITORING** "pmix.srv.monitor" (bool)

25 Enable PMIx internal monitoring by the PMIx server.

26 **PMIX\_HOMOGENEOUS\_SYSTEM** "pmix.homo" (bool)

27 The nodes comprising the session are homogeneous - i.e., they each contain the same  
28 number of identical packages, fabric interfaces, GPUs, and other devices.



## 29 Description

30 Initialize the PMIx server support library, and provide a pointer to a **pmix\_server\_module\_t**  
31 structure containing the caller's callback functions. The array of **pmix\_info\_t** structs is used to  
32 pass additional info that may be required by the server when initializing. For example, it may  
33 include the **PMIX\_SERVER\_TOOL\_SUPPORT** attribute, thereby indicating that the daemon is  
34 willing to accept connection requests from tools.



1 Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module*  
 2 structure. Doing so indicates that the host environment will not provide support for multi-node  
 3 operations such as **PMIx\_Fence**, but does intend to support local clients access to information.

## 4 17.1.2 PMIx\_server\_finalize

### 5 Summary

6 Finalize the PMIx server library.

### 7 Format

*PMIx v1.0*

```
8 pmix_status_t
9 PMIx_server_finalize(void);
```

10 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### 11 Description

12 Finalize the PMIx server support library, terminating all connections to attached tools and any local  
 13 clients. All memory usage is released.

## 14 17.1.3 Server Initialization Attributes

15 These attributes are used to direct the configuration and operation of the PMIx server library by  
 16 passing them into **PMIx\_server\_init**.

17 **PMIX\_TOPOLOGY2** "pmix.topo2" (**pmix\_topology\_t**)

18 Provide a pointer to an implementation-specific description of the local node topology.

19 **PMIX\_SERVER\_SHARE\_TOPOLOGY** "pmix.srvr.share" (**bool**)

20 The PMIx server is to share its copy of the local node topology (whether given to it or  
 21 self-discovered) with any clients.

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

23 Disable legacy UNIX socket (usock) support.

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

25 POSIX *mode\_t* (9 bits valid).

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

27 Use only one rendezvous socket, letting priorities and/or environment parameters select the  
 28 active transport.

29 **PMIX\_SERVER\_TOOL\_SUPPORT** "pmix.srvr.tool" (**bool**)

30 The host RM wants to declare itself as willing to accept tool connection requests.

31 **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.

3 **PMIX\_SERVER\_SYSTEM\_SUPPORT** "pmix.srvr.sys" (bool)

4 The host RM wants to declare itself as being the local system server for PMIx connection  
5 requests.

6 **PMIX\_SERVER\_SESSION\_SUPPORT** "pmix.srvr.sess" (bool)

7 The host RM wants to declare itself as being the local session server for PMIx connection  
8 requests.

9 **PMIX\_SERVER\_START\_TIME** "pmix.srvr.strtime" (char\*)

10 Time when the server started - i.e., when the server created it's rendezvous file (given in  
11 ctime string format).

12 **PMIX\_SERVER\_TMPDIR** "pmix.srvr.tmpdir" (char\*)

13 Top-level temporary directory for all client processes connected to this server, and where the  
14 PMIx server will place its tool rendezvous point and contact information.

15 **PMIX\_SYSTEM\_TMPDIR** "pmix.sys.tmpdir" (char\*)

16 Temporary directory for this system, and where a PMIx server that declares itself to be a  
17 system-level server will place a tool rendezvous point and contact information.

18 **PMIX\_SERVER\_ENABLE\_MONITORING** "pmix.srv.monitor" (bool)

19 Enable PMIx internal monitoring by the PMIx server.

20 **PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (char\*)

21 Name of the namespace to use for this PMIx server.

22 **PMIX\_SERVER\_RANK** "pmix.srv.rank" (pmix\_rank\_t)

23 Rank of this PMIx server.

24 **PMIX\_SERVER\_GATEWAY** "pmix.srv.gway" (bool)

25 Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes  
26 (e.g., logging to email).

27 **PMIX\_SERVER\_SCHEDULER** "pmix.srv.sched" (bool)

28 Server is supporting system scheduler and desires access to appropriate WLM-supporting  
29 features. Indicates that the library is to be initialized for scheduler support.

30 **PMIX\_EXTERNAL\_PROGRESS** "pmix.evext" (bool)

31 The host shall progress the PMIx library via calls to [PMIx\\_Progress](#)

32 **PMIX\_HOMOGENEOUS\_SYSTEM** "pmix.homo" (bool)

33 The nodes comprising the session are homogeneous - i.e., they each contain the same  
34 number of identical packages, fabric interfaces, GPUs, and other devices.

## 35 17.2 Server Support Functions

36 The following APIs allow the RM daemon that hosts the PMIx server library to request specific  
37 services from the PMIx library.

### 38 17.2.1 `PMIx_generate_regex`

#### 39 Summary

40 Generate a compressed representation of the input string.

1 *PMIx v1.0*

## Format

C

```
2 pmix_status_t  
3 PMIx_generate_regex(const char *input, char **output);
```

C

4 **IN** *input*

String to process (string)

6 **OUT** *output*

Compressed representation of *input* (array of bytes)

8 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

## Description

Given a comma-separated list of *input* values, generate a reduced size representation of the input that can be passed down to the PMIx server library's **PMIx\_server\_register\_nspace** API for parsing. The order of the individual values in the *input* string is preserved across the operation. The caller is responsible for releasing the returned data.

The precise compressed representations will be implementation specific. The regular expression itself is not required to be a printable string nor to obey typical string constraints (e.g., include a **NULL** terminator byte). However, all PMIx implementations are required to include a colon-delimited **NULL**-terminated string at the beginning of the output representation that can be printed for diagnostic purposes and identifies the method used to generate the representation. The following identifiers are reserved by the PMIx Standard:

- "**raw:\0**" - indicates that the expression following the identifier is simply the comma-delimited input string (no processing was performed).
- "**pmix:\0**" - a PMIx-unique regular expression represented as a **NULL**-terminated string following the identifier.
- "**blob:\0**" - a PMIx-unique regular expression that is not represented as a **NULL**-terminated string following the identifier. Additional implementation-specific metadata may follow the identifier along with the data itself. For example, a compressed binary array format based on the *zlib* compression package, with the size encoded in the space immediately following the identifier.

Communicating the resulting output should be done by first packing the returned expression using the **PMIx\_Data\_pack**, declaring the input to be of type **PMIX\_REGEX**, and then obtaining the resulting blob to be communicated using the **PMIX\_DATA\_BUFFER\_UNLOAD** macro. The reciprocal method can be used on the remote end prior to passing the regex into **PMIx\_server\_register\_nspace**. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

## 1 17.2.2 `PMIx_generate_ppn`

### 2 Summary

3 Generate a compressed representation of the input identifying the processes on each node.

### 4 *PMIx v1.0* Format

5 `pmix_status_t`

6 `PMIx_generate_ppn(const char *input, char **ppn);`

7 **IN** `input`

8 String to process (string)

9 **OUT** `ppn`

10 Compressed representation of *input* (array of bytes)

11 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 12 Description

13 The input shall consist of a semicolon-separated list of ranges representing the ranks of processes  
14 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  
15 correspond to the node name provided at that position in the input to `PMIx_generate_regex`.  
16 Thus, in the example, ranks 1-4 would be located on the first node of the comma-separated list of  
17 names provided to `PMIx_generate_regex`, and ranks 2-5 would be on the second name in the  
18 list.

19 Rules governing the format of the returned regular expression are the same as those specified for  
20 `PMIx_generate_regex`, as detailed [here](#).

## 21 17.2.3 `PMIx_server_register_namespace`

### 22 Summary

23 Setup the data about a particular namespace.

### 24 *PMIx v1.0* Format

25 `pmix_status_t`

26 `PMIx_server_register_namespace(const pmix_namespace_t nspace,`  
27 `int nlocalprocs,`  
28 `pmix_info_t info[], size_t ninfo,`  
29 `pmix_op_cbfunc_t cbfunc,`  
30 `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 A successful return indicates that the request is being processed and the result will be returned in  
 17 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
 18 from the API. The callback function, *cbfunc*, is only called when `PMIX_SUCCESS` is returned.

19 Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- 20 • `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and  
 21 returned *success* - the *cbfunc* will not be called

22 If none of the above return codes are appropriate, then an implementation must return either a  
 23 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Required Attributes

24 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_namespace` 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\_NAMESPACE** 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\_namespace** - 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\_NAMESPACE** 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\_namespace** - 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)

Maximum number of process that can be simultaneously executing in a session. Note that this attribute is equivalent to the **PMIX\_MAX\_PROCS** attribute for the *session* realm - it is included in the PMI 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. Defaults to the *job* realm. 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. Defaults to the *job* realm.
- 9     • **PMIX\_RM\_NAME** "pmix.rm.name" (char\*)  
10         String name of the RM. As this information is not related to the namespace, it is best  
11         passed using the **PMIx\_server\_register\_resources** API.
- 12    • **PMIX\_RM\_VERSION** "pmix.rm.version" (char\*)  
13         RM version string. As this information is not related to the namespace, it is best passed  
14         using the **PMIx\_server\_register\_resources** API.
- 15    • **PMIX\_SERVER\_HOSTNAME** "pmix.srvr.host" (char\*)  
16         Host where target PMIx server is located. As this information is not related to the  
17         namespace, it is best passed using the **PMIx\_server\_register\_resources** API.

18 Job-realm information may be passed as individual **pmix\_info\_t** entries, or as part of a  
19 **pmix\_data\_array\_t** using the **PMIX\_JOB\_INFO\_ARRAY** attribute. The list of data  
20 referenced in this way shall include:

- 21    • **PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (char\*)  
22         Name of the namespace to use for this PMIx server. Identifies the namespace of the PMIx  
23         server itself
- 24    • **PMIX\_SERVER\_RANK** "pmix.srv.rank" (pmix\_rank\_t)  
25         Rank of this PMIx server. Identifies the rank of the PMIx server itself.
- 26    • **PMIX\_NAMESPACE** "pmix.namespace" (char\*)  
27         Namespace of the job - may be a numerical value expressed as a string, but is often an  
28         alphanumeric string carrying information solely of use to the system. Required to be  
29         unique within the scope of the host environment. Identifies the namespace of the job  
30         being registered.
- 31    • **PMIX\_JOBID** "pmix.jobid" (char\*)  
32         Job identifier assigned by the scheduler to the specified job - may be identical to the  
33         namespace, but is often a numerical value expressed as a string (e.g., "12345.3").
- 34    • **PMIX\_JOB\_SIZE** "pmix.job.size" (uint32\_t)  
35         Total number of processes in the specified job across all contained applications. Note that  
36         this value can be different from **PMIX\_MAX\_PROCS**. For example, users may choose to  
37         subdivide an allocation (running several jobs in parallel within it), and dynamic

1 programming models may support adding and removing processes from a running *job*  
2 on-the-fly. In the latter case, PMIx events may be used to notify processes within the job  
3 that the job size has changed.

- 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. Defaults to the *job* realm. Retrieval of this attribute defaults to the job level  
8 unless an appropriate specification is given (e.g., **PMIX\_SESSION\_INFO**).
- 9 • **PMIX\_NODE\_MAP** "`pmix.nmap`" (`char*`)  
10 Regular expression of nodes currently hosting processes in the specified realm - see  
11 [17.2.3.2](#) for an explanation of its generation. Defaults to the *job* realm.
- 12 • **PMIX\_PROC\_MAP** "`pmix.pmap`" (`char*`)  
13 Regular expression describing processes on each node in the specified realm - see [17.2.3.2](#)  
14 for an explanation of its generation. Defaults to the *job* realm.

15 plus the following optional information:

- 16 • **PMIX\_NPROC\_OFFSET** "`pmix.offset`" (`pmix_rank_t`)  
17 Starting global rank of the specified job.
- 18 • **PMIX\_JOB\_NUM\_APPS** "`pmix.job.napps`" (`uint32_t`)  
19 Number of applications in the specified job. This is a required attribute if more than one  
20 application is included in the job.
- 21 • **PMIX\_MAPBY** "`pmix.mapby`" (`char*`)  
22 Process mapping policy - when accessed using **PMIx\_Get**, use the  
23 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
24 provided namespace. Supported values are launcher specific.
- 25 • **PMIX\_RANKBY** "`pmix.rankby`" (`char*`)  
26 Process ranking policy - when accessed using **PMIx\_Get**, use the  
27 **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for  
28 the provided namespace. Supported values are launcher specific.
- 29 • **PMIX\_BINDTO** "`pmix.bindto`" (`char*`)  
30 Process binding policy - when accessed using **PMIx\_Get**, use the  
31 **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
32 provided namespace. Supported values are launcher specific.
- 33 • **PMIX\_HOSTNAME\_KEEP\_FQDN** "`pmix.fqdn`" (`bool`)  
34 FQDNs are being retained by the PMIx library.
- 35 • **PMIX\_ANL\_MAP** "`pmix.anlmap`" (`char*`)  
36 Process map equivalent to **PMIX\_PROC\_MAP** expressed in Argonne National  
37 Laboratory's PMI-1/PMI-2 notation. Defaults to the *job* realm.



- 1 • **PMIX\_TDIR\_RMCLEAN** "pmix.tdir.rmclean" (bool)
- 2     Resource Manager will cleanup assigned temporary directory trees.
- 3 • **PMIX\_CRYPTO\_KEY** "pmix.sec.key" (pmix\_byte\_object\_t)
- 4     Blob containing crypto key.

5 If more than one application is included in the namespace, then the host environment is also  
 6 required to supply data consisting of the following items for each application in the job, passed as a  
 7 **pmix\_data\_array\_t** using the **PMIX\_APP\_INFO\_ARRAY** attribute:

- 8 • **PMIX\_APPNUM** "pmix.appnum" (uint32\_t)
- 9     The application number within the job in which the specified process is a member. This  
 10     attribute must appear at the beginning of the array.
- 11 • **PMIX\_APP\_SIZE** "pmix.app.size" (uint32\_t)
- 12     Number of processes in the specified application, regardless of their execution state - i.e.,  
 13     this number may include processes that either failed to start or have already terminated.
- 14 • **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)
- 15     Maximum number of processes that can be executed in the specified realm. Typically, this  
 16     is a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
 17     description. Defaults to the *job* realm. Requires use of the **PMIX\_APP\_INFO** attribute  
 18     to avoid ambiguity when retrieving it.
- 19 • **PMIX\_APPLDR** "pmix.aldr" (pmix\_rank\_t)
- 20     Lowest rank in the specified application.
- 21 • **PMIX\_WDIR** "pmix.wdir" (char\*)
- 22     Working directory for spawned processes. This attribute is required for all registrations,  
 23     but may be provided as an individual **pmix\_info\_t** entry if only one application is  
 24     included in the namespace.
- 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"). This attribute is required for all registrations, but may be provided as an  
 28     individual **pmix\_info\_t** entry if only one application is included in the namespace.

29 plus the following optional information:

- 30 • **PMIX\_PSET\_NAMES** "pmix.pset.nms" (pmix\_data\_array\_t\*)
- 31     Returns an array of **char\*** string names of the process sets in which the given process is  
 32     a member.
- 33 • **PMIX\_APP\_MAP\_TYPE** "pmix.apmap.type" (char\*)
- 34     Type of mapping used to layout the application (e.g., **cyclic**). This attribute may be  
 35     provided as an individual **pmix\_info\_t** entry if only one application is included in the  
 36     namespace.
- 37 • **PMIX\_APP\_MAP\_REGEX** "pmix.apmap.regex" (char\*)

1 Regular expression describing the result of the process mapping. This attribute may be  
2 provided as an individual `pmix_info_t` entry if only one application is included in the  
3 namespace.

4 The data may also include attributes provided by the host environment that identify the  
5 programming model (as specified by the user) being executed within the application. The PMIx  
6 server library may utilize this information to customize the environment to fit that model (e.g.,  
7 adding environmental variables specified by the corresponding standard for that model):

- 8 • `PMIX_PROGRAMMING_MODEL` "`pmix.pgm.model`" (`char*`)  
9 Programming model being initialized (e.g., "MPI" or "OpenMP").
- 10 • `PMIX_MODEL_LIBRARY_NAME` "`pmix.mdl.name`" (`char*`)  
11 Programming model implementation ID (e.g., "OpenMPI" or "MPICH").
- 12 • `PMIX_MODEL_LIBRARY_VERSION` "`pmix.mld.vrs`" (`char*`)  
13 Programming model version string (e.g., "2.1.1").

14 Node-realm information may be passed as individual `pmix_info_t` entries if only one node will  
15 host processes from the job being registered, or as part of a `pmix_data_array_t` using the  
16 `PMIX_NODE_INFO_ARRAY` attribute when multiple nodes are involved in the job. The list of data  
17 referenced in this way shall include:

- 18 • `PMIX_NODEID` "`pmix.nodeid`" (`uint32_t`)  
19 Node identifier expressed as the node's index (beginning at zero) in an array of nodes  
20 within the active session. The value must be unique and directly correlate to the  
21 `PMIX_HOSTNAME` of the node - i.e., users can interchangeably reference the same  
22 location using either the `PMIX_HOSTNAME` or corresponding `PMIX_NODEID`.
- 23 • `PMIX_HOSTNAME` "`pmix.hname`" (`char*`)  
24 Name of the host, as returned by the `gethostname` utility or its equivalent. As this  
25 information is not related to the namespace, it can be passed using the  
26 `PMIx_server_register_resources` API. However, either it or the  
27 `PMIX_NODEID` must be included in the array to properly identify the node.
- 28 • `PMIX_HOSTNAME_ALIASES` "`pmix.alias`" (`char*`)  
29 Comma-delimited list of names by which the target node is known. As this information is  
30 not related to the namespace, it is best passed using the  
31 `PMIx_server_register_resources` API.
- 32 • `PMIX_LOCAL_SIZE` "`pmix.local.size`" (`uint32_t`)  
33 Number of processes in the specified job or application realm on the caller's node.  
34 Defaults to job realm unless the `PMIX_APP_INFO` and the `PMIX_APPNUM` qualifiers are  
35 given.
- 36 • `PMIX_NODE_SIZE` "`pmix.node.size`" (`uint32_t`)  
37 Number of processes across all jobs that are executing upon the node.
- 38 • `PMIX_LOCALLDR` "`pmix.lldr`" (`pmix_rank_t`)

1           Lowest rank within the specified job on the node (defaults to current node in absence of  
2           **PMIX\_HOSTNAME** or **PMIX\_NODEID** qualifier).

- 3     • **PMIX\_LOCAL\_PEERS** "pmix.lpeers" (char\*)  
4         Comma-delimited list of ranks that are executing on the local node within the specified  
5         namespace – shortcut for **PMIx\_Resolve\_peers** for the local node.

6 plus the following information for the server's own node:

- 7     • **PMIX\_TMPDIR** "pmix.tmpdir" (char\*)  
8         Full path to the top-level temporary directory assigned to the session.
- 9     • **PMIX\_NSDIR** "pmix.nsdire" (char\*)  
10         Full path to the temporary directory assigned to the specified job, under **PMIX\_TMPDIR**.
- 11    • **PMIX\_LOCAL\_PROCS** "pmix.lprocs" (pmix\_proc\_t array)  
12         Array of **pmix\_proc\_t** of all processes executing on the local node – shortcut for  
13         **PMIx\_Resolve\_peers** for the local node and a **NULL** namespace argument. The  
14         process identifier is ignored for this attribute.

15 The data may also include the following optional information for the server's own node:

- 16    • **PMIX\_LOCAL\_CPUSETS** "pmix.lcpus" (pmix\_data\_array\_t)  
17         A **pmix\_data\_array\_t** array of string representations of the PU binding bitmaps  
18         applied to each local *peer* on the caller's node upon launch. Each string shall begin with  
19         the name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap  
20         string itself. The array shall be in the same order as the processes returned by  
21         **PMIX\_LOCAL\_PEERS** for that namespace.
- 22    • **PMIX\_AVAIL\_PHYS\_MEMORY** "pmix.pmem" (uint64\_t)  
23         Total available physical memory on a node. As this information is not related to the  
24         namespace, it can be passed using the **PMIx\_server\_register\_resources** API.

25 and the following optional information for other nodes:

- 26    • **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)  
27         Maximum number of processes that can be executed in the specified realm. Typically, this  
28         is a constraint imposed by a scheduler or by user settings in a hostfile or other resource  
29         description. Defaults to the *job* realm. Requires use of the **PMIX\_NODE\_INFO** attribute  
30         to avoid ambiguity when retrieving it.

31 Process-realm information shall include the following data for each process in the job, passed as a  
32 **pmix\_data\_array\_t** using the **PMIX\_PROC\_INFO\_ARRAY** attribute:

- 33    • **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)  
34         Process rank within the job, starting from zero.
- 35    • **PMIX\_APPNUM** "pmix.appnum" (uint32\_t)  
36         The application number within the job in which the specified process is a member. This  
37         attribute may be omitted if only one application is present in the namespace.

- 1 • **PMIX\_APP\_RANK** "pmix.apprank" (pmix\_rank\_t)  
2 Rank of the specified process within its application. This attribute may be omitted if only  
3 one application is present in the namespace.
- 4 • **PMIX\_GLOBAL\_RANK** "pmix.grank" (pmix\_rank\_t)  
5 Rank of the specified process spanning across all jobs in this session, starting with zero.  
6 Note that no ordering of the jobs is implied when computing this value. As jobs can start  
7 and end at random times, this is defined as a continually growing number - i.e., it is not  
8 dynamically adjusted as individual jobs and processes are started or terminated.
- 9 • **PMIX\_LOCAL\_RANK** "pmix.lrank" (uint16\_t)  
10 Rank of the specified process on its node - refers to the numerical location (starting from  
11 zero) of the process on its node when counting only those processes from the same job  
12 that share the node, ordered by their overall rank within that job.
- 13 • **PMIX\_NODE\_RANK** "pmix.nrank" (uint16\_t)  
14 Rank of the specified process on its node spanning all jobs- refers to the numerical location  
15 (starting from zero) of the process on its node when counting all processes (regardless of  
16 job) that share the node, ordered by their overall rank within the job. The value represents  
17 a snapshot in time when the specified process was started on its node and is not  
18 dynamically adjusted as processes from other jobs are started or terminated on the node.
- 19 • **PMIX\_NODEID** "pmix.nodeid" (uint32\_t)  
20 Node identifier expressed as the node's index (beginning at zero) in an array of nodes  
21 within the active session. The value must be unique and directly correlate to the  
22 **PMIX\_HOSTNAME** of the node - i.e., users can interchangeably reference the same  
23 location using either the **PMIX\_HOSTNAME** or corresponding **PMIX\_NODEID**.
- 24 • **PMIX\_REINCARNATION** "pmix.reinc" (uint32\_t)  
25 Number of times this process has been re-instantiated - i.e, a value of zero indicates that  
26 the process has never been restarted. 5
- 27 • **PMIX\_SPAWNED** "pmix.spawned" (bool)  
28 **true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return  
29 status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.

30 plus the following information for processes that are local to the server:

- 31 • **PMIX\_LOCALITY\_STRING** "pmix.locstr" (char\*)  
32 String describing a process's bound location - referenced using the process's rank. The  
33 string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon.  
34 The remainder of the string represents the corresponding locality as expressed by the  
35 underlying implementation. The entire string must be passed to  
36 **PMIx\_Get\_relative\_locality** for processing. Note that hosts are only required to  
37 provide locality strings for local client processes - thus, a call to **PMIx\_Get** for the  
38 locality string of a process that returns **PMIX\_ERR\_NOT\_FOUND** indicates that the  
39 process is not executing on the same node.

- 1 • **PMIX\_PROCDIR** "pmix.pdir" (char\*)  
2 Full path to the subdirectory under **PMIX\_NSDIR** assigned to the specified process.
- 3 • **PMIX\_PACKAGE\_RANK** "pmix.pkgrank" (uint16\_t)  
4 Rank of the specified process on the *package* where this process resides - refers to the  
5 numerical location (starting from zero) of the process on its package when counting only  
6 those processes from the same job that share the package, ordered by their overall rank  
7 within that job. Note that processes that are not bound to PUs within a single specific  
8 package cannot have a package rank.

9 and the following optional information - note that some of this information can be derived from  
10 information already provided by other attributes, but it may be included here for ease of retrieval by  
11 users:

- 12 • **PMIX\_HOSTNAME** "pmix.hname" (char\*)  
13 Name of the host, as returned by the **gethostname** utility or its equivalent.
- 14 • **PMIX\_CPuset** "pmix.cpuset" (char\*)  
15 A string representation of the PU binding bitmap applied to the process upon launch. The  
16 string shall begin with the name of the library that generated it (e.g., "hwloc") followed by  
17 a colon and the bitmap string itself.
- 18 • **PMIX\_CPuset\_BITMAP** "pmix.bitmap" (pmix\_cpuset\_t\*)  
19 Bitmap applied to the process upon launch.
- 20 • **PMIX\_DEVICE\_DISTANCES** "pmix.dev.dist" (pmix\_data\_array\_t)  
21 Return an array of **pmix\_device\_distance\_t** containing the minimum and  
22 maximum distances of the given process location to all devices of the specified type on the  
23 local node.

---

24  
25 Attributes not directly provided by the host environment may be derived by the PMIx server library  
26 from other required information and included in the data made available to the server library's  
27 clients.

---

28 **Description**

29 Pass job-related information to the PMIx server library for distribution to local client processes.

### Advice to PMIx server hosts

1 Host environments are required to execute this operation prior to starting any local application  
2 process within the given namespace.

3 The PMIx server must register all namespaces that will participate in collective operations with  
4 local processes. This means that the server must register a namespace even if it will not host any  
5 local processes from within that namespace if any local process of another namespace might at  
6 some point perform an operation involving one or more processes from the new namespace. This is  
7 necessary so that the collective operation can identify the participants and know when it is locally  
8 complete.

9 The caller must also provide the number of local processes that will be launched within this  
10 namespace. This is required for the PMIx server library to correctly handle collectives as a  
11 collective operation call can occur before all the local processes have been started.

12 A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

### Advice to users

13 The number of local processes for any given namespace is generally fixed at the time of application  
14 launch. Calls to **PMIx\_Spawn** result in processes launched in their own namespace, not that of  
15 their parent. However, it is possible for processes to *migrate* to another node via a call to  
16 **PMIx\_Job\_control\_nb**, thus resulting in a change to the number of local processes on both  
17 the initial node and the node to which the process moved. It is therefore critical that applications  
18 not migrate processes without first ensuring that PMIx-based collective operations are not in  
19 progress, and that no such operations be initiated until process migration has completed.

## 1 17.2.3.1 Namespace registration attributes

2 The following attributes are defined specifically for use with the  
3 `PMIx_server_register_namespace` API: `PMIX_REGISTER_NODATA`  
4 `"pmix.reg.nodata"` (bool)  
5 Registration is for this namespace only, do not copy job data.

6 The following attributes are used to assemble information according to its data realm (*session*, *job*,  
7 *application*, *node*, or *process* as defined in Section 6.1) for registration where ambiguity may exist -  
8 see 17.2.3.2 for examples of their use.

9 `PMIX_SESSION_INFO_ARRAY` "pmix.ssn.arr" (`pmix_data_array_t`)

10 Provide an array of `pmix_info_t` containing session-realm information. The  
11 `PMIX_SESSION_ID` attribute is required to be included in the array.

12 `PMIX_JOB_INFO_ARRAY` "pmix.job.arr" (`pmix_data_array_t`)

13 Provide an array of `pmix_info_t` containing job-realm information. The  
14 `PMIX_SESSION_ID` attribute of the *session* containing the *job* is required to be included in  
15 the array whenever the PMIx server library may host multiple sessions (e.g., when executing  
16 with a host RM daemon). As information is registered one job (aka namespace) at a time via  
17 the `PMIx_server_register_namespace` API, there is no requirement that the array  
18 contain either the `PMIX_NAMESPACE` or `PMIX_JOBID` attributes when used in that context  
19 (though either or both of them may be included). At least one of the job identifiers must be  
20 provided in all other contexts where the job being referenced is ambiguous.

21 `PMIX_APP_INFO_ARRAY` "pmix.app.arr" (`pmix_data_array_t`)

22 Provide an array of `pmix_info_t` containing application-realm information. The  
23 `PMIX_NAMESPACE` or `PMIX_JOBID` attributes of the *job* containing the application, plus its  
24 `PMIX_APPNUM` attribute, must to be included in the array when the array is *not* included as  
25 part of a call to `PMIx_server_register_namespace` - i.e., when the job containing the  
26 application is ambiguous. The job identification is otherwise optional.

27 `PMIX_PROC_INFO_ARRAY` "pmix.pdata" (`pmix_data_array_t`)

28 Provide an array of `pmix_info_t` containing process-realm information. The  
29 `PMIX_RANK` and `PMIX_NAMESPACE` attributes, or the `PMIX_PROCID` attribute, are required  
30 to be included in the array when the array is not included as part of a call to  
31 `PMIx_server_register_namespace` - i.e., when the job containing the process is  
32 ambiguous. All three may be included if desired. When the array is included in some  
33 broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID`  
34 attribute must be included (the others are optional).

35 `PMIX_NODE_INFO_ARRAY` "pmix.node.arr" (`pmix_data_array_t`)

36 Provide an array of `pmix_info_t` containing node-realm information. At a minimum,  
37 either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the  
38 array, though both may be included.

39 Note that these assemblages can be used hierarchically:

- 40 • a `PMIX_JOB_INFO_ARRAY` might contain multiple `PMIX_APP_INFO_ARRAY` elements,  
41 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

PMIx implementations must be capable of properly parsing and storing any hierarchical depth of information arrays. The resulting stored values are must to be accessible via both `PMIx_Get` and `PMIx_Query_info_nb` APIs, assuming appropriate directives are provided by the caller.

## 17.2.3.2 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it describes how information provided in the `info` parameter of the `PMIx_server_register_nspace` shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

### Advice to PMIx server hosts

Creating the `info` array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single `pmix_info_t` structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final `info` array can be allocated based on the number of elements on the list, and then the values in the list object `pmix_info_t` structures transferred to the corresponding array element utilizing the `PMIX_INFO_XFER` macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a `session` or `job`. PMIx provides several tools to facilitate this operation, beginning by constructing an argv-like array of node names. This array is then passed to the `PMIx_generate_regex` function to create a regular expression parseable by the PMIx server library, as shown below:



```

1  char **nodes = NULL;
2  char *nodelist;
3  char *regex;
4  size_t n;
5  pmix_status_t rc;
6  pmix_info_t info;
7
8  /* loop over an array of nodes, adding each
9   * name to the array */
10 for (n=0; n < num_nodes; n++) {
11     /* filter the nodes to ignore those not included
12      * in the target range (session, job, etc.). In
13      * this example, all nodes are accepted */
14     PMIX_ARGV_APPEND(&nodes, node[n]->name);
15 }
16
17 /* join into a comma-delimited string */
18 nodelist = PMIX_ARGV_JOIN(nodes, ',');
19
20 /* release the array */
21 PMIX_ARGV_FREE(nodes);
22
23 /* generate regex */
24 rc = PMIx_generate_regex(nodelist, &regex);
25
26 /* release list */
27 free(nodelist);
28
29 /* pass the regex as the value to the PMIX_NODE_MAP key */
30 PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_REGEX);
31 /* release the regex */
32 free(regex);

```

33 Changing the filter criteria allows the construction of node maps for any level of information. A  
34 description of the returned regular expression is provided [here](#).

35 A similar method is used to construct the map of processes on each node from the namespace being  
36 registered. This may be done for each information level of interest (e.g., to identify the process map  
37 for the entire *job* or for each *application* in the job) by changing the search criteria. An example is  
38 shown below for the case of creating the process map for a *job*:

```

1  char **ndppn;
2  char rank[30];
3  char **ppnarray = NULL;
4  char *ppn;
5  char *localranks;
6  char *regex;
7  size_t n, m;
8  pmix_status_t rc;
9  pmix_info_t info;
10
11 /* loop over an array of nodes */
12 for (n=0; n < num_nodes; n++) {
13     /* for each node, construct an array of ranks on that node */
14     ndppn = NULL;
15     for (m=0; m < node[n]->num_procs; m++) {
16         /* ignore processes that are not part of the target job */
17         if (!PMIX_CHECK_NAMESPACE(targetjob, node[n]->proc[m].nspace)) {
18             continue;
19         }
20         snprintf(rank, 30, "%d", node[n]->proc[m].rank);
21         PMIX_ARGV_APPEND(&ndppn, rank);
22     }
23     /* convert the array into a comma-delimited string of ranks */
24     localranks = PMIX_ARGV_JOIN(ndppn, ',');
25     /* release the local array */
26     PMIX_ARGV_FREE(ndppn);
27     /* add this node's contribution to the overall array */
28     PMIX_ARGV_APPEND(&ppnarray, localranks);
29     /* release the local list */
30     free(localranks);
31 }
32
33 /* join into a semicolon-delimited string */
34 ppn = PMIX_ARGV_JOIN(ppnarray, ';');
35
36 /* release the array */
37 PMIX_ARGV_FREE(ppnarray);
38
39 /* generate ppn regex */
40 rc = PMIX_generate_ppn(ppn, &regex);
41
42 /* release list */

```

```

1  free (ppn) ;
2
3  /* pass the regex as the value to the PMIX_PROC_MAP key */
4  PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_REGEX);
5  /* release the regex */
6  free (regex);

```

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.

The *info* array at this point might look like (where the labels identify the corresponding attribute - e.g., “Session ID” corresponds to the `PMIX_SESSION_ID` attribute):

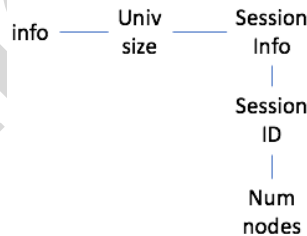


Figure 17.1.: Session-level information elements

- Job-level information includes all job-specific values such as `PMIX_JOB_SIZE`, `PMIX_JOB_NUM_APPS`, and `PMIX_JOBID`. Since each invocation of `PMIx_server_register_nspace` describes a single *job*, job-specific values 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` identified by the `PMIX_JOB_INFO_ARRAY` attribute - this is required in cases where non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the job. Note

1 that since the invocation only involves a single namespace, there is no need to include the  
2 **PMIX\_NAMESPACE** attribute in the array.

3 Upon conclusion of this step, the *info* array might look like:

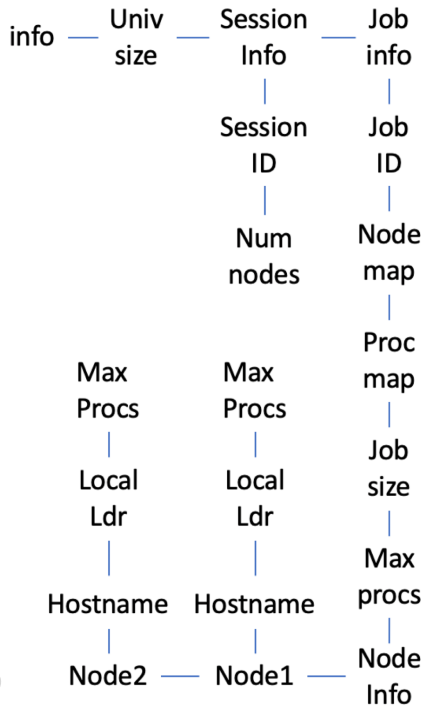


Figure 17.2.: Job-level information elements

4 Note that in this example, **PMIX\_NUM\_NODES** is not required as that information is contained in  
5 the **PMIX\_NODE\_MAP** attribute. Similarly, **PMIX\_JOB\_SIZE** is not technically required as that  
6 information is contained in the **PMIX\_PROC\_MAP** when combined with the corresponding node  
7 map - however, there is no issue with including the job size as a separate entry.

8 The example also illustrates the hierarchical use of the **PMIX\_NODE\_INFO\_ARRAY** attribute.  
9 In this case, we have chosen to pass several job-related values for each node - since those values  
10 are non-unique across the job, they must be passed in a node-info container. Note that the choice  
11 of what information to pass into the PMIx server library versus what information to derive from  
12 other values at time of request is left to the host environment. PMIx implementors in turn may, if  
13 they choose, pre-parse registration data to create expanded views (thus enabling faster response  
14 to requests at the expense of memory footprint) or to compress views into tighter representations  
15 (thus trading minimized footprint for longer response times).

- 16 • Application-level information includes all application-specific values such as **PMIX\_APP\_SIZE**

and **PMIX\_APPLDR**. If the *job* contains only a single *application*, then the application-specific values can be specified independently - i.e., in their own **pmix\_info\_t** elements of the *info* array - or as a **pmix\_data\_array\_t** array of **pmix\_info\_t** using the **PMIX\_APP\_INFO\_ARRAY** attribute and identified by including the **PMIX\_APPNUM** attribute in 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 17.3, assuming there are two applications in the job being registered:

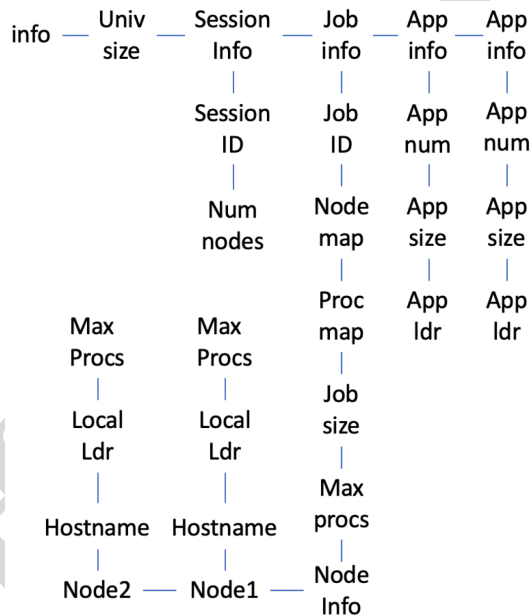


Figure 17.3.: Application-level information elements

- 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 17.4:
- 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

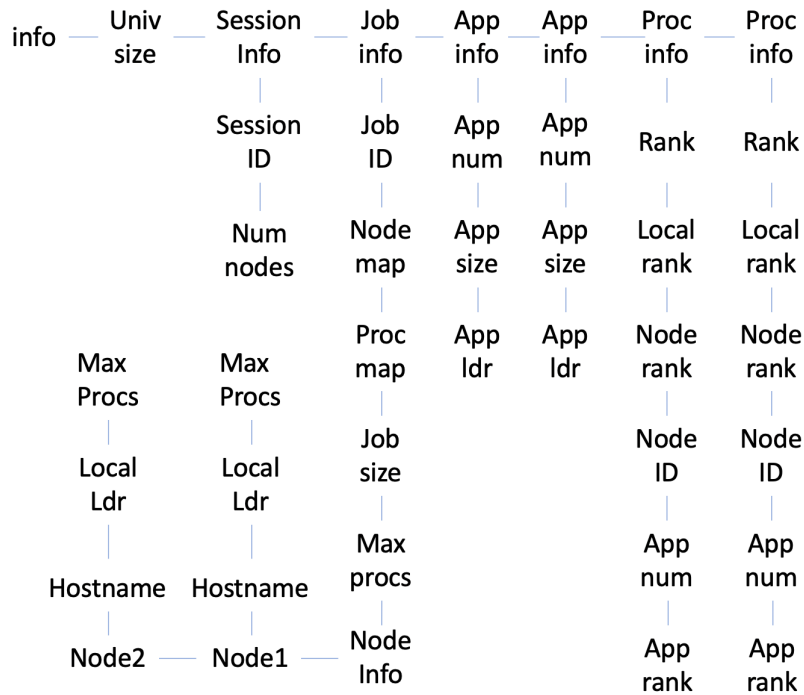


Figure 17.4.: Process-level information elements

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:

## 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_NAMESPACE(targetjob, mynode->proc[m].nspace)) {
10             continue;
11         }
12         snprintf(rank, 30, "%d", mynode->proc[m].rank);
13         PMIX_ARGV_APPEND(&ndppn, rank);
14     }
15     /* convert the array into a comma-delimited string of ranks */
16     localranks = PMIX_ARGV_JOIN(ndppn, ',');
17     /* release the local array */
18     PMIX_ARGV_FREE(ndppn);
19
20     /* pass the string as the value to the PMIX_LOCAL_PEERS key */
21     PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
22
23     /* release the list */
24     free(localranks);

```

## C

25 The `PMIX_LOCAL_CPUSSETS` value is constructed in a similar manner. In the provided  
26 example, it is assumed that an Hardware Locality (HWLOC) cpuset representation (a  
27 comma-delimited string of processor IDs) of the processors assigned to each process has  
28 previously been generated and stored on the process description. Thus, the value can be  
29 constructed as shown below:

## C

```

30     char **ndcpus = NULL;
31     char *localcpus;
32     size_t m;
33     pmix_info_t info;
34
35     for (m=0; m < mynode->num_procs; m++) {
36         /* ignore processes that are not part of the target job */
37         if (!PMIX_CHECK_NAMESPACE(targetjob, mynode->proc[m].nspace)) {
38             continue;

```

```

1      }
2      PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
3  }
4  /* convert the array into a colon-delimited string */
5  localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
6  /* release the local array */
7  PMIX_ARGV_FREE(ndcpus);
8
9  /* pass the string as the value to the PMIX_LOCAL_CPUSSETS key */
10 PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSSETS, localcpus, PMIX_STRING);
11
12 /* release the list */
13 free(localcpus);

```

C

14 Note that for efficiency, these two values can be computed at the same time.

15 The final *info* array might therefore look like the diagram in 17.5:

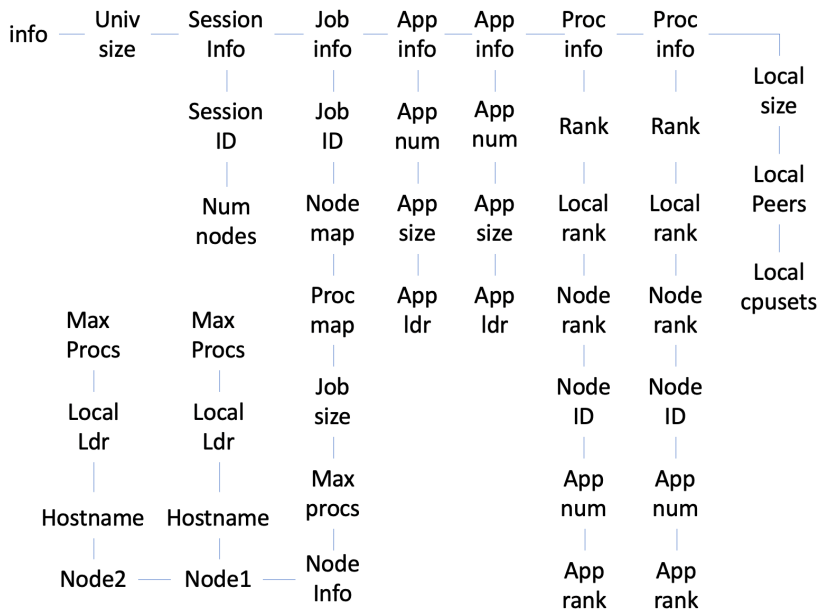


Figure 17.5.: Final information array



## 1 17.2.4 PMIx\_server\_deregister\_namespace

### 2 Summary

3 Deregister a namespace.

### 4 *PMIx v1.0* Format

```
void PMIx_server_deregister_namespace(const pmix_namespace_t namespace,  
                                     pmix_op_cbfunc_t cbfunc, void *cbdata);
```

#### 7 IN namespace

8 Namespace (string)

#### 9 IN cbfunc

10 Callback function [pmix\\_op\\_cbfunc\\_t](#). A **NULL** function reference indicates that the  
11 function is to be executed as a blocking operation. (function reference)

#### 12 IN cbdata

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

### 14 Description

15 Deregister the specified *namespace* and purge all objects relating to it, including any client information  
16 from that namespace. This is intended to support persistent PMIx servers by providing an  
17 opportunity for the host RM to tell the PMIx server library to release all memory for a completed  
18 job. Note that the library must not invoke the callback function prior to returning from the API, and  
19 that a **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

## 20 17.2.5 PMIx\_server\_register\_resources

### 21 Summary

22 Register non-namespace related information with the local PMIx server library.

### 23 *PMIx v4.0* Format

```
pmix_status_t  
PMIx_server_register_resources(pmix_info_t info[], size_t ninfo,  
                              pmix_op_cbfunc_t cbfunc,  
                              void *cbdata);
```

#### 28 IN info

29 Array of info structures (array of handles)

#### 30 IN ninfo

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

1 **IN** **cbfunc**

2 Callback function [pmix\\_op\\_cbfunc\\_t](#). A **NULL** function reference indicates that the  
3 function is to be executed as a blocking operation (function reference)

4 **IN** **cbdata**

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

6 **Description**

7 Pass information about resources not associated with a given namespace to the PMIx server library  
8 for distribution to local client processes. This includes information on fabric devices, GPUs, and  
9 other resources. All information provided through this API shall be made available to each job as  
10 part of its job-level information. Duplicate information provided with the  
11 [PMIx\\_server\\_register\\_namespace](#) API shall override any information provided by this  
12 function for that namespace, but only for that specific namespace.

13 Returns [PMIX\\_SUCCESS](#) or a negative value indicating the error.

▼ **Advice to PMIx server hosts** ▼

14 Note that information passed in this manner could also have been included in a call to  
15 [PMIx\\_server\\_register\\_namespace](#) - e.g., as part of a [PMIX\\_NODE\\_INFO\\_ARRAY](#) array.  
16 This API is provided as a logical alternative for code clarity, especially where multiple jobs may be  
17 supported by a single PMIx server library instance, to avoid multiple registration of static resource  
18 information.

19 A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

20 **17.2.6 PMIx\_server\_deregister\_resources**

21 **Summary**

22 Remove specified non-namespace related information from the local PMIx server library.

## Format

C

```
pmix_status_t
PMIx_server_deregister_resources(pmix_info_t info[], size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata);
```

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)

## Description

Remove information about resources not associated with a given namespace from the PMIx server library. Only the *key* fields of the provided *info* array shall be used for the operation - the associated values shall be ignored except where they serve as qualifiers to the request. For example, to remove a specific fabric device from a given node, the *info* array might include a [PMIX\\_NODE\\_INFO\\_ARRAY](#) containing the [PMIX\\_NODEID](#) or [PMIX\\_HOSTNAME](#) identifying the node hosting the device, and the [PMIX\\_FABRIC\\_DEVICE\\_NAME](#) specifying the device to be removed. Alternatively, the device could be removed using only the [PMIX\\_DEVICE\\_ID](#) as this is unique across the overall system.

Returns [PMIX\\_SUCCESS](#) or a negative value indicating the error.

### Advice to PMIx server hosts

As information not related to namespaces is considered *static*, there is no requirement that the host environment deregister resources prior to finalizing the PMIx server library. The server library shall properly cleanup as part of its normal finalize operations. Deregistration of resources is only required, therefore, when the host environment determines that client processes should no longer have access to that information.

A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

## 17.2.7 PMIx\_server\_register\_client

### Summary

Register a client process with the PMIx server library.

## Format

C

```
pmix_status_t
PMIx_server_register_client(const pmix_proc_t *proc,
                           uid_t uid, gid_t gid,
                           void *server_object,
                           pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

### IN **proc**

[pmix\\_proc\\_t](#) structure (handle)

### IN **uid**

user id (integer)

### IN **gid**

group id (integer)

### IN **server\_object**

(memory reference)

### 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)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## Description

Register a client process with the PMIx server library.

The host server can also, if it desires, provide an object it wishes to be returned when a server function is called that relates to a specific process. For example, the host server may have an object that tracks the specific client. Passing the object to the library allows the library to provide that object to the host server during subsequent calls related to that client, such as a [pmix\\_server\\_client\\_connected2\\_fn\\_t](#) function. This allows the host server to access the object without performing a lookup based on the client's namespace and rank.

## Advice to PMIx server hosts

1 Host environments are required to execute this operation prior to starting the client process. The  
2 expected user ID and group ID of the child process allows the server library to properly authenticate  
3 clients as they connect by requiring the two values to match. Accordingly, the detected user and  
4 group ID's of the connecting process are not included in the  
5 `pmix_server_client_connected2_fn_t` server module function.

## Advice to PMIx library implementers

6 For security purposes, the PMIx server library should check the user and group ID's of a  
7 connecting process against those provided for the declared client process identifier via the  
8 `PMIx_server_register_client` prior to completing the connection.

## 17.2.8 `PMIx_server_deregister_client`

### Summary

Deregister a client and purge all data relating to it.

### Format

*PMIx v1.0*

C

```
void  
PMIx_server_deregister_client(const pmix_proc_t *proc,  
                             pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

### IN `proc`

`pmix_proc_t` structure (handle)

### 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)

### Description

The `PMIx_server_deregister_namespace` 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.

## 1 17.2.9 PMIx\_server\_setup\_fork

### 2 Summary

3 Setup the environment of a child process to be forked by the host.

### 4 Format

```
5 pmix_status_t  
6 PMIx_server_setup_fork(const pmix_proc_t *proc,  
7                        char ***env);
```

8 **IN** `proc`

9 `pmix_proc_t` structure (handle)

10 **IN** `env`

11 Environment array (array of strings)

12 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 13 Description

14 Setup the environment of a child process to be forked by the host so it can correctly interact with  
15 the PMIx server.

16 The PMIx client needs some setup information so it can properly connect back to the server. This  
17 function will set appropriate environmental variables for this purpose, and will also provide any  
18 environmental variables that were specified in the launch command (e.g., via `PMIx_Spawn`) plus  
19 other values (e.g., variables required to properly initialize the client's fabric library).

### Advice to PMIx server hosts

20 Host environments are required to execute this operation prior to starting the client process.

## 21 17.2.10 PMIx\_server\_dmodex\_request

### 22 Summary

23 Define a function by which the host server can request modex data from the local PMIx server.

## Format

C

```
pmix_status_t
PMIx_server_dmodex_request(const pmix_proc_t *proc,
                           pmix_dmodex_response_fn_t cbfunc,
                           void *cbdata);
```

C

**IN** `proc`

`pmix_proc_t` structure (handle)

**IN** `cbfunc`

Callback function `pmix_dmodex_response_fn_t` (function reference)

**IN** `cbdata`

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

A successful return indicates that the request is being processed and the result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API. The callback function, `cbfunc`, is only called when `PMIX_SUCCESS` is returned.

## Description

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the `PMIx_Put` and `PMIx_Commit` functions followed by a `PMIx_Fence` barrier that globally exchanges the posted information. However, the barrier operation represents a significant time impact at large scale.

PMIx supports an alternative wireup method known as *Direct Modex* that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the `PMIx_server_dmodex_request` function, receiving the response in the provided `cbfunc` once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

## Advice to users

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

## 1 17.2.10.1 Server Direct Modex Response Callback Function

2 The [PMIx\\_server\\_dmodex\\_request](#) callback function.

### 3 Summary

4 Provide a function by which the local PMIx server library can return connection and other data  
5 posted by local application processes to the host resource manager.

### 6 Format

*PMIx v1.0*

C

```
7 typedef void (*pmix_dmodex_response_fn_t) (  
8             pmix_status_t status,  
9             char *data, size_t sz,  
10            void *cbdata);
```

C

### 11 IN status

12 Returned status of the request ([pmix\\_status\\_t](#))

### 13 IN data

14 Pointer to a data "blob" containing the requested information (handle)

### 15 IN sz

16 Number of bytes in the *data* blob (integer)

### 17 IN cbdata

18 Data passed into the initial call to [PMIx\\_server\\_dmodex\\_request](#) (memory reference)

### 19 Description

20 Define a function to be called by the PMIx server library for return of information posted by a local  
21 application process (via [PMIx\\_Put](#) with subsequent [PMIx\\_Commit](#)) in response to a request  
22 from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd  
23 upon return from the function.

## 24 17.2.11 PMIx\_server\_setup\_application

### 25 Summary

26 Provide a function by which a launcher can request application-specific setup data prior to launch of  
27 a *job*.



## Format

C

```
pmix_status_t
PMIX_server_setup_application(const pmix_namespace_t nspace,
                             pmix_info_t info[], size_t ninfo,
                             pmix_setup_application_cbfunc_t cbfunc,
                             void *cbdata);
```

C

**IN nspace**  
namespace (string)

**IN info**  
Array of info structures (array of handles)

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

**IN cbfunc**  
Callback function `pmix_setup_application_cbfunc_t` (function reference)

**IN cbdata**  
Data to be passed to the *cbfunc* callback function (memory reference)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

### Required Attributes

PMIx libraries that support this operation are required to support the following:

**PMIX\_SETUP\_APP\_ENVARS** "pmix.setup.env" (bool)  
Harvest and include relevant environmental variables.

**PMIX\_SETUP\_APP\_NONENVARS** "pmix.setup.nenv" (bool)  
Include all relevant data other than environmental variables.

**PMIX\_SETUP\_APP\_ALL** "pmix.setup.all" (bool)  
Include all relevant data.

**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\*)

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_FABRIC_SEC_KEY` "pmix.alloc.nsec" (`pmix_byte_object_t`)  
15 Request that the allocation include a fabric security key for the spawned job.

16 `PMIX_ALLOC_FABRIC_TYPE` "pmix.alloc.nettype" (`char*`)  
17 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

18 `PMIX_ALLOC_FABRIC_PLANE` "pmix.alloc.netplane" (`char*`)  
19 ID string for the *fabric plane* to be used for the requested allocation.

20 `PMIX_ALLOC_FABRIC_ENDPTS` "pmix.alloc.endpts" (`size_t`)  
21 Number of endpoints to allocate per *process* in the job.

22 `PMIX_ALLOC_FABRIC_ENDPTS_NODE` "pmix.alloc.endpts.nd" (`size_t`)  
23 Number of endpoints to allocate per *node* for the job.

24 `PMIX_PROC_MAP` "pmix.pmap" (`char*`)  
25 Regular expression describing processes on each node in the specified realm - see 17.2.3.2  
26 for an explanation of its generation. Defaults to the *job* realm.

27 `PMIX_NODE_MAP` "pmix.nmap" (`char*`)  
28 Regular expression of nodes currently hosting processes in the specified realm - see 17.2.3.2  
29 for an explanation of its generation. Defaults to the *job* realm.

▲-----▲  
▼-----▼ **Optional Attributes** -----▼

30 PMIx libraries that support this operation may support the following:

31 `PMIX_ALLOC_BANDWIDTH` "pmix.alloc.bw" (`float`)  
32 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
33 request.

34 `PMIX_ALLOC_FABRIC_QOS` "pmix.alloc.netqos" (`char*`)  
35 Fabric quality of service level for the job being requested in an allocation request.

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

2 Return information regarding the session realm of the target process. In this context,  
3 indicates that the information provided in the **PMIX\_NODE\_MAP** is for the entire session and  
4 not just the indicated namespace. Thus, subsequent calls to this API may omit node-level  
5 information - e.g., the library may not need to include information on the devices on each  
6 node in a subsequent call.

7 The following optional attributes may be provided by the host environment to identify the  
8 programming model (as specified by the user) being executed within the application. The PMIx  
9 server library may utilize this information to harvest/forward model-specific environmental  
10 variables, record the programming model associated with the application, etc.

- 11 • **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (char\*)  
12 Programming model being initialized (e.g., "MPI" or "OpenMP").
  - 13 • **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (char\*)  
14 Programming model implementation ID (e.g., "OpenMPI" or "MPICH").
  - 15 • **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (char\*)  
16 Programming model version string (e.g., "2.1.1").
- 

## 17 Description

18 Provide a function by which the RM can request application-specific setup data (e.g., environmental  
19 variables, fabric configuration and security credentials) from supporting PMIx server library  
20 subsystems prior to initiating launch of a job.

21 This is defined as a non-blocking operation in case contributing subsystems need to perform some  
22 potentially time consuming action (e.g., query a remote service) before responding. The returned  
23 data must be distributed by the host environment and subsequently delivered to the local PMIx  
24 server on each node where application processes will execute, prior to initiating execution of those  
25 processes.

### ▼ Advice to PMIx server hosts ▼

26 Host environments are required to execute this operation prior to launching a job. In addition to  
27 supported directives, the *info* array must include a description of the *job* using the  
28 **PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes.

29 Note that the function can be called on a per-application basis if the **PMIX\_PROC\_MAP** and  
30 **PMIX\_NODE\_MAP** are provided only for the corresponding application (as opposed to the entire  
31 job) each time.

### ▼ Advice to PMIx library implementers ▼

32 Support for harvesting of environmental variables and providing of local configuration information  
33 by the PMIx implementation is optional.

## 1 17.2.11.1 Server Setup Application Callback Function

2 The `PMIx_server_setup_application` callback function.

### 3 Summary

4 Provide a function by which the resource manager can receive application-specific environmental  
5 variables and other setup data prior to launch of an application.

### 6 *PMIx v2.0* Format

```
typedef void (*pmix_setup_application_cfunc_t) (  
    pmix_status_t status,  
    pmix_info_t info[], size_t ninfo,  
    void *provided_cbdata,  
    pmix_op_cfunc_t cfunc, void *cbdata);
```

#### 12 IN status

13 returned status of the request (`pmix_status_t`)

#### 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 provided\_cbdata

19 Data originally passed to call to `PMIx_server_setup_application` (memory  
20 reference)

#### 21 IN cfunc

22 `pmix_op_cfunc_t` function to be called when processing completed (function reference)

#### 23 IN cbdata

24 Data to be passed to the *cfunc* callback function (memory reference)

### 25 Description

26 Define a function to be called by the PMIx server library for return of application-specific setup  
27 data in response to a request from the host RM. The returned *info* array is owned by the PMIx  
28 server library and will be free'd when the provided *cfunc* is called.

## 29 17.2.11.2 Server Setup Application Attributes

30 *PMIx v3.0*

Attributes specifically defined for controlling contents of application setup data.

31 `PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)`

32 Harvest and include relevant environmental variables.

33 `PMIX_SETUP_APP_NONENVARS ""pmix.setup.nenv" (bool)`

34 Include all relevant data other than environmental variables.

35 `PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)`

36 Include all relevant data.

## 1 17.2.12 `PMIx_Register_attributes`

### 2 Summary

3 Register host environment attribute support for a function.

### 4 *PMIx v4.0* Format

C

```
5 pmix_status_t  
6 PMIx_Register_attributes(char *function,  
7 pmix_regattr_t attrs[],  
8 size_t nattrs);
```

C

#### 9 IN `function`

10 String name of function (string)

#### 11 IN `attrs`

12 Array of `pmix_regattr_t` describing the supported attributes (handle)

#### 13 IN `nattrs`

14 Number of elements in `attrs` (`size_t`)

15 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 16 Description

17 The `PMIx_Register_attributes` function is used by the host environment to register with  
18 its `PMIx` server library the attributes it supports for each `pmix_server_module_t` function.  
19 The *function* is the string name of the server module function (e.g., "register\_events",  
20 "validate\_credential", or "allocate") whose attributes are being registered. See the  
21 `pmix_regattr_t` entry for a description of the `attrs` array elements.

22 Note that the host environment can also query the library (using the `PMIx_Query_info_nb`  
23 API) for its attribute support both at the server, client, and tool levels once the host has executed  
24 `PMIx_server_init` since the server will internally register those values.

### Advice to `PMIx` server hosts

25 Host environments are strongly encouraged to register all supported attributes immediately after  
26 initializing the library to ensure that user requests are correctly serviced.

## Advice to PMIx library implementers

PMIx implementations are *required* to register all internally supported attributes for each API during initialization of the library (i.e., when the process calls their respective PMIx init function). Specifically, the implementation *must not* register supported attributes upon first call to a given API as this would prevent users from discovering supported attributes prior to first use of an API.

It is the implementation's responsibility to associate registered attributes for a given `pmix_server_module_t` function with their corresponding user-facing API. Supported attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by the level (see Section 8.1.5) at which the attribute is supported.

Note that attributes can/will be registered on an API for each level. It is *required* that the implementation support user queries for supported attributes on a per-level basis. Duplicate registrations at the *same* level for a function *shall* return an error - however, duplicate registrations at *different* levels *shall* be independently tracked.

### 17.2.12.1 Attribute registration constants

Constants supporting attribute registration.

**PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION** The attributes for an identical function have already been registered at the specified level (host, server, or client).

### 17.2.12.2 Attribute registration structure

The `pmix_regattr_t` structure is used to register attribute support for a PMIx function.

PMIx v4.0

```
typedef struct pmix_regattr {
    char *name;
    pmix_key_t *string;
    pmix_data_type_t type;
    pmix_info_t *info;
    size_t ninfo;
    char **description;
} pmix_regattr_t;
```

Note that in this structure:

- the *name* is the actual name of the attribute - e.g., "PMIX\_MAX\_PROCS"
- the *string* is the literal string value of the attribute - e.g., "pmix.max.size" for the **PMIX\_MAX\_PROCS** attribute
- *type* must be a PMIx data type identifying the type of data associated with this attribute.

- the *info* array contains machine-usable 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.

### 17.2.12.3 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.

### 17.2.12.4 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

PMIx v4.0

		▼	C	▶
1	<b>PMIX_REGATTR_CONSTRUCT (m)</b>	▲	C	▶
2	<b>IN m</b>			
3	Pointer to the structure to be initialized (pointer to <code>pmix_regattr_t</code> )			
4	<b>Destruct the regattr structure</b>			
5	Destruct the <code>pmix_regattr_t</code> fields, releasing all strings.			
	<i>PMIx v4.0</i>	▼	C	▶
6	<b>PMIX_REGATTR_DESTRUCT (m)</b>	▲	C	▶
7	<b>IN m</b>			
8	Pointer to the structure to be destructed (pointer to <code>pmix_regattr_t</code> )			
9	<b>Create a regattr array</b>			
10	Allocate and initialize an array of <code>pmix_regattr_t</code> structures.			
	<i>PMIx v4.0</i>	▼	C	▶
11	<b>PMIX_REGATTR_CREATE (m, n)</b>	▲	C	▶
12	<b>INOUT m</b>			
13	Address where the pointer to the array of <code>pmix_regattr_t</code> structures shall be stored			
14	(handle)			
15	<b>IN n</b>			
16	Number of structures to be allocated ( <code>size_t</code> )			
17	<b>Free a regattr array</b>			
18	Release an array of <code>pmix_regattr_t</code> structures.			
	<i>PMIx v4.0</i>	▼	C	▶
19	<b>PMIX_REGATTR_FREE (m, n)</b>	▲	C	▶
20	<b>INOUT m</b>			
21	Pointer to the array of <code>pmix_regattr_t</code> structures (handle)			
22	<b>IN n</b>			
23	Number of structures in the array ( <code>size_t</code> )			



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

### Load a regattr structure

Load values into a `pmix_regattr_t` structure. The macro can be called multiple times to add as many strings as desired to the same structure by passing the same address and a `NULL` key to the macro. Note that the `t` type value must be given each time.

```
▼ _____ C _____ ▼  
PMIX_REGATTR_LOAD(a, n, k, t, ni, v)  
▲ _____ C _____ ▲
```

- IN a**  
Pointer to the structure to be loaded (pointer to `pmix_proc_t`)
- IN n**  
String name of the attribute (string)
- IN k**  
Key value to be loaded (`pmix_key_t`)
- IN t**  
Type of data associated with the provided key (`pmix_data_type_t`)
- IN ni**  
Number of `pmix_info_t` elements to be allocated in *info* (`size_t`)
- IN v**  
One-line description to be loaded (more can be added separately) (string)

### Transfer a regattr to another regattr

Non-destructively transfer the contents of a `pmix_regattr_t` structure to another one.

```
PMIx v4.0 ▼ _____ C _____ ▼  
PMIX_REGATTR_XFER(m, n)  
▲ _____ C _____ ▲
```

- INOUT m**  
Pointer to the destination `pmix_regattr_t` structure (handle)
- IN n**  
Pointer to the source `pmix_regattr_t` structure (handle)

## 17.2.13 PMIx\_server\_setup\_local\_support

### Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

## Format

C

```
pmix_status_t
PMIx_server_setup_local_support(const pmix_namespace_t nspace,
                               pmix_info_t info[], size_t ninfo,
                               pmix_op_cbfunc_t cbfunc,
                               void *cbdata);
```

C

### IN **nspace**

Namespace (string)

### IN **info**

Array of info structures (array of handles)

### IN **ninfo**

Number of elements in the *info* array (**size\_t**)

### 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)

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called

If none of the above return codes are appropriate, then an implementation must return either a general **PMIx** error code or an implementation defined error code as described in Section 3.1.1.

## 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**.

1 Host environments must register the *nspc* using `PMIx_server_register_nspace` prior to  
2 calling this API to ensure that all namespace-related information required to support this function is  
3 available to the library. This eliminates the need to include any of the registration information in the  
4 *info* array passed to this API.

## 5 17.2.14 PMIx\_server\_IOF\_deliver

### 6 Summary

7 Provide a function by which the host environment can pass forwarded Input/Output (IO) to the  
8 PMIx server library for distribution to its clients.

### 9 Format

*PMIx v3.0*

```
10 pmix_status_t  
11 PMIx_server_IOF_deliver(const pmix_proc_t *source,  
12                        pmix_iof_channel_t channel,  
13                        const pmix_byte_object_t *bo,  
14                        const pmix_info_t info[], size_t ninfo,  
15                        pmix_op_cbfunc_t cbfunc, void *cbdata);
```

#### 16 IN source

17 Pointer to `pmix_proc_t` identifying source of the IO (handle)

#### 18 IN channel

19 IO channel of the data (`pmix_iof_channel_t`)

#### 20 IN bo

21 Pointer to `pmix_byte_object_t` containing the payload to be delivered (handle)

#### 22 IN info

23 Array of `pmix_info_t` metadata describing the data (array of handles)

#### 24 IN ninfo

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

#### 26 IN cbfunc

27 Callback function `pmix_op_cbfunc_t`. A `NULL` function reference indicates that the  
28 function is to be executed as a blocking operation (function reference)

#### 29 IN cbdata

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

31 A successful return indicates that the request is being processed and the result will be returned in  
32 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
33 from the API. The callback function, *cbfunc*, is only called when `PMIX_SUCCESS` is returned.

34 Returns `PMIX_SUCCESS` or one of the following error codes when the condition described occurs:

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called

If none of the above return codes are appropriate, then an implementation must return either a general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### Description

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients. The PMIx server library is responsible for determining which of its clients have actually registered for the provided data and delivering it. The *cbfunc* callback function will be called once the PMIx server library no longer requires access to the provided data.

## 17.2.15 PMIx\_server\_collect\_inventory

### Summary

Collect inventory of resources on a node.

### Format

PMIx v3.0

```
pmix_status_t
PMIx_server_collect_inventory(const pmix_info_t directives[],
                             size_t ndirs,
                             pmix_info_cbfunc_t cbfunc,
                             void *cbdata);
```

#### IN directives

Array of **pmix\_info\_t** directing the request (array of handles)

#### IN ndirs

Number of elements in the *directives* array (**size\_t**)

#### IN cbfunc

Callback function to return collected data (**pmix\_info\_cbfunc\_t** function reference)

#### IN cbdata

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

A successful return indicates that the request is being processed and the result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

## Description

Provide a function by which the host environment can request its PMIx server library collect an inventory of local resources. Supported resources depends upon the PMIx implementation, but may include the local node topology and fabric interfaces.

### Advice to PMIx server hosts

This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested information. Inventory collection is expected to be a rare event – at system startup and upon command from a system administrator. Inventory updates are expected to initiate a smaller operation involving only the changed information. For example, replacement of a node would generate an event to notify the scheduler with an inventory update without invoking a global inventory operation.

## 17.2.16 PMIx\_server\_deliver\_inventory

### Summary

Pass collected inventory to the PMIx server library for storage.

### Format

PMIx v3.0

```
pmix_status_t
PMIx_server_deliver_inventory(const pmix_info_t info[],
                              size_t ninfo,
                              const pmix_info_t directives[],
                              size_t ndirs,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata);
```

- IN info**  
Array of `pmix_info_t` containing the inventory (array of handles)
- IN ninfo**  
Number of elements in the *info* array (`size_t`)
- IN directives**  
Array of `pmix_info_t` directing the request (array of handles)
- IN ndirs**  
Number of elements in the *directives* array (`size_t`)
- 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 Returns one of the following:

2 A successful return indicates that the request is being processed and the result will be returned in  
 3 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
 4 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

5 Returns PMIX\_SUCCESS or one of the following error codes when the condition described occurs:

- 6 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
 7 returned *success* - the *cbfunc* will not be called

8 If none of the above return codes are appropriate, then an implementation must return either a  
 9 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

### 10 Description

11 Provide a function by which the host environment can pass inventory information obtained from a  
 12 node (as a result of a call to **PMIx\_server\_collect\_inventory**) to the PMIx server library  
 13 for storage. Inventory data is subsequently used by the PMIx server library for allocations in  
 14 response to **PMIx\_server\_setup\_application**, and may be available to the library's host  
 15 via the **PMIx\_Get** API (depending upon PMIx implementation). The *cbfunc* callback function  
 16 will be called once the PMIx server library no longer requires access to the provided data.

## 17 17.2.17 PMIx\_server\_generate\_locality\_string

### 18 Summary

19 Generate a PMIx locality string from a given cpuset.

### 20 Format

C

```
21 pmix_status_t
22 PMIx_server_generate_locality_string(const pmix_cpuset_t *cpuset,
23                                     char **locality);
```

C

### 24 IN cpuset

25 Pointer to a **pmix\_cpuset\_t** containing the bitmap of assigned PUs (handle)

### 26 OUT locality

27 String representation of the PMIx locality corresponding to the input bitmap (**char\***)

28 A successful return indicates that the returned string contains the generated locality string.

29 Returns **PMIX\_SUCCESS** or a negative value indicating the error.

## Description

Provide a function by which the host environment can generate a PMIx locality string for inclusion in the call to `PMIx_server_register_namespace`. This function shall only be called for local client processes, with the returned locality included in the job-level information (via the `PMIX_LOCALITY_STRING` attribute) provided to local clients. Local clients can use these strings as input to determine the relative locality of their local peers via the `PMIx_Get_relative_locality` 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 corresponding locality as expressed by the underlying implementation.

## 17.2.18 `PMIx_server_generate_cpuset_string`

### Summary

Generate a PMIx string representation of the provided *cpuset*.

### Format

*PMIx v4.0*

```
pmix_status_t
PMIx_server_generate_cpuset_string(const pmix_cpuset_t *cpuset,
                                  char **cpuset_string);
```

### 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*`)

A successful return indicates that the returned string contains the generated *cpuset* representation string.

Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 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_namespace`. This function shall only be called for local client processes, with the returned string included in the job-level information (via the `PMIX_CPUSSET` attribute) provided to local clients. Local clients can use these strings as input to obtain their PU bindings via the `PMIx_Parse_cpuset_string` 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.

## 1 17.2.18.1 Cpuset Structure

2 The `pmix_cpuset_t` structure contains a character string identifying the source of the bitmap  
3 (e.g., "hwloc") and a pointer to the corresponding implementation-specific structure (e.g.,  
4 `hwloc_cpuset_t`).

*PMIx v4.0*

```
▼ C _____ ▼  
typedef struct pmix_cpuset {  
    char *source;  
    void *bitmap;  
} pmix_cpuset_t;  
▲ C _____ ▲
```

## 9 17.2.18.2 Cpuset support macros

10 The following macros support the `pmix_cpuset_t` structure.

### 11 Initialize the cpuset structure

12 Initialize the `pmix_cpuset_t` fields.

*PMIx v4.0*

```
▼ C _____ ▼  
PMIX_CPUSSET_CONSTRUCT (m)  
▲ C _____ ▲
```

14 **IN** m

15 Pointer to the structure to be initialized (pointer to `pmix_cpuset_t`)

### 16 Destruct the cpuset structure

17 Destruct the `pmix_cpuset_t` fields.

*PMIx v4.0*

```
▼ C _____ ▼  
PMIX_CPUSSET_DESTRUCT (m)  
▲ C _____ ▲
```

19 **IN** m

20 Pointer to the structure to be destructed (pointer to `pmix_cpuset_t`)

### 21 Create a cpuset array

22 Allocate and initialize a `pmix_cpuset_t` array.

*PMIx v4.0*

```
▼ C _____ ▼  
PMIX_CPUSSET_CREATE (m, n)  
▲ C _____ ▲
```

24 **INOUT** m

25 Address where the pointer to the array of `pmix_cpuset_t` structures shall be stored  
26 (handle)

27 **IN** n

28 Number of structures to be allocated (size\_t)



1 **Release a cpuset array**  
2 Deconstruct and free a `pmix_cpuset_t` array.

*PMIx v4.0*

▼ `PMIX_CPUSSET_FREE(m, n)` C

3 `PMIX_CPUSSET_FREE(m, n)`

4 **INOUT** `m`

5 Address the array of `pmix_cpuset_t` structures to be released (handle)

6 **IN** `n`

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

## 8 17.2.19 `PMIx_server_define_process_set`

### 9 **Summary**

10 Define a PMIx process set.

### 11 **Format**

*PMIx v4.0*

▼ `pmix_status_t PMIx_server_define_process_set(const pmix_proc_t members[], size_t nmembers, char *pset_name);` C

12 `pmix_status_t`

13 `PMIx_server_define_process_set(const pmix_proc_t members[],`  
14 `size_t nmembers,`  
15 `char *pset_name);`

16 **IN** `members`

17 Pointer to an array of `pmix_proc_t` containing the identifiers of the processes in the  
18 process set (handle)

19 **IN** `nmembers`

20 Number of elements in `members` (integer)

21 **IN** `pset_name`

22 String name of the process set being defined (`char*`)

23 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 24 **Description**

25 Provide a function by which the host environment can create a process set. The PMIx server shall  
26 alert all local clients of the new process set (including process set name and membership) via the  
27 `PMIX_PROCESS_SET_DEFINE` event.

#### ▼ **Advice to PMIx server hosts** ▼

28 The host environment is responsible for ensuring:

- 29
- 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
- 30
- 31

## 1 17.2.20 PMIx\_server\_delete\_process\_set

### 2 Summary

3 Delete a PMIx process set name

### 4 Format

*PMIx v4.0*

C

5 `pmix_status_t`

6 `PMIx_server_delete_process_set(char *pset_name);`

C

7 **IN** `pset_name`

8 String name of the process set being deleted (`char*`)

9 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 10 Description

11 Provide a function by which the host environment can delete a process set name. The PMIx server  
12 shall alert all local clients of the process set name being deleted via the  
13 `PMIX_PROCESS_SET_DELETE` event. Deletion of the name has no impact on the member  
14 processes.

#### Advice to PMIx server hosts

15 The host environment is responsible for ensuring consistent knowledge of process set membership  
16 across all involved PMIx servers.

## 17 17.3 Server Function Pointers

18 PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the  
19 protocol. This method allows RMs to implement the server without being burdened with PMIx  
20 internal details. When a request is received from the client, the corresponding server function will  
21 be called with the information.

22 Any functions not supported by the RM can be indicated by a `NULL` for the function pointer. PMIx  
23 implementations are required to return a `PMIX_ERR_NOT_SUPPORTED` status to all calls to  
24 functions that require host environment support and are not backed by a corresponding server  
25 module entry. Host environments may, if they choose, include a function pointer for operations they  
26 have not yet implemented and simply return `PMIX_ERR_NOT_SUPPORTED`.

27 Functions that accept directives (i.e., arrays of `pmix_info_t` structures) must check any provided  
28 directives for those marked as *required* via the `PMIX_INFO_REQD` flag. PMIx client and server  
29 libraries are required to mark any such directives with the `PMIX_INFO_REQD_PROCESSED` flag  
30 should they have handled the request. Any required directive that has not been marked therefore  
31 becomes the responsibility of the host environment. If a required directive that hasn't been

1 processed by a lower level cannot be supported by the host, then the  
2 `PMIX_ERR_NOT_SUPPORTED` error constant must be returned. If the directive can be processed  
3 by the host, then the host shall do so and mark the attribute with the  
4 `PMIX_INFO_REQD_PROCESSED` flag.

5 The host RM will provide the function pointers in a `pmix_server_module_t` structure passed  
6 to `PMIx_server_init`. The module structure and associated function references are defined in  
7 this section.

### Advice to PMIx server hosts

8 For performance purposes, the host server is required to return as quickly as possible from all  
9 functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx  
10 server support library to handle multiple client requests as quickly and scalably as possible.

11 All data passed to the host server functions is “owned” by the PMIx server support library and  
12 must not be free'd. Data returned by the host server via callback function is owned by the host  
13 server, which is free to release it upon return from the callback

## 14 17.3.1 `pmix_server_module_t` Module

### 15 Summary

16 List of function pointers that a PMIx server passes to `PMIx_server_init` during startup.

### 17 Format

```
18 typedef struct pmix_server_module_4_0_0_t {  
19     /* v1x interfaces */  
20     pmix_server_client_connected_fn_t    client_connected;    // DEPRECATED  
21     pmix_server_client_finalized_fn_t    client_finalized;  
22     pmix_server_abort_fn_t               abort;  
23     pmix_server_fence_nb_fn_t           fence_nb;  
24     pmix_server_dmodex_req_fn_t         direct_modex;  
25     pmix_server_publish_fn_t            publish;  
26     pmix_server_lookup_fn_t             lookup;  
27     pmix_server_unpublish_fn_t          unpublish;  
28     pmix_server_spawn_fn_t              spawn;  
29     pmix_server_connect_fn_t            connect;  
30     pmix_server_disconnect_fn_t         disconnect;  
31     pmix_server_register_events_fn_t    register_events;  
32     pmix_server_deregister_events_fn_t  deregister_events;  
33     pmix_server_listener_fn_t           listener;  
34     /* v2x interfaces */  
35     pmix_server_notify_event_fn_t       notify_event;
```

```

1     pmix_server_query_fn_t           query;
2     pmix_server_tool_connection_fn_t tool_connected;
3     pmix_server_log_fn_t            log;
4     pmix_server_alloc_fn_t          allocate;
5     pmix_server_job_control_fn_t     job_control;
6     pmix_server_monitor_fn_t         monitor;
7     /* v3x interfaces */
8     pmix_server_get_cred_fn_t        get_credential;
9     pmix_server_validate_cred_fn_t   validate_credential;
10    pmix_server_iof_fn_t              iof_pull;
11    pmix_server_stdin_fn_t            push_stdin;
12    /* v4x interfaces */
13    pmix_server_grp_fn_t               group;
14    pmix_server_fabric_fn_t           fabric;
15    pmix_server_client_connected2_fn_t client_connected2;
16 } pmix_server_module_t;

```

C

Advice to PMIx server hosts

17 Note that some PMIx implementations *require* the use of C99-style designated initializers to clearly  
18 correlate each provided function pointer with the correct member of the  
19 [pmix\\_server\\_module\\_t](#) structure as the location/ordering of struct members may change over  
20 time.

## 21 17.3.2 pmix\_server\_client\_connected\_fn\_t

### 22 Summary

23 Notify the host server that a client connected to this server. This function module entry has been  
24 **DEPRECATED** in favor of [pmix\\_server\\_client\\_connected2\\_fn\\_t](#).

## Format

```
typedef pmix_status_t (*pmix_server_client_connected_fn_t) (  
    const pmix_proc_t *proc,  
    void* server_object,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata);
```

### IN `proc`

`pmix_proc_t` structure (handle)

### IN `server_object`

object reference (memory reference)

### 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 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

### Description

This function module entry has been DEPRECATED in favor of `pmix_server_client_connected2_fn_t`. If both functions are provided, the PMIx library will ignore this function module entry in favor of its replacement.

## 17.3.3 `pmix_server_client_connected2_fn_t`

### Summary

Notify the host server that a client connected to this server - this version of the original function definition has been extended to include an array of `pmix_info_t`, thereby allowing the PMIx server library to pass additional information identifying the client to the host environment.

## Format

C

```
typedef pmix_status_t (*pmix_server_client_connected2_fn_t) (  
    const pmix_proc_t *proc,  
    void* server_object,  
    pmix_info_t info[], size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

C

- IN** **proc**  
**pmix\_proc\_t** structure (handle)
- IN** **server\_object**  
object reference (memory reference)
- 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:

- **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. The PMIx server library is to immediately terminate the connection.

## Description

Notify the host environment that a client has called **PMIx\_Init**. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The *server\_object* parameter will be the value of the *server\_object* parameter passed to **PMIx\_server\_register\_client** by the host server when registering the connecting client. A host server can choose to not be notified when clients connect by setting **pmix\_server\_client\_connected2\_fn\_t** to **NULL**.

It is possible that only a subset of the clients in a namespace call **PMIx\_Init**. The server's **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 17.3.4 `pmix_server_client_finalized_fn_t`

### 6 Summary

7 Notify the host environment that a client called `PMIx_Finalize`.

### 8 Format

*PMIx v1.0*

C

```
9 typedef pmix_status_t (*pmix_server_client_finalized_fn_t) (  
10     const pmix_proc_t *proc,  
11     void* server_object,  
12     pmix_op_cbfunc_t cbfunc,  
13     void *cbdata);
```

C

#### 14 IN `proc`

15 `pmix_proc_t` structure (handle)

#### 16 IN `server_object`

17 object reference (memory reference)

#### 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 by the host environment - result  
24 will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function  
25 prior to returning 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 • a `PMIx` error constant indicating either an error in the input or that the request was immediately  
29 processed and failed - the `cbfunc` will not be called

## Description

Notify the host environment that a client called `PMIx_Finalize`. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The `server_object` parameter will be the value of the `server_object` parameter passed to `PMIx_server_register_client` by the host server when registering the connecting client. If provided, an implementation of `pmix_server_client_finalized_fn_t` is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting `pmix_server_client_finalized_fn_t` to `NULL`.

Note that the host server is only being informed that the client has called `PMIx_Finalize`. The client might not have exited. If a client exits without calling `PMIx_Finalize`, the server support library will not call the `pmix_server_client_finalized_fn_t` implementation.

### Advice to PMIx server hosts

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.

## 17.3.5 `pmix_server_abort_fn_t`

### Summary

Notify the host environment that a local client called `PMIx_Abort`.

### Format

C

```
typedef pmix_status_t (*pmix_server_abort_fn_t) (
    const pmix_proc_t *proc,
    void *server_object,
    int status,
    const char msg[],
    pmix_proc_t procs[],
    size_t nprocs,
    pmix_op_cbfnc_t cbfunc,
    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 • **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 host must not invoke the callback function  
 21 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
- 24 • **PMIX\_ERR\_PARAM\_VALUE\_NOT\_SUPPORTED** indicating that the host environment supports  
 25 this API, but the request includes processes that the host environment cannot abort - e.g., if the  
 26 request is to abort subsets of processes from a namespace, or processes outside of the caller's  
 27 own namespace, and the host environment does not permit such operations. In this case, none of  
 28 the specified processes will be terminated - 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

## Description

A local client called `PMIx_Abort`. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server library to release the client. The array of `procs` indicates which processes are to be terminated. A `NULL` for the `procs` array indicates that all processes in the caller's namespace are to be aborted, including itself - this is the equivalent of passing a `pmix_proc_t` array element containing the caller's namespace and a rank value of `PMIX_RANK_WILDCARD`.

## 17.3.6 pmix\_server\_fencebn\_fn\_t

### Summary

At least one client called either `PMIx_Fence` or `PMIx_Fence_nb`.

### Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_fencebn_fn_t) (  
    const pmix_proc_t procs[],  
    size_t nprocs,  
    const pmix_info_t info[],  
    size_t ninfo,  
    char *data, size_t ndata,  
    pmix_modex_cbfunc_t cbfunc,  
    void *cbdata);
```

C

- IN procs**  
Array of `pmix_proc_t` structures identifying operation participants(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 data**  
(string)
- IN ndata**  
(integer)
- IN cbfunc**  
Callback function `pmix_modex_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 host must not invoke the callback function  
3 prior to returning from the API.
- 4 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
5 request, even though the function entry was provided in the server module - the *cbfunc* will not  
6 be called
- 7 • a PMIx error constant indicating either an error in the input or that the request was immediately  
8 processed and failed - the *cbfunc* will not be called

### Required Attributes

9 PMIx libraries are required to pass any provided attributes to the host environment for processing.

10 The following attributes are required to be supported by all host environments:

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

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

### Optional Attributes

17 The following attributes are optional for host environments:

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

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

### Advice to PMIx server hosts

22 Host environment are required to return **PMIX\_ERR\_NOT\_SUPPORTED** if passed an attributed  
23 marked as **PMIX\_INFO\_REQD** that they do not support, even if support for that attribute is  
24 optional.

## Description

All local clients in the provided array of *procs* called either `PMIx_Fence` or `PMIx_Fence_nb`. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the `PMIx_Fence/PMIx_Fence_nb` operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's `pmix_server_fence_nb_fn_t` function.

The provided data is to be collectively shared with all PMIx servers involved in the fence operation, and returned in the modex *cbfunc*. A **NULL** data value indicates that the local processes had no data to contribute.

The array of *info* structs is used to pass user-requested options to the server. This can include directives as to the algorithm to be used to execute the fence operation. The directives are optional unless the `PMIX_INFO_REQD` flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

### Advice to PMIx library implementers

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. 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. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

```
uint8_t *blob1, *blob2, *total;
size_t sz_blob1, sz_blob2, sz_total;

sz_total = sz_blob1 + sz_blob2;
total = (uint8_t*)malloc(sz_total);
memcpy(total, blob1, sz_blob1);
memcpy(&total[sz_blob1], blob2, sz_blob2);
```

Note that the ordering of the data blobs does not matter. The host is responsible for free'ing the *data* object passed to it by the PMIx server library.

## 1 17.3.6.1 Modex Callback Function

### 2 Summary

3 The `pmix_modex_cbfunc_t` is used by the `pmix_server_fencefn_t` and  
4 `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex Business Card  
5 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

### 12 IN status

13 Status associated with the operation (handle)

### 14 IN data

15 Data to be passed (pointer)

### 16 IN ndata

17 size of the data (`size_t`)

### 18 IN cbdata

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

### 20 IN release\_fn

21 Callback for releasing *data* (function pointer)

### 22 IN release\_cbdata

23 Pointer to be passed to *release\_fn* (memory reference)

### 24 Description

25 A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data  
26 in response to “fence” and “get” operations. The returned blob contains the data collected from  
27 each server participating in the operation.

## 28 17.3.7 pmix\_server\_dmodex\_req\_fn\_t

### 29 Summary

30 Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
31 hosts the specified process to obtain and return a direct modex blob for that process.

## Format

```
typedef pmix_status_t (*pmix_server_dmodex_req_fn_t) (  
    const pmix_proc_t *proc,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_modex_cbfunc_t cbfunc,  
    void *cbdata);
```

### IN `proc`

`pmix_proc_t` structure identifying the process whose data is being requested (handle)

### IN `info`

Array of `info` structures (array of handles)

### IN `ninfo`

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

### IN `cbfunc`

Callback function `pmix_modex_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 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 `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 required to pass any provided attributes to the host environment for processing.

All host environments are required to support the following attributes:

**PMIX\_REQUIRED\_KEY** "pmix.req.key" (`char*`)

Identifies a key that must be included in the requested information. If the specified key is not already available, then the PMIx servers are required to delay response to the `dmodex` request until either the key becomes available or the request times out.

## 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

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to **PMIx\_Put** and **PMIx\_Commit**.

The array of *info* structs is used to pass user-requested options to the server. This can include a timeout to preclude an indefinite wait for data that may never become available. The directives are optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

### 17.3.7.1 Dmodex attributes

**PMIX\_REQUIRED\_KEY** "pmix.req.key" (char\*)

Identifies a key that must be included in the requested information. If the specified key is not already available, then the PMIx servers are required to delay response to the dmodex request until either the key becomes available or the request times out.

### 17.3.8 pmix\_server\_publish\_fn\_t

#### Summary

Publish data per the PMIx API specification.

#### Format

C

```
typedef pmix_status_t (*pmix_server_publish_fn_t) (
    const pmix_proc_t *proc,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```

PMIx v1.0

1 **IN** `proc`  
 2 `pmix_proc_t` structure of the process publishing the data (handle)  
 3 **IN** `info`  
 4 Array of info structures (array of handles)  
 5 **IN** `ninfo`  
 6 Number of elements in the `info` array (integer)  
 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)

11 Returns one of the following:

- 12 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
 13 will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function  
 14 prior to returning from the API.
- 15 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
 16 returned `success` - the `cbfunc` will not be called
- 17 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
 18 request, even though the function entry was provided in the server module - the `cbfunc` will not  
 19 be called
- 20 • a PMIx error constant indicating either an error in the input or that the request was immediately  
 21 processed and failed - the `cbfunc` will not be called

### Required Attributes

22 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
 23 In addition, the following attributes are required to be included in the passed `info` array:

24 **PMIX\_USERID** "`pmix.euid`" (`uint32_t`)

25 Effective user ID of the connecting process.

26 **PMIX\_GRPID** "`pmix.egid`" (`uint32_t`)

27 Effective group ID of the connecting process.

---

28  
 29 Host environments that implement this entry point are required to support the following attributes:

30 **PMIX\_RANGE** "`pmix.range`" (`pmix_data_range_t`)

31 Define constraints on the processes that can access the provided data. Only processes that  
 32 meet the constraints are allowed to access it.

33 **PMIX\_PERSISTENCE** "`pmix.persist`" (`pmix_persistence_t`)



1 Declare how long the datastore shall retain the provided data. The datastore is to delete the  
2 data upon reaching the persistence criterion.

### Optional Attributes

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

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

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

### Description

8 Publish data per the **PMIx\_Publish** specification. The callback is to be executed upon  
9 completion of the operation. The default data range is left to the host environment, but expected to  
10 be **PMIX\_RANGE\_SESSION**, and the default persistence **PMIX\_PERSIST\_SESSION** or their  
11 equivalent. These values can be specified by including the respective attributed in the *info* array.

12 The persistence indicates how long the server should retain the data.

### Advice to PMIx server hosts

13 The host environment is not required to guarantee support for any specific range - i.e., the  
14 environment does not need to return an error if the data store doesn't support a specified range so  
15 long as it is covered by some internally defined range. However, the server must return an error (a)  
16 if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of  
17 published info by the original publisher - it is left to the discretion of the host environment to allow  
18 info-key-based flags to modify this behavior.

19 The **PMIX\_USERID** and **PMIX\_GRPID** of the publishing process will be provided to support  
20 authorization-based access to published information and must be returned on any subsequent  
21 lookup request.  
22

## 23 17.3.9 pmix\_server\_lookup\_fn\_t

### 24 Summary

25 Lookup published data.

## Format

```
typedef pmix_status_t (*pmix_server_lookup_fn_t) (  
    const pmix_proc_t *proc,  
    char **keys,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_lookup_cbfunc_t cbfunc,  
    void *cbdata);
```

### IN `proc`

`pmix_proc_t` structure of the process seeking the data (handle)

### IN `keys`

(array of strings)

### IN `info`

Array of info structures (array of handles)

### IN `ninfo`

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

### IN `cbfunc`

Callback function `pmix_lookup_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 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

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:

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user ID of the connecting process.

1 **PMIX\_GRPID** "pmix.egid" (uint32\_t)  
2 Effective group ID of the connecting process.

3  
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.

8 **PMIX\_WAIT** "pmix.wait" (int)  
9 Caller requests that the PMIx server wait until at least the specified number of values are  
10 found (a value of zero indicates *all* and is the default).

11  
12   
13  Optional Attributes 

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

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

19 **Description**

20 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys  
21 identifying the data being requested.

22 The array of *info* structs is used to pass user-requested options to the server. The default data range  
23 is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**. This can include a  
24 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.

25  Advice to PMIx server hosts 

26 The **PMIX\_USERID** and **PMIX\_GRPID** of the requesting process will be provided to support  
27 authorization-based access to published information. The host environment is not required to  
28 guarantee support for any specific range - i.e., the environment does not need to return an error if  
29 the data store doesn't support a specified range so long as it is covered by some internally defined  
range.

## 1 17.3.10 pmix\_server\_unpublish\_fn\_t

### 2 Summary

3 Delete data from the data store.

### 4 Format

PMIx v1.0

C

```
5 typedef pmix_status_t (*pmix_server_unpublish_fn_t) (  
6     const pmix_proc_t *proc,  
7     char **keys,  
8     const pmix_info_t info[],  
9     size_t ninfo,  
10    pmix_op_cbfunc_t cbfunc,  
11    void *cbdata);
```

C

12 **IN** `proc`  
13 `pmix_proc_t` structure identifying the process making the request (handle)

14 **IN** `keys`  
15 (array of strings)

16 **IN** `info`  
17 Array of info structures (array of handles)

18 **IN** `ninfo`  
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

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:

**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.

Host environments that implement this entry point are required to support the following attributes:

**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.

## 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

Delete data from the data store. The host server will be passed a **NULL**-terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The default data range is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**. The callback is to be executed upon completion of the delete procedure.

## 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.

## 1 17.3.11 pmix\_server\_spawn\_fn\_t

### 2 Summary

3 Spawn a set of applications/processes as per the [PMIx\\_Spawn](#) API.

### 4 Format

C

```
5 typedef pmix_status_t (*pmix_server_spawn_fn_t) (  
6     const pmix_proc_t *proc,  
7     const pmix_info_t job_info[],  
8     size_t ninfo,  
9     const pmix_app_t apps[],  
10    size_t napps,  
11    pmix_spawn_cbfunc_t cbfunc,  
12    void *cbdata);
```

C

- 13 **IN** `proc`  
14 [pmix\\_proc\\_t](#) structure of the process making the request (handle)
- 15 **IN** `job_info`  
16 Array of info structures (array of handles)
- 17 **IN** `ninfo`  
18 Number of elements in the *jobinfo* array (integer)
- 19 **IN** `apps`  
20 Array of [pmix\\_app\\_t](#) structures (array of handles)
- 21 **IN** `napps`  
22 Number of elements in the *apps* array (integer)
- 23 **IN** `cbfunc`  
24 Callback function [pmix\\_spawn\\_cbfunc\\_t](#) (function reference)
- 25 **IN** `cbdata`  
26 Data to be passed to the callback function (memory reference)

27 Returns one of the following:

- 28
- 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 host must not invoke the callback function  
31 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  
processed and failed - the *cbfunc* will not be called

## Required Attributes

PMIx server 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:

**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\_SPAWNED** "pmix.spawned" (bool)

**true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.

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

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

**PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)

The requesting process is a PMIx tool.

**PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)

The requesting process is a PMIx client.

---

Host environments that provide this module entry point are required to pass the **PMIX\_SPAWNED** and **PMIX\_PARENT\_ID** attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job\_info* or the *info* array of an element of the *apps* array:

**PMIX\_WDIR** "pmix.wdir" (char\*)

Working directory for spawned processes.

**PMIX\_SET\_SESSION\_CWD** "pmix.ssn cwd" (bool)

Set the current working directory to the session working directory assigned by the RM - can be assigned to the entire job (by including attribute in the *job\_info* array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.

**PMIX\_PREFIX** "pmix.prefix" (char\*)

Prefix to use for starting spawned processes - i.e., the directory where the executables can be found.

**PMIX\_HOST** "pmix.host" (char\*)

Comma-delimited list of hosts to use for spawned processes.

**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.preloadfiles" (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\_DISPLAY\_MAP** "pmix.dispmap" (bool)

15 Display process mapping upon spawn.

16 **PMIX\_PPR** "pmix.ppr" (char\*)

17 Number of processes to spawn on each identified resource.

18 **PMIX\_MAPBY** "pmix.mapby" (char\*)

19 Process mapping policy - when accessed using **PMIx\_Get**, use the  
20 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
21 provided namespace. Supported values are launcher specific.

22 **PMIX\_RANKBY** "pmix.rankby" (char\*)

23 Process ranking policy - when accessed using **PMIx\_Get**, use the  
24 **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
25 provided namespace. Supported values are launcher specific.

26 **PMIX\_BINDTO** "pmix.bindto" (char\*)

27 Process binding policy - when accessed using **PMIx\_Get**, use the  
28 **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
29 provided namespace. Supported values are launcher specific.

30 **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)

31 Spawned process rank that is to receive any forwarded **stdin**.

32 **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (pmix\_rank\_t)



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 remains  
3 available. A rank of **PMIX\_RANK\_WILDCARD** indicates that all processes in the spawned  
4 job are potential recipients. The requester will issue a call to **PMIx\_IOF\_push** to initiate  
5 the actual forwarding of information to specified targets - this attribute simply requests that  
6 the IL retain the ability to forward the information to the designated targets.

7 **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)

8 Requests that the ability to forward the **stdout** of the spawned processes be maintained.  
9 The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
10 other options for delivery of the forwarded output.

11 **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)

12 Requests that the ability to forward the **stderr** of the spawned processes be maintained.  
13 The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
14 other options for delivery of the forwarded output.

15 **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)

16 Included in the **pmix\_info\_t** array of a **pmix\_app\_t**, this attribute declares that the  
17 application consists of debugger daemons and shall be governed accordingly. If used as the  
18 sole **pmix\_app\_t** in a **PMIx\_Spawn** request, then the **PMIX\_DEBUG\_TARGET** attribute  
19 must also be provided (in either the *job\_info* or in the *info* array of the **pmix\_app\_t**) to  
20 identify the namespace to be debugged so that the launcher can determine where to place the  
21 spawned daemons. If neither **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** nor  
22 **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** is specified, then the launcher shall default to a  
23 placement policy of one daemon per process in the target job.

24 **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)

25 Tag **stdout/stderr** with the identity of the source process - can be assigned to the entire  
26 job (by including attribute in the *job\_info* array) or on a per-application basis in the *info*  
27 array for each **pmix\_app\_t**.

28 **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)

29 Timestamp output - can be assigned to the entire job (by including attribute in the *job\_info*  
30 array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.

31 **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)

32 Merge **stdout** and **stderr** streams - can be assigned to the entire job (by including  
33 attribute in the *job\_info* array) or on a per-application basis in the *info* array for each  
34 **pmix\_app\_t**.

35 **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)

36 Direct output (both **stdout** and **stderr**) into files of form "<filename>.rank" - can be  
37 assigned to the entire job (by including attribute in the *job\_info* array) or on a per-application  
38 basis in the *info* array for each **pmix\_app\_t**.

39 **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)

1 Mark the `argv` with the rank of the process.

2 **PMIX\_CPUS\_PER\_PROC** "`pmix.cpusperproc`" (`uint32_t`)

3 Number of PUs to assign to each rank - when accessed using `PMIx_Get`, use the  
4 **PMIX\_RANK\_WILDCARD** value for the rank to discover the PUs/process assigned to the  
5 provided namespace.

6 **PMIX\_NO\_PROCS\_ON\_HEAD** "`pmix.nolocal`" (`bool`)

7 Do not place processes on the head node.

8 **PMIX\_NO\_OVERSUBSCRIBE** "`pmix.noover`" (`bool`)

9 Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a  
10 node.

11 **PMIX\_REPORT\_BINDINGS** "`pmix.rebind`" (`bool`)

12 Report bindings of the individual processes.

13 **PMIX\_CPU\_LIST** "`pmix.cpulist`" (`char*`)

14 List of PUs to use for this job - when accessed using `PMIx_Get`, use the  
15 **PMIX\_RANK\_WILDCARD** value for the rank to discover the PU list used for the provided  
16 namespace.

17 **PMIX\_JOB\_RECOVERABLE** "`pmix.recover`" (`bool`)

18 Application supports recoverable operations.

19 **PMIX\_JOB\_CONTINUOUS** "`pmix.continuous`" (`bool`)

20 Application is continuous, all failed processes should be immediately restarted.

21 **PMIX\_MAX\_RESTARTS** "`pmix.maxrestarts`" (`uint32_t`)

22 Maximum number of times to restart a process - when accessed using `PMIx_Get`, use the  
23 **PMIX\_RANK\_WILDCARD** value for the rank to discover the max restarts for the provided  
24 namespace.

25 **PMIX\_TIMEOUT** "`pmix.timeout`" (`int`)

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



## 29 Description

30 Spawn a set of applications/processes as per the `PMIx_Spawn` API. Note that applications are not  
31 required to be MPI or any other programming model. Thus, the host server cannot make any  
32 assumptions as to their required support. The callback function is to be executed once all processes  
33 have been started. An error in starting any application or process in this request shall cause all  
34 applications and processes in the request to be terminated, and an error returned to the originating  
35 caller.

36 Note that a timeout can be specified in the `job_info` array to indicate that failure to start the  
37 requested job within the given time should result in termination to avoid hangs.

### 1 17.3.11.1 Server spawn attributes

2 **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)

3 The requesting process is a PMIx tool.

4 **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)

5 The requesting process is a PMIx client.

### 6 17.3.12 pmix\_server\_connect\_fn\_t

#### 7 Summary

8 Record the specified processes as *connected*.

#### 9 Format

*PMIx v1.0*

C

```
10 typedef pmix_status_t (*pmix_server_connect_fn_t) (  
11     const pmix_proc_t procs[],  
12     size_t nprocs,  
13     const pmix_info_t info[],  
14     size_t ninfo,  
15     pmix_op_cbfunc_t cbfunc,  
16     void *cbdata);
```

C

17 **IN procs**  
18 Array of [pmix\\_proc\\_t](#) structures identifying participants (array of handles)

19 **IN nprocs**  
20 Number of elements in the *procs* array (integer)

21 **IN info**  
22 Array of info structures (array of handles)

23 **IN ninfo**  
24 Number of elements in the *info* array (integer)

25 **IN cbfunc**  
26 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

27 **IN cbdata**  
28 Data to be passed to the callback function (memory reference)

29 Returns one of the following:

- 30
- 31 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
32 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
prior to returning from the API.
  - 33 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
34 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

PMIx libraries are required to pass any provided attributes to the host environment for processing.

### 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

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix\_server\_connect\_fn\_t** function, and the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

### Advice to PMIx library implementers

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. 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.

## 17.3.13 pmix\_server\_disconnect\_fn\_t

### Summary

Disconnect a previously connected set of processes.

## Format

C

```
typedef pmix_status_t (*pmix_server_disconnect_fn_t) (  
    const pmix_proc_t procs[],  
    size_t nprocs,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata);
```

C

### IN **procs**

Array of [pmix\\_proc\\_t](#) structures identifying participants (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:

- [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

PMIx libraries are required to pass any provided attributes to the host environment for processing.

## 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

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server's `pmix_server_disconnect_fn_t` function, and the host environment has completed any required supporting operations.

### Advice to PMIx library implementers

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. 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.

A **PMIX\_ERR\_INVALID\_OPERATION** error must be returned if the specified set of *procs* was not previously *connected* via a call to the `pmix_server_connect_fn_t` function.

## 17.3.14 pmix\_server\_register\_events\_fn\_t

### Summary

Register to receive notifications for the specified events.

## Format

C

```
typedef pmix_status_t (*pmix_server_register_events_fn_t) (  
    pmix_status_t *codes,  
    size_t ncodes,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata);
```

C

### IN codes

Array of `pmix_status_t` values (array of handles)

### IN ncodes

Number of elements in the `codes` 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:

- **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

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:

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user ID of the connecting process.

1 **PMIX\_GRPID** "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).

## 19 **17.3.15 pmix\_server\_deregister\_events\_fn\_t**

### 20 **Summary**

21 Deregister to receive notifications for the specified events.



## Format

C

```
typedef pmix_status_t (*pmix_server_deregister_events_fn_t) (  
    pmix_status_t *codes,  
    size_t ncodes,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata);
```

C

### IN codes

Array of `pmix_status_t` values (array of handles)

### IN ncodes

Number of elements in the `codes` 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:

- **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

### Description

Deregister to receive notifications for the specified events to which the PMIx server has previously registered.

### Advice to PMIx library implementers

The PMIx server library must track all client registrations. This module function shall only be called when:

- the library is deregistering environmental codes (i.e., a PMIx codes in the range between **PMIX\_EVENT\_SYS\_BASE** and **PMIX\_EVENT\_SYS\_OTHER**, inclusive) or codes that lies outside the defined PMIx range of constants; and

- no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it remain.

## 17.3.16 `pmix_server_notify_event_fn_t`

### Summary

Notify the specified processes of an event.

### Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_notify_event_fn_t) (  
    pmix_status_t code,  
    const pmix_proc_t *source,  
    pmix_data_range_t range,  
    pmix_info_t info[],  
    size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata);
```

C

#### IN code

The `pmix_status_t` event code being referenced structure (handle)

#### IN source

`pmix_proc_t` of process that generated the event (handle)

#### IN range

`pmix_data_range_t` range over which the event is to be distributed (handle)

#### IN info

Optional array of `pmix_info_t` structures containing additional information on the event (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:

- **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.

- 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:

11 **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

14 Notify the specified processes (described through a combination of *range* and attributes provided in  
15 the *info* array) of an event generated either by the PMIx server itself or by one of its local clients.  
16 The process generating the event is provided in the *source* parameter, and any further descriptive  
17 information is included in the *info* array.  
18

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 17.3.17 `pmix_server_listener_fn_t`

### 26 Summary

27 Register a socket the host server can monitor for connection requests.

## Format

C

```
typedef pmix_status_t (*pmix_server_listener_fn_t) (  
    int listening_sd,  
    pmix_connection_cbfunc_t cbfunc,  
    void *cbdata);
```

C

**IN** `incoming_sd`

(integer)

**IN** `cbfunc`

Callback function `pmix_connection_cbfunc_t` (function reference)

**IN** `cbdata`

(memory reference)

Returns `PMIX_SUCCESS` indicating that the request is accepted, or a negative value corresponding to a PMIx error constant indicating that the request has been rejected.

## Description

Register a socket the host environment can monitor for connection requests, harvest them, and then call the PMIx server library's internal callback function for further processing. A listener thread is essential to efficiently harvesting connection requests from large numbers of local clients such as occur when running on large SMPs. The host server listener is required to call `accept` on the incoming connection request, and then pass the resulting socket to the provided `cbfunc`. A `NULL` for this function will cause the internal PMIx server to spawn its own listener thread.

### 17.3.17.1 PMIx Client Connection Callback Function

#### Summary

Callback function for incoming connection request from a local client.

#### Format

C

*PMIx v1.0*

```
typedef void (*pmix_connection_cbfunc_t) (  
    int incoming_sd, void *cbdata);
```

C

**IN** `incoming_sd`

(integer)

**IN** `cbdata`

(memory reference)

#### Description

Callback function for incoming connection requests from local clients - only used by host environments that wish to directly handle socket connection requests.

## 1 17.3.18 pmix\_server\_query\_fn\_t

### 2 Summary

3 Query information from the resource manager.

### 4 Format

PMIx v2.0

C

```
5 typedef pmix_status_t (*pmix_server_query_fn_t) (  
6     pmix_proc_t *proct,  
7     pmix_query_t *queries,  
8     size_t nqueries,  
9     pmix_info_cbfunc_t cbfunc,  
10    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\\_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

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:

**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.

## Optional Attributes

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

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

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

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

Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** 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\*)

Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace whose process table is being queried.

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

Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each process in the specified namespace executing on the same node as the requester, ordered by process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME** indicating the host whose local process table is being queried. By default, the query assumes that the host upon which the request was made is to be used.

**PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)

Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

**PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)

Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

1 **PMIX\_QUERY\_MEMORY\_USAGE** "pmix.qry.mem" (bool)  
2 Return information on memory usage for the processes indicated in the qualifiers.  
3 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** and **PMIX\_RANK**, or **PMIX\_PROCID** of  
4 specific process(es) whose memory usage is being requested.

5 **PMIX\_QUERY\_LOCAL\_ONLY** "pmix.qry.local" (bool)  
6 Constrain the query to local information only. NO QUALIFIERS.

7 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (bool)  
8 Report only average values for sampled information. NO QUALIFIERS.

9 **PMIX\_QUERY\_REPORT\_MINMAX** "pmix.qry.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\_NAMESPACE** of the namespace whose info is being  
16 requested (defaults to allocation containing the caller).

### Description

Query information from the host environment. The query will include the namespace/rank of the process that is requesting the info, an array of **pmix\_query\_t** describing the request, and a callback function/data for the return.

#### Advice to PMIx library implementers

The PMIx server library should not block in this function as the host environment may, depending upon the information being requested, require significant time to respond.

## 17.3.19 pmix\_server\_tool\_connection\_fn\_t

### Summary

Register that a tool has connected to the server.

## Format

C

```
typedef void (*pmix_server_tool_connection_fn_t) (  
    pmix_info_t info[], size_t ninfo,  
    pmix_tool_connection_cbfunc_t cbfunc,  
    void *cbdata);
```

C

### IN `info`

Array of `pmix_info_t` structures (array of handles)

### IN `ninfo`

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

### IN `cbfunc`

Callback function `pmix_tool_connection_cbfunc_t` (function reference)

### IN `cbdata`

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

## Required Attributes

PMIx libraries are required to pass the following attributes in the `info` array:

**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\_TOOL\_NAMESPACE** "pmix.tool.namespace" (`char*`)

Name of the namespace to use for this tool. This must be included only if the tool already has an assigned namespace.

**PMIX\_TOOL\_RANK** "pmix.tool.rank" (`uint32_t`)

Rank of this tool. This must be included only if the tool already has an assigned rank.

**PMIX\_CREDENTIAL** "pmix.cred" (`char*`)

Security credential assigned to the process.



## Optional Attributes

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

**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)

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.

**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\_VERSION\_INFO** "pmix.version" (char\*)

PMIx version of the library being used by the connecting process.

## Description

Register that a tool has connected to the server, possibly requesting that the tool be assigned a namespace/rank identifier for further interactions. The **pmix\_info\_t** array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

If the tool already has an assigned process identifier, then this must be indicated in the *info* array. The host is responsible for checking that the provided namespace does not conflict with any currently known assignments, returning an appropriate error in the callback function if a conflict is found.

The host environment is solely responsible for authenticating and authorizing the connection using whatever means it deems appropriate. If certificates or other authentication information are required, then the tool must provide them. The conclusion of those operations shall be communicated back to the PMIx server library via the callback function.

Approval or rejection of the connection request shall be returned in the *status* parameter of the **pmix\_tool\_connection\_cbfunc\_t**. If the connection is refused, the PMIx server library must terminate the connection attempt. The host must not execute the callback function prior to returning from the API.

### 1 17.3.19.1 Tool connection attributes

2 Attributes associated with tool connections.

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\_VERSION\_INFO** "pmix.version" (char\*)

8 PMIx version of the library being used by the connecting process.

### 9 17.3.19.2 PMIx Tool Connection Callback Function

#### 10 Summary

11 Callback function for incoming tool connections.

#### 12 Format

*PMIx v2.0*

```
typedef void (*pmix_tool_connection_cbfunc_t) (  
    pmix_status_t status,  
    pmix_proc_t *proc, void *cbdata);
```

16 **IN status**

[pmix\\_status\\_t](#) value (handle)

18 **IN proc**

[pmix\\_proc\\_t](#) structure containing the identifier assigned to the tool (handle)

20 **IN cbdata**

Data to be passed (memory reference)

#### 22 Description

23 Callback function for incoming tool connections. The host environment shall provide a  
24 namespace/rank identifier for the connecting tool.

#### Advice to PMIx server hosts

25 It is assumed that **rank=0** will be the normal assignment, but allow for the future possibility of a  
26 parallel set of tools connecting, and thus each process requiring a unique rank.

### 27 17.3.20 pmix\_server\_log\_fn\_t

#### 28 Summary

29 Log data on behalf of a client.

## Format

```
typedef void (*pmix_server_log_fn_t) (  
    const pmix_proc_t *client,  
    const pmix_info_t data[], size_t ndata,  
    const pmix_info_t directives[], size_t ndirs,  
    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

- IN client**  
pmix\_proc\_t structure (handle)
- IN data**  
Array of info structures (array of handles)
- IN ndata**  
Number of elements in the *data* array (integer)
- IN directives**  
Array of info structures (array of handles)
- IN ndirs**  
Number of elements in the *directives* array (integer)
- IN cbfunc**  
Callback function pmix\_op\_cbfunc\_t (function reference)
- IN cbdata**  
Data to be passed to the callback function (memory reference)

### 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:

- 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.

---

Host environments that provide this module entry point are required to support the following attributes:

- PMIX\_LOG\_STDERR** "pmix.log.stderr" (char\*)  
Log string to *stderr*.
- PMIX\_LOG\_STDOUT** "pmix.log.stdout" (char\*)  
Log string to *stdout*.
- 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

14 Log data on behalf of a client. This function is not intended for output of computational results, but  
15 rather for reporting status and error messages. The host must not execute the callback function prior  
16 to returning from the API.  
17

## 18 17.3.21 pmix\_server\_alloc\_fn\_t

### 19 Summary

20 Request allocation operations on behalf of a client.

## Format

```
typedef pmix_status_t (*pmix_server_alloc_fn_t) (  
    const pmix_proc_t *client,  
    pmix_alloc_directive_t directive,  
    const pmix_info_t data[],  
    size_t ndata,  
    pmix_info_cbfunc_t cbfunc,  
    void *cbdata);
```

### IN client

`pmix_proc_t` structure of process making request (handle)

### IN directive

Specific action being requested (`pmix_alloc_directive_t`)

### IN data

Array of info structures (array of handles)

### IN ndata

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

### IN cbfunc

Callback function `pmix_info_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 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

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:

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user ID of the connecting process.

1 **PMIX\_GRPID** "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 17.3.22 pmix\_server\_job\_control\_fn\_t

### 36 Summary

37 Execute a job control action on behalf of a client.

## Format

C

```
typedef pmix_status_t (*pmix_server_job_control_fn_t) (  
    const pmix_proc_t *requestor,  
    const pmix_proc_t targets[],  
    size_t ntargets,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    pmix_info_cbfunc_t cbfunc,  
    void *cbdata);
```

C

- IN requestor**  
`pmix_proc_t` structure of requesting process (handle)
- IN targets**  
Array of proc structures (array of handles)
- IN ntargets**  
Number of elements in the *targets* array (integer)
- IN directives**  
Array of info structures (array of handles)
- IN ndirs**  
Number of elements in the *info* array (integer)
- IN cbfunc**  
Callback function `pmix_info_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 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 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.pvning" (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\_cbfnc\_t** array of  
24 **pmix\_info\_t** structures.

## 25 17.3.23 pmix\_server\_monitor\_fn\_t

### 26 Summary

27 Request that a client be monitored for activity.

## Format

C

```
typedef pmix_status_t (*pmix_server_monitor_fn_t) (  
    const pmix_proc_t *requestor,  
    const pmix_info_t *monitor,  
    pmix_status_t error,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    pmix_info_cbfunc_t cbfunc,  
    void *cbdata);
```

C

**IN requestor**

[pmix\\_proc\\_t](#) structure of requesting process (handle)

**IN monitor**

[pmix\\_info\\_t](#) identifying the type of monitor being requested (handle)

**IN error**

Status code to use in generating event if alarm triggers (integer)

**IN directives**

Array of info structures (array of handles)

**IN ndirs**

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

**IN cbfunc**

Callback function [pmix\\_info\\_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 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

This entry point is only called for monitoring requests that are not directly supported by the PMIx server library itself.

## Required Attributes

1 If supported by the PMix server library, then the library must not pass any supported attributes to  
2 the host environment. Any attributes provided by the client that are not directly supported by the  
3 server library must be passed to the host environment if it provides this module entry. In addition,  
4 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\_GRPID** "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.bdrops" (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 17.3.24 pmix\_server\_get\_cred\_fn\_t

### 8 Summary

9 Request a credential from the host environment.

### 10 Format

*PMIx v3.0*

```
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);
```

### 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:

**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.

## Optional Attributes

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

**PMIX\_CRED\_TYPE** "pmix.sec.ctype" (char\*)

When passed in **PMIx\_Get\_credential**, a prioritized, comma-delimited list of desired credential types for use in environments where multiple authentication mechanisms may be available. When returned in a callback function, a string identifier of the credential type.

**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 host environment.

## 17.3.24.1 Credential callback function

### Summary

Callback function to return a requested security credential

## Format

C

```
typedef void (*pmix_credential_cbfunc_t) (  
    pmix_status_t status,  
    pmix_byte_object_t *credential,  
    pmix_info_t info[], size_t ninfo,  
    void *cbdata);
```

C

### IN status

[pmix\\_status\\_t](#) value (handle)

### IN credential

[pmix\\_byte\\_object\\_t](#) structure containing the security credential (handle)

### IN info

Array of provided by the system to pass any additional information about the credential - e.g., the identity of the issuing agent. (handle)

### IN ninfo

Number of elements in *info* ([size\\_t](#))

### IN cbdata

Object passed in original request (memory reference)

## Description

Define a callback function to return a requested security credential. Information provided by the issuing agent can subsequently be used by the application for a variety of purposes. Examples include:

- checking identified authorizations to determine what requests/operations are feasible as a means to steering [workflows](#)
- compare the credential type to that of the local SMS for compatibility

### Advice to users

The credential is opaque and therefore understandable only by a service compatible with the issuer. The *info* array is owned by the PMIx library and is not to be released or altered by the receiving party.

## 17.3.25 pmix\_server\_validate\_cred\_fn\_t

### Summary

Request validation of a credential.

## Format

```
typedef pmix_status_t (*pmix_server_validate_cred_fn_t) (  
    const pmix_proc_t *proc,  
    const pmix_byte_object_t *cred,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    pmix_validation_cbfunc_t cbfunc,  
    void *cbdata);
```

### IN `proc`

`pmix_proc_t` structure of requesting process (handle)

### IN `cred`

Pointer to `pmix_byte_object_t` containing the credential (handle)

### IN `directives`

Array of info structures (array of handles)

### IN `ndirs`

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

### IN `cbfunc`

Callback function to return the result (`pmix_validation_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*
- **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

If the PMIx library does not itself validate the 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:

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

Effective user ID of the connecting process.



1 **PMIX\_GRPID** "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 **17.3.26 Credential validation callback function**

### 14 **Summary**

15 Callback function for security credential validation.

## Format

C

```
typedef void (*pmix_validation_cfunc_t) (  
    pmix_status_t status,  
    pmix_info_t info[], size_t ninfo,  
    void *cbdata);
```

C

### IN status

[pmix\\_status\\_t](#) value (handle)

### IN info

Array of [pmix\\_info\\_t](#) provided by the system to pass any additional information about the authentication - e.g., the effective userid and group id of the certificate holder, and any related authorizations (handle)

### IN ninfo

Number of elements in *info* ([size\\_t](#))

### IN cbdata

Object passed in original request (memory reference)

The returned status shall be one of the following:

- [PMIX\\_SUCCESS](#), indicating that the request was processed and returned *success* (i.e., the credential was both valid and any information it contained was successfully processed). Details of the result will be returned in the *info* array
- a PMIx error constant indicating either an error in the parsing of the credential or that the request was refused

## Description

Define a validation callback function to indicate if a provided credential is valid, and any corresponding information regarding authorizations and other security matters.

### Advice to users

The precise contents of the array will depend on the host environment and its associated security system. At the minimum, it is expected (but not required) that the array will contain entries for the [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.

## 17.3.27 pmix\_server\_iof\_fn\_t

### Summary

Request the specified IO channels be forwarded from the given array of processes.

## Format

C

```
typedef pmix_status_t (*pmix_server_iof_fn_t) (  
    const pmix_proc_t procs[],  
    size_t nprocs,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    pmix_iof_channel_t channels,  
    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

- IN** **procs**  
Array `pmix_proc_t` identifiers whose IO is being requested (handle)
- IN** **nprocs**  
Number of elements in *procs* (`size_t`)
- IN** **directives**  
Array of `pmix_info_t` structures further defining the request (array of handles)
- IN** **ndirs**  
Number of elements in the *info* array (integer)
- IN** **channels**  
Bitmask identifying the channels to be forwarded (`pmix_iof_channel_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
- **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

The following attributes are required to be included in the passed *info* array:

**PMIX\_USERID** "pmix.euid" (`uint32_t`)

1           Effective user ID of the connecting process.

2     **PMIX\_GRPID** "pmix.egid" (uint32\_t)

3           Effective group ID of the connecting process.

4

5     Host environments that provide this module entry point are required to support the following  
6     attributes:

7     **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csize" (uint32\_t)

8           The requested size of the PMIx server cache in bytes for each specified channel. By default,  
9           the server is allowed (but not required) to drop all bytes received beyond the max size.

10    **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)

11           In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
12           cache.

13    **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)

14           In an overflow situation, the PMIx server is to drop any new bytes received until room  
15           becomes available in the cache (default).



▼----- Optional Attributes -----▼

16    The following attributes may be supported by a host environment.

17    **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)

18           Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
19           specified number of bytes is collected to avoid being called every time a block of IO arrives.  
20           The PMIx tool library will execute the callback and reset the collection counter whenever the  
21           specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
22           the callback upon a call to deregister the respective channel.

23    **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)

24           Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
25           size, this prevents IO from being held indefinitely while waiting for another payload to  
26           arrive.



## Description

Request the specified IO channels be forwarded from the given array of processes. An error shall be returned in the callback function if the requested service from any of the requested processes cannot be provided.

### Advice to PMIx library implementers

The forwarding of stdin is a *push* process - processes cannot request that it be *pulled* from some other source. Requests including the `PMIX_FWD_STDIN_CHANNEL` channel will return a `PMIX_ERR_NOT_SUPPORTED` error.

## 17.3.27.1 IOF delivery function

### Summary

Callback function for delivering forwarded IO to a process.

### Format

PMIx v3.0

C

```
typedef void (*pmix_iof_cfunc_t) (  
    size_t iofhdlr, pmix_iof_channel_t channel,  
    pmix_proc_t *source, char *payload,  
    pmix_info_t info[], size_t ninfo);
```

C

#### IN iofhdlr

Registration number of the handler being invoked (`size_t`)

#### IN channel

bitmask identifying the channel the data arrived on (`pmix_iof_channel_t`)

#### IN source

Pointer to a `pmix_proc_t` identifying the namespace/rank of the process that generated the data (`char*`)

#### IN payload

Pointer to character array containing the data.

#### IN info

Array of `pmix_info_t` provided by the source containing metadata about the payload. This could include `PMIX_IOF_COMPLETE` (handle)

#### IN ninfo

Number of elements in *info* (`size_t`)

## Description

Define a callback function for delivering forwarded IO to a process. This function will be called whenever data becomes available, or a specified buffering size and/or time has been met.

### Advice to users

Multiple strings may be included in a given *payload*, and the *payload* may *not* be **NULL** terminated. 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.

## 17.3.28 pmix\_server\_stdin\_fn\_t

### Summary

Pass standard input data to the host environment for transmission to specified recipients.

### Format

PMIx v3.0

C

```
typedef pmix_status_t (*pmix_server_stdin_fn_t) (  
    const pmix_proc_t *source,  
    const pmix_proc_t targets[],  
    size_t ntargets,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    const pmix_byte_object_t *bo,  
    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

- IN source**  
[pmix\\_proc\\_t](#) structure of source process (handle)
- IN targets**  
Array of [pmix\\_proc\\_t](#) target identifiers (handle)
- IN ntargets**  
Number of elements in the *targets* array (integer)
- IN directives**  
Array of info structures (array of handles)
- IN ndirs**  
Number of elements in the *info* array (integer)
- IN bo**  
Pointer to [pmix\\_byte\\_object\\_t](#) containing the payload (handle)
- IN cbfunc**  
Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)
- IN cbdata**  
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.  
▲-----▲

17 **Description**

18 Passes stdin to the host environment for transmission to specified recipients. The host environment  
19 is responsible for forwarding the data to all locations that host the specified *targets* and delivering  
20 the payload to the PMIx server library connected to those clients.

21 **17.3.29 pmix\_server\_grp\_fn\_t**

22 **Summary**

23 Request group operations (construct, destruct, etc.) on behalf of a set of processes.

## Format

C

```
typedef pmix_status_t (*pmix_server_grp_fn_t) (  
    pmix_group_operation_t op,  
    char grp[],  
    const pmix_proc_t procs[],  
    size_t nprocs,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    pmix_info_cbfunc_t cbfunc,  
    void *cbdata);
```

C

- IN op**  
[pmix\\_group\\_operation\\_t](#) value indicating operation the host is requested to perform (integer)
- IN grp**  
Character string identifying the group (string)
- IN procs**  
Array of [pmix\\_proc\\_t](#) identifiers of participants (handle)
- IN nprocs**  
Number of elements in the *procs* array (integer)
- IN directives**  
Array of info structures (array of handles)
- IN ndirs**  
Number of elements in the *info* array (integer)
- IN cbfunc**  
Callback function [pmix\\_info\\_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
- **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



## Optional Attributes

The following attributes may be supported by a host environment.

**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\_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\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

Data collected during group construction to ensure communication between group members is supported upon completion of the 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\_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.

The following attributes may be included in the host's response:

**PMIX\_GROUP\_ID** "pmix.grp.id" (char\*)

User-provided group identifier - as the group identifier may be used in PMIx operations, the user is required to ensure that the provided ID is unique within the scope of the host environment (e.g., by including some user-specific or application-specific prefix or suffix to the string).

**PMIX\_GROUP\_MEMBERSHIP** "pmix.grp.mbrs" (pmix\_data\_array\_t\*)

Array **pmix\_proc\_t** identifiers identifying the members of the specified group.

**PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (size\_t)

Context identifier assigned to the group by the host RM.

**PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

Data collected during group construction to ensure communication between group members is supported upon completion of the operation.

## Description

Perform the specified operation across the identified processes, plus any special actions included in the directives. Return the result of any special action requests in the callback function when the operation is completed. Actions may include a request (`PMIX_GROUP_ASSIGN_CONTEXT_ID`) that the host assign a unique numerical (size\_t) ID to this group - if given, the `PMIX_RANGE` attribute will specify the range across which the ID must be unique (default to `PMIX_RANGE_SESSION`).

### 17.3.29.1 Group Operation Constants

*PMIx v4.0*

The `pmix_group_operation_t` structure is a `uint8_t` value for specifying group operations. All values were originally defined in version 4 of the standard unless otherwise marked.

**PMIX\_GROUP\_CONSTRUCT** Construct a group composed of the specified processes - used by a PMIx server library to direct host operation.

**PMIX\_GROUP\_DESTRUCT** Destruct the specified group - used by a PMIx server library to direct host operation.

### 17.3.30 pmix\_server\_fabric\_fn\_t

#### Summary

Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

#### Format

*PMIx v4.0*

```
typedef pmix_status_t (*pmix_server_fabric_fn_t) (  
    const pmix_proc_t *requestor,  
    pmix_fabric_operation_t op,  
    const pmix_info_t directives[],  
    size_t ndirs,  
    pmix_info_cbfunc_t cbfunc,  
    void *cbdata);
```

**IN requestor**

`pmix_proc_t` identifying the requestor (handle)

**IN op**

`pmix_fabric_operation_t` value indicating operation the host is requested to perform (integer)

**IN directives**

Array of info structures (array of handles)

**IN ndirs**

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

1 **IN** `cbfunc`  
2 Callback function `pmix_info_cbfunc_t` (function reference)  
3 **IN** `cbdata`  
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.
- 9 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
10 returned `success` - the `cbfunc` will not be called
- 11 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
12 request, even though the function entry was provided in the server module - the `cbfunc` will not  
13 be called
- 14 • a PMIx error constant indicating either an error in the input or that the request was immediately  
15 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.  
34

## CHAPTER 18

# Tools and Debuggers

---

1 The term *tool* widely refers to programs executed by the user or system administrator on a  
2 command line. Tools frequently interact with either the SMS, user applications, or both to perform  
3 administrative and support functions. For example, a debugger tool might be used to remotely  
4 control the processes of a parallel application, monitoring their behavior on a step-by-step basis.  
5 Historically, such tools were custom-written for each specific host environment due to the  
6 customized and/or proprietary nature of the environment's interfaces.

7 The advent of PMIx offers the possibility for creating portable tools capable of interacting with  
8 multiple RMs without modification. Possible use-cases include:

- 9 • querying the status of scheduling queues and estimated allocation time for various resource  
10 options
- 11 • job submission and allocation requests
- 12 • querying job status for executing applications
- 13 • launching, monitoring, and debugging applications

14 Enabling these capabilities requires some extensions to the PMIx Standard (both in terms of APIs  
15 and attributes), and utilization of client-side APIs for more tool-oriented purposes.

16 This chapter defines specific APIs related to tools, provides tool developers with an overview of the  
17 support provided by PMIx, and serves to guide RM vendors regarding roles and responsibilities of  
18 RMs to support tools. As the number of tool-specific APIs and attributes is fairly small, the bulk of  
19 the chapter serves to provide a "theory of operation" for tools and debuggers. Description of the  
20 APIs themselves is therefore deferred to the Section 18.5 later in the chapter.

## 21 18.1 Connection Mechanisms

22 The key to supporting tools lies in providing mechanisms by which a tool can connect to a PMIx  
23 server. Application processes are able to connect because their local RM daemon provides them  
24 with the necessary contact information upon execution. A command-line tool, however, isn't  
25 spawned by an RM daemon, and therefore lacks the information required for rendezvous with a  
26 PMIx server.

27 Once a tool has started, it initializes PMIx as a tool (via `PMIx_tool_init`) if its access is  
28 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  
12 required connection information, or are able to discover it via one of several defined rendezvous  
13 protocols. Connection discovery centers around the existence of *rendezvous files* containing the  
14 necessary connection information, as illustrated in Fig. 18.1.

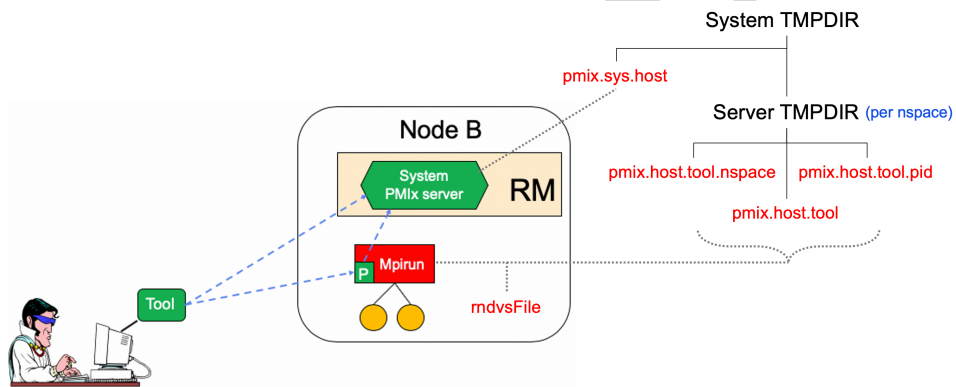


Figure 18.1.: Tool rendezvous files

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:

- 1 • *pmix.host.tool.namespace* where *host* is the string returned by the `gethostname` system call and  
2 *namespace* is the namespace of the server.
- 3 • *pmix.host.tool.pid* where *host* is the string returned by the `gethostname` system call and *pid* is  
4 the PID of the server.
- 5 • *pmix.host.tool* where *host* is the string returned by the `gethostname` system call. Note that  
6 servers which are not given a namespace-specific `PMIX_SERVER_TMPDIR` attribute may not  
7 generate this file due to conflicts should multiple servers be present on the node.

8 The files are identical and may be implemented as symlinks to a single instance. The individual file  
9 names are composed so as to aid the search process should a tool wish to connect to a server  
10 identified by its namespace or PID.

11 Servers will additionally provide a rendezvous file in any given location if the path (either absolute  
12 or relative) and filename is specified either during `PMIx_server_init` using the  
13 `PMIX_LAUNCHER_RENDEZVOUS_FILE` attribute, or by the `PMIX_LAUNCHER_RNDZ_FILE`  
14 environmental variable prior to executing the process containing the server. This latter mechanism  
15 may be the preferred mechanism for tools such as debuggers that need to fork/exec a launcher (e.g.,  
16 "mpixec") and then rendezvous with it. This is described in more detail in Section 18.2.2.

17 Rendezvous file ownerships are set to the UID and GID of the server that created them, with  
18 permissions set according to the desires of the implementation and/or system administrator policy.  
19 All connection attempts are first governed by read access privileges to the target rendezvous file -  
20 thus, the combination of permissions, UID, and GID of the rendezvous files act as a first-level of  
21 security for tool access.

22 A tool may connect to as many servers at one time as the implementation supports, but is limited to  
23 designating only one such connection as its *primary* server. This is done to avoid confusion when  
24 the tool calls an API as to which server should service the request. The first server the tool connects  
25 to is automatically designated as the *primary* server.

26 Tools are allowed to change their primary server at any time via the `PMIx_tool_set_server`  
27 API, and to connect/disconnect from a server as many times as desired. Note that standing requests  
28 (e.g., event registrations) with the current primary server may be lost and/or may not be transferred  
29 when transitioning to another primary server - PMIx implementors are not required to maintain or  
30 transfer state across tool-server connections.

31 Tool process identifiers are assigned by one of the following methods:

- 32 • If `PMIX_TOOL_NAMESPACE` is given, then the namespace of the tool will be assigned that value.  
33 – If `PMIX_TOOL_RANK` is also given, then the rank of the tool will be assigned that value.  
34 – If `PMIX_TOOL_RANK` is not given, then the rank will be set to a default value of zero.
- 35 • If a process ID is not provided and the tool connects to a server, then one will be assigned by the  
36 host environment upon connection to that server.

- If a process ID is not provided and the tool does not connect to a server (e.g., if `PMIX_TOOL_DO_NOT_CONNECT` is given), then the tool shall self-assign a unique identifier. This is often done using some combination involving hostname and PID.

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_attach_to_server`.

## 18.1.1 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_attach_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.

Note that the PMIx implementation may choose to introduce a "delayed connection" protocol between steps in the precedence chain - i.e., the library may cycle several times, checking for creation of the rendezvous file each time after a delay of some period of time, thereby allowing the tool to wait for the server to create the rendezvous file before either returning an error or continuing to the next step in the chain.

- 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. An error will be returned if the specified file cannot be found.
- 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.



- 1 • If `PMIX_SERVER_PIDINFO` was provided, then the tool will search for a rendezvous file  
2 created by a PMIx server of the given PID in the `PMIX_SERVER_TMPDIR` directory. An error  
3 will be returned if a matching rendezvous file cannot be found.
- 4 • If `PMIX_SERVER_NAMESPACE` is given, then the tool will search for a rendezvous file created by a  
5 PMIx server of the given namespace in the `PMIX_SERVER_TMPDIR` directory. An error will  
6 be returned if a matching rendezvous file cannot be found.
- 7 • If `PMIX_CONNECT_TO_SYSTEM` is given, then the tool will search for a system-level  
8 rendezvous file created by a PMIx server in the `PMIX_SYSTEM_TMPDIR` directory. An error  
9 will be returned if a matching rendezvous file cannot be found.
- 10 • If `PMIX_CONNECT_SYSTEM_FIRST` is given, then the tool will look for a system-level  
11 rendezvous file created by a PMIx server in the `PMIX_SYSTEM_TMPDIR` directory. If found,  
12 then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous  
13 file is not found or connection is refused — the PMIx library will silently continue to the next  
14 option.
- 15 • By default, the tool will search the directory tree under the `PMIX_SERVER_TMPDIR` directory  
16 for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the  
17 connection. If no rendezvous files are found, or all contacted servers refuse connection, then the  
18 PMIx library will return an error. No "delayed connection" protocols may be utilized at this point.

19 Note that there can be multiple local servers - one from the system plus others from launchers and  
20 active jobs. The PMIx tool connection search method is not guaranteed to pick a particular server  
21 unless directed to do so. Tools can obtain a list of servers available on their local node using the  
22 `PMIx_Query_info` APIs with the `PMIX_QUERY_AVAIL_SERVERS` key.

## 23 18.1.2 Connecting to a remote server

24 Connecting to remote servers is complicated due to the lack of access to the previously-described  
25 rendezvous files. Two methods are required to be supported, both based on the caller having explicit  
26 knowledge of either connection information or a path to a local file that contains such information:

- 27 • If `PMIX_TOOL_ATTACHMENT_FILE` is given, then the tool will attempt to read the specified  
28 file and connect to the server based on the information contained within it. The format of the  
29 attachment file is identical to the rendezvous files described in earlier in this section.
- 30 • If `PMIX_SERVER_URI` or `PMIX_TCP_URI` is given, then connection will be attempted to the  
31 server at the specified URI. Note that it is an error for both of these attributes to be specified.  
32 `PMIX_SERVER_URI` is the preferred method as it is more generalized — `PMIX_TCP_URI` is  
33 provided for those cases where the user specifically wants to use the TCP transport for the  
34 connection and wants to error out if it isn't available or cannot be used.

35 Additional methods may be provided by particular PMIx implementations. For example, the tool  
36 may use `ssh` to launch a `probe` process onto the remote node so that the probe can search the  
37 `PMIX_SYSTEM_TMPDIR` and `PMIX_SERVER_TMPDIR` directories for rendezvous files,



1 relaying the discovered information back to the requesting tool. If sufficient information is found to  
2 allow for remote connection, then the tool can use it to establish the connection. Note that this  
3 method is not required to be supported - it is provided here as an example and left to the discretion  
4 of PMIx implementors.

### 5 **18.1.3 Attaching to running jobs**

6 When attaching to a running job, the tool must connect to a PMIx server that is associated with that  
7 job - e.g., a server residing in the host environment's local daemon that spawned one or more of the  
8 job's processes, or the server residing in the launcher that is overseeing the job. Identifying an  
9 appropriate server can sometimes prove challenging, particularly in an environment where multiple  
10 job launchers may be in operation, possibly under control of the same user.

11 In cases where the user has only the one job of interest in operation on the local node (e.g., when  
12 engaged in an interactive session on the node from which the launcher was executed), the normal  
13 rendezvous file discovery method can often be used to successfully connect to the target job, even  
14 in the presence of jobs executed by other users. The permissions and security authorizations can, in  
15 many cases, reliably ensure that only the one connection can be made. However, this is not  
16 guaranteed in all cases.

17 The most common method, therefore, for attaching to a running job is to specify either the PID of  
18 the job's launcher or the namespace of the launcher's job (note that the launcher's namespace  
19 frequently differs from the namespace of the job it has launched). Unless the application processes  
20 themselves act as PMIx servers, connection must be to the servers in the daemons that oversee the  
21 application. This is typically either daemons specifically started by the job's launcher process, or  
22 daemons belonging to the host environment, that are responsible for starting the application's  
23 processes and oversee their execution.

24 Identifying the correct PID or namespace can be accomplished in a variety of ways, including:

- 25 • Using typical OS or host environment tools to obtain a listing of active jobs and perusing those to  
26 find the target launcher.
- 27 • Using a PMIx-based tool attached to a system-level server to query the active jobs and their  
28 command lines, thereby identifying the application of interest and its associated launcher.
- 29 • Manually recording the PID of the launcher upon starting the job.

30 Once the namespace and/or PID of the target server has been identified, either of the previous  
31 methods can be used to connect to it.

### 32 **18.1.4 Tool initialization attributes**

33 The following attributes are passed to the `PMIx_tool_init` API for use when initializing the  
34 PMIx library.

35 `PMIX_TOOL_NAMESPACE` "pmix.tool.namespace" (char\*)

1 Name of the namespace to use for this tool.  
2 **PMIX\_TOOL\_RANK** "pmix.tool.rank" (uint32\_t)  
3 Rank of this tool.  
4 **PMIX\_LAUNCHER** "pmix.tool.launcher" (bool)  
5 Tool is a launcher and needs to create rendezvous files.

## 6 18.1.5 Tool initialization environmental variables

7 The following environmental variables are used during **PMIx\_tool\_init** and  
8 **PMIx\_server\_init** to control various rendezvous-related operations when the process is  
9 started manually (e.g., on a command line) or by a fork/exec-like operation.

### 10 **PMIX\_LAUNCHER\_RNDZ\_URI**

11 The spawned tool is to be connected back to the spawning tool using the given URI so that  
12 the spawning tool can provide directives (e.g., a **PMIx\_Spawn** command) to it.

### 13 **PMIX\_LAUNCHER\_RNDZ\_FILE**

14 If the specified file does not exist, this variable contains the absolute path of the file where  
15 the spawned tool is to store its connection information so that the spawning tool can connect  
16 to it. If the file does exist, it contains the information specifying the server to which the  
17 spawned tool is to connect.

### 18 **PMIX\_KEEPA\_LIVE\_PIPE**

19 An integer **read**-end of a POSIX pipe that the tool should monitor for closure, thereby  
20 indicating that the parent tool has terminated. Used, for example, when a tool fork/exec's an  
21 intermediate launcher that should self-terminate if the originating tool exits.

22 Note that these environmental variables should be cleared from the environment after use and prior  
23 to forking child processes to avoid potentially unexpected behavior by the child processes.

## 24 18.1.6 Tool connection attributes

25 These attributes are defined to assist PMIx-enabled tools to connect with a PMIx server by passing  
26 them into either the **PMIx\_tool\_init** or the **PMIx\_tool\_attach\_to\_server** APIs - thus,  
27 they are not typically accessed via the **PMIx\_Get** API.

28 **PMIX\_SERVER\_PIDINFO** "pmix.srvr.pidinfo" (pid\_t)

29 PID of the target PMIx server for a tool.

30 **PMIX\_CONNECT\_TO\_SYSTEM** "pmix.cnct.sys" (bool)

31 The requester requires that a connection be made only to a local, system-level PMIx server.

32 **PMIX\_CONNECT\_SYSTEM\_FIRST** "pmix.cnct.sys.first" (bool)

33 Preferentially, look for a system-level PMIx server first.

34 **PMIX\_SERVER\_URI** "pmix.srvr.uri" (char\*)

35 URI of the PMIx server to be contacted.

36 **PMIX\_SERVER\_HOSTNAME** "pmix.srvr.host" (char\*)

37 Host where target PMIx server is located.

38 **PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)

1 Maximum number of times to try to connect to PMIx server - the default value is  
2 implementation specific.

3 **PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (uint32\_t)

4 Time in seconds between connection attempts to a PMIx server - the default value is  
5 implementation specific.

6 **PMIX\_TOOL\_DO\_NOT\_CONNECT** "pmix.tool.nocon" (bool)

7 The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

8 **PMIX\_TOOL\_CONNECT\_OPTIONAL** "pmix.tool.conopt" (bool)

9 The tool shall connect to a server if available, but otherwise continue to operate unconnected.

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 **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** "pmix.tool.lncrnd" (char\*)

14 Pathname of file where the launcher is to store its connection information so that the  
15 spawning tool can connect to it.

16 **PMIX\_PRIMARY\_SERVER** "pmix.pri.srvr" (bool)

17 The server to which the tool is connecting shall be designated the *primary* server once  
18 connection has been accomplished.

19 **PMIX\_WAIT\_FOR\_CONNECTION** "pmix.wait.conn" (bool)

20 Wait until the specified process has connected to the requesting tool or server, or the  
21 operation times out (if the **PMIX\_TIMEOUT** directive is included in the request).

## 22 18.2 Launching Applications with Tools

23 Tool-directed launches require that the tool include the **PMIX\_LAUNCHER** attribute when calling  
24 **PMIx\_tool\_init**. Two launch modes are supported:

- 25 • *Direct launch* where the tool itself is directly responsible for launching all processes, including  
26 debugger daemons, using either the RM or daemons launched by the tool – i.e., there is no  
27 *intermediate launcher* (IL) such as *mpiexec*. The case where the tool is self-contained (i.e., uses  
28 its own daemons without interacting with an external entity such as the RM) lies outside the  
29 scope of this Standard; and
- 30 • *Indirect launch* where all processes are started via an IL such as *mpiexec* and the tool itself is not  
31 directly involved in launching application processes or debugger daemons. Note that the IL may  
32 utilize the RM to launch processes and/or daemons under the tool's direction.

33 Either of these methods can be executed interactively or by a batch script. Note that not all host  
34 environments may support the direct launch method.

### 35 18.2.1 Direct launch

36 In the direct-launch use-case (Fig. 18.2), the tool itself performs the role of the launcher. Once  
37 invoked, the tool connects to an appropriate PMIx server - e.g., a system-level server hosted by the

1 RM. The tool is responsible for assembling the description of the application to be launched (e.g.,  
 2 by parsing its command line) into a spawn request containing an array of `pmix_app_t`  
 3 applications and `pmix_info_t` job-level information. An allocation of resources may or may not  
 4 have been made in advance – if not, then the spawn request must include allocation request  
 5 information.

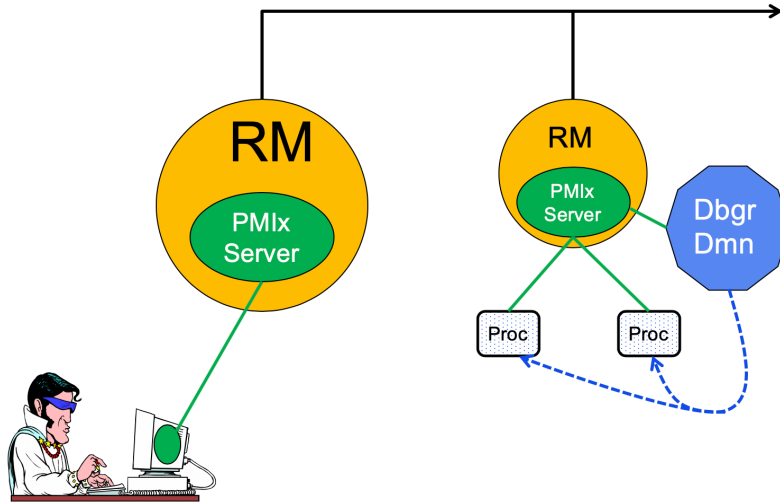


Figure 18.2.: Direct Launch

6 In addition to the attributes described in `PMIX_Spawn`, the tool may optionally wish to include the  
 7 following tool-specific attributes in the `job_info` argument to that API (the debugger-related  
 8 attributes are discussed in more detail in Section 18.4):

- 9 • `PMIX_FWD_STDIN` "`pmix.fwd.stdin`" (`pmix_rank_t`)  
 10 The requester intends to push information from its `stdin` to the indicated process. The  
 11 local spawn agent should, therefore, ensure that the `stdin` channel to that process  
 12 remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the  
 13 spawned job are potential recipients. The requester will issue a call to `PMIX_IOF_push`  
 14 to initiate the actual forwarding of information to specified targets - this attribute simply  
 15 requests that the IL retain the ability to forward the information to the designated targets.
- 16 • `PMIX_FWD_STDOUT` "`pmix.fwd.stdout`" (`bool`)  
 17 Requests that the ability to forward the `stdout` 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_FWD_STDERR` "`pmix.fwd.stderr`" (`bool`)  
 21 Requests that the ability to forward the `stderr` of the spawned processes be maintained.  
 22 The requester will issue a call to `PMIX_IOF_pull` to specify the callback function and

1 other options for delivery of the forwarded output.

- 2 ● **PMIX\_FWD\_STDDIAG** "pmix.fwd.stddiag" (bool)  
3 Requests that the ability to forward the diagnostic channel (if it exists) of the spawned  
4 processes be maintained. The requester will issue a call to **PMIx\_IOF\_pull** to specify  
5 the callback function and other options for delivery of the forwarded output.
- 6 ● **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csize" (uint32\_t)  
7 The requested size of the PMIx server cache in bytes for each specified channel. By  
8 default, the server is allowed (but not required) to drop all bytes received beyond the max  
9 size.
- 10 ● **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)  
11 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
12 cache.
- 13 ● **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)  
14 In an overflow situation, the PMIx server is to drop any new bytes received until room  
15 becomes available in the cache (default).
- 16 ● **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)  
17 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until  
18 the specified number of bytes is collected to avoid being called every time a block of IO  
19 arrives. The PMIx tool library will execute the callback and reset the collection counter  
20 whenever the specified number of bytes becomes available. Any remaining buffered data  
21 will be *flushed* to the callback upon a call to deregister the respective channel.
- 22 ● **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)  
23 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
24 size, this prevents IO from being held indefinitely while waiting for another payload to  
25 arrive.
- 26 ● **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (bool)  
27 Requests that output be prefixed with the nspace,rank of the source and a string  
28 identifying the channel (**stdout**, **stderr**, etc.).
- 29 ● **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (bool)  
30 Requests that output be marked with the time at which the data was received by the tool -  
31 note that this will differ from the time at which the data was collected from the source.
- 32 ● **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)  
33 Requests that output be formatted in XML.
- 34 ● **PMIX\_NOHUP** "pmix.nohup" (bool)  
35 Any processes started on behalf of the calling tool (or the specified namespace, if such  
36 specification is included in the list of attributes) should continue after the tool disconnects  
37 from its server.
- 38 ● **PMIX\_NOTIFY\_JOB\_EVENTS** "pmix.note.jev" (bool)

1 Requests that the launcher generate the [PMIX\\_EVENT\\_JOB\\_START](#),  
2 [PMIX\\_LAUNCH\\_COMPLETE](#), and [PMIX\\_EVENT\\_JOB\\_END](#) events. Each event is to  
3 include at least the namespace of the corresponding job and a  
4 [PMIX\\_EVENT\\_TIMESTAMP](#) indicating the time the event occurred. Note that the  
5 requester must register for these individual events, or capture and process them by  
6 registering a default event handler instead of individual handlers and then process the  
7 events based on the returned status code. Another common method is to register one event  
8 handler for all job-related events, with a separate handler for non-job events - see  
9 [PMIx\\_Register\\_event\\_handler](#) for details.

10 • [PMIX\\_NOTIFY\\_COMPLETION](#) "pmix.notecomp" (bool)

11 Requests that the launcher generate the [PMIX\\_EVENT\\_JOB\\_END](#) event for normal or  
12 abnormal termination of the spawned job. The event shall include the returned status code  
13 ([PMIX\\_JOB\\_TERM\\_STATUS](#)) for the corresponding job; the identity ([PMIX\\_PROCID](#))  
14 and exit status ([PMIX\\_EXIT\\_CODE](#)) of the first failed process, if applicable; and a  
15 [PMIX\\_EVENT\\_TIMESTAMP](#) indicating the time the termination occurred. Note that the  
16 requester must register for the event or capture and process it within a default event  
17 handler.

18 • [PMIX\\_LOG\\_JOB\\_EVENTS](#) "pmix.log.jev" (bool)

19 Requests that the launcher log the [PMIX\\_EVENT\\_JOB\\_START](#),  
20 [PMIX\\_LAUNCH\\_COMPLETE](#), and [PMIX\\_EVENT\\_JOB\\_END](#) events using [PMIx\\_Log](#),  
21 subject to the logging attributes of Section 13.4.3.

22 • [PMIX\\_LOG\\_COMPLETION](#) "pmix.logcomp" (bool)

23 Requests that the launcher log the [PMIX\\_EVENT\\_JOB\\_END](#) event for normal or  
24 abnormal termination of the spawned job using [PMIx\\_Log](#), subject to the logging  
25 attributes of Section 13.4.3. The event shall include the returned status code  
26 ([PMIX\\_JOB\\_TERM\\_STATUS](#)) for the corresponding job; the identity ([PMIX\\_PROCID](#))  
27 and exit status ([PMIX\\_EXIT\\_CODE](#)) of the first failed process, if applicable; and a  
28 [PMIX\\_EVENT\\_TIMESTAMP](#) indicating the time the termination occurred.

29 • [PMIX\\_DEBUG\\_STOP\\_ON\\_EXEC](#) "pmix.dbg.exec" (bool)

30 Included in either the [pmix\\_info\\_t](#) array in a [pmix\\_app\\_t](#) description (if the  
31 directive applies only to that application) or in the [job\\_info](#) array if it applies to all  
32 applications in the given spawn request. Indicates that the application is being spawned  
33 under a debugger, and that the local launch agent is to pause the resulting application  
34 processes on first instruction for debugger attach. The launcher (RM or IL) is to generate  
35 the [PMIX\\_LAUNCH\\_COMPLETE](#) event when all processes are stopped at the exec point.

36 • [PMIX\\_DEBUG\\_STOP\\_IN\\_INIT](#) "pmix.dbg.init" (bool)

37 Included in either the [pmix\\_info\\_t](#) array in a [pmix\\_app\\_t](#) description (if the  
38 directive applies only to that application) or in the [job\\_info](#) array if it applies to all  
39 applications in the given spawn request. Indicates that the specified application is being  
40 spawned under a debugger. The PMIx client library in each resulting application process  
41 shall notify its PMIx server that it is pausing and then pause during [PMIx\\_Init](#) of the

spawned processes until either released by debugger modification of an appropriate variable or 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_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 either released by debugger modification of an appropriate variable or 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.

The tool then calls the `PMIx_Spawn` API so that the PMIx library can communicate the spawn 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.

## 18.2.2 Indirect launch

In the indirect launch use-case, the application processes are started via an intermediate launcher (e.g., `mpiexec`) that is itself started by the tool (see Fig 18.3). Thus, at a high level, this is a



two-stage launch procedure to start the application: the tool (henceforth referred to as the *initiator*) 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 initiator parse and/or understand the command line of the IL. Instead, the indirect launch procedure supports either of two methods: one where the initiator assumes responsibility for parsing its command line to obtain the application as well as the IL and its options, and another where the initiator defers the command line parsing to the IL. Both of these methods are described in the following sections.

### 18.2.2.1 Initiator-based command line parsing

This method utilizes a first call to the `PMIx_Spawn` API to start the IL itself, and then uses a second call to `PMIx_Spawn` to request that the IL spawn the actual job. The burden of analyzing the initial command line to separately identify the IL's command line from the application itself falls upon the initiator. An example is provided below:

```
$ initiator --launcher "mpiexec --verbose" -n 3 ./app <appoptions>
```

The initiator spawns the IL using the same procedure for launching an application - it begins by assembling the description of the IL into a spawn request containing an array of `pmix_app_t` and `pmix_info_t` job-level information. Note that this step does not include any information regarding the application itself - only the launcher is included. In addition, the initiator must include the rendezvous URI in the environment so the IL knows how to connect back to it.

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.

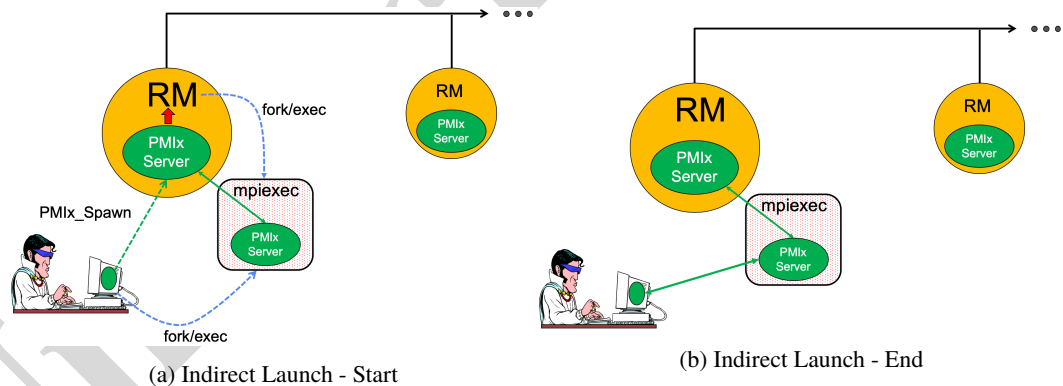


Figure 18.3.: Indirect launch procedure

The initiator may optionally wish to include the following tool-specific attributes in the `job_info` argument to `PMIx_Spawn` - note that these attributes refer only to the behavior of the IL itself and not the eventual job to be launched:

- `PMIX_FWD_STDIN` `"pmix.fwd.stdin"` (`pmix_rank_t`)



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`)  
8 Requests that the ability to forward the `stdout` of the spawned processes be maintained.  
9 The requester will issue a call to `PMIX_IOF_pull` to specify the callback function and  
10 other options for delivery of the forwarded output.
- 11 • `PMIX_FWD_STDERR` "`pmix.fwd.stderr`" (`bool`)  
12 Requests that the ability to forward the `stderr` of the spawned processes be maintained.  
13 The requester will issue a call to `PMIX_IOF_pull` to specify the callback function and  
14 other options for delivery of the forwarded output.
- 15 • `PMIX_FWD_STDDIAG` "`pmix.fwd.stddiag`" (`bool`)  
16 Requests that the ability to forward the diagnostic channel (if it exists) of the spawned  
17 processes be maintained. The requester will issue a call to `PMIX_IOF_pull` to specify  
18 the callback function and other options for delivery of the forwarded output.
- 19 • `PMIX_IOF_CACHE_SIZE` "`pmix.iof.csize`" (`uint32_t`)  
20 The requested size of the PMIx server cache in bytes for each specified channel. By  
21 default, the server is allowed (but not required) to drop all bytes received beyond the max  
22 size.
- 23 • `PMIX_IOF_DROP_OLDEST` "`pmix.iof.old`" (`bool`)  
24 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
25 cache.
- 26 • `PMIX_IOF_DROP_NEWEST` "`pmix.iof.new`" (`bool`)  
27 In an overflow situation, the PMIx server is to drop any new bytes received until room  
28 becomes available in the cache (default).
- 29 • `PMIX_IOF_BUFFERING_SIZE` "`pmix.iof.bsize`" (`uint32_t`)  
30 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until  
31 the specified number of bytes is collected to avoid being called every time a block of IO  
32 arrives. The PMIx tool library will execute the callback and reset the collection counter  
33 whenever the specified number of bytes becomes available. Any remaining buffered data  
34 will be *flushed* to the callback upon a call to deregister the respective channel.
- 35 • `PMIX_IOF_BUFFERING_TIME` "`pmix.iof.btime`" (`uint32_t`)  
36 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
37 size, this prevents IO from being held indefinitely while waiting for another payload to  
38 arrive.
- 39 • `PMIX_IOF_TAG_OUTPUT` "`pmix.iof.tag`" (`bool`)

1 Requests that output be prefixed with the nspace,rank of the source and a string  
2 identifying the channel (`stdout`, `stderr`, etc.).

- 3 • **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (bool)

4 Requests that output be marked with the time at which the data was received by the tool -  
5 note that this will differ from the time at which the data was collected from the source.

- 6 • **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)

7 Requests that output be formatted in XML.

- 8 • **PMIX\_NOHUP** "pmix.nohup" (bool)

9 Any processes started on behalf of the calling tool (or the specified namespace, if such  
10 specification is included in the list of attributes) should continue after the tool disconnects  
11 from its server.

- 12 • **PMIX\_LAUNCHER\_DAEMON** "pmix.lnch.dmn" (char\*)

13 Path to executable that is to be used as the backend daemon for the launcher. This replaces  
14 the launcher's own daemon with the specified executable. Note that the user is therefore  
15 responsible for ensuring compatibility of the specified executable and the host launcher.

- 16 • **PMIX\_FORKEXEC\_AGENT** "pmix.frkex.agnt" (char\*)

17 Path to executable that the launcher's backend daemons are to fork/exec in place of the  
18 actual application processes. The fork/exec agent shall connect back (as a PMIx tool) to  
19 the launcher's daemon to receive its spawn instructions, and is responsible for starting the  
20 actual application process it replaced. See Section 18.4.3 for details.

- 21 • **PMIX\_EXEC\_AGENT** "pmix.exec.agnt" (char\*)

22 Path to executable that the launcher's backend daemons are to fork/exec in place of the  
23 actual application processes. The launcher's daemon shall pass the full command line of  
24 the application on the command line of the exec agent, which shall not connect back to the  
25 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
26 process in its own place. See Section 18.4.3 for details.

- 27 • **PMIX\_DEBUG\_STOP\_IN\_INIT** "pmix.dbg.init" (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 PMIx client library in each resulting application process  
32 shall notify its PMIx server that it is pausing and then pause during `PMIx_Init` of the  
33 spawned processes until either released by debugger modification of an appropriate  
34 variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL)  
35 is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when  
36 all processes have reached the pause point. In this context, the initiator is directing the IL  
37 to stop in `PMIx_tool_init`. This gives the initiator a chance to connect to the IL and  
38 register for events prior to the IL launching the application job.

39 and the following optional variables in the environment of the IL:

- **PMIX\_KEEPALIVE\_PIPE** - an integer **read**-end of a POSIX pipe that the IL should monitor for closure, thereby indicating that the initiator has terminated.

The initiator then calls the **PMIx\_Spawn** API so that the PMIx library can either communicate the spawn request to a server (if connected to one), or locally spawn the IL itself if not connected to a server and the PMIx implementation includes self-spawn support. **PMIx\_Spawn** shall return an error if neither of these conditions is met.

When initialized by the IL, the **PMIx\_tool\_init** function must perform two operations:

- check for the presence of the **PMIX\_KEEPALIVE\_PIPE** environmental variable - if provided, then the library shall monitor the pipe for closure, providing a **PMIX\_EVENT\_JOB\_END** event when the pipe closes (thereby indicating the termination of the initiator). The IL should register for this event after completing **PMIx\_tool\_init** - the initiator's namespace can be obtained via a call to **PMIx\_Get** with the **PMIX\_PARENT\_ID** key. Note that this feature will only be available if the spawned IL is local to the initiator.
- check for the **PMIX\_LAUNCHER\_RNDZ\_URI** environmental parameter - if found, the library shall connect back to the initiator using the **PMIx\_tool\_attach\_to\_server** API, retaining its current server as its primary server.

Once the IL completes **PMIx\_tool\_init**, it must register for the **PMIX\_EVENT\_JOB\_END** termination event and then idle until receiving that event - either directly from the initiator, or from the PMIx library upon detecting closure of the keepalive pipe. The IL idles in the intervening time as it is solely acting as a relay (if connected to a server that is performing the actual application launch) or as a PMIx server responding to spawn requests.

Upon return from the **PMIx\_Spawn** API, the initiator should set the spawned IL as its primary server using the **PMIx\_tool\_set\_server** API with the namespace returned by **PMIx\_Spawn** and any valid rank (a rank of zero would ordinarily be used as only one IL process is typically started). It is advisable to set a connection timeout value when calling this function. The initiator can then proceed to spawn the actual application according to the procedure described in Section 18.2.1.

### 18.2.2.2 IL-based command line parsing

In the case where the initiator cannot parse its command line, it must defer that parsing to the IL. A common example is provided below:

```
$ initiator mpiexec --verbose -n 3 ./app <appoptions>
```

For this situation, the initiator proceeds as above with only one notable exception: instead of calling **PMIx\_Spawn** twice (once to start the IL and again to start the actual application), the initiator only calls that API one time:

- The *app* parameter passed to the spawn request contains only one **pmix\_app\_t** that contains the entire command line, including both launcher and application(s).
- The launcher executable must be in the *app.cmd* field and in *app.argv[0]*, with the rest of the command line appended to the *app.argv* array.

- Any job-level directives for the IL itself (e.g., `PMIX_FORKEXEC_AGENT` or `PMIX_FWD_STDOUT`) are included in the `job_info` parameter of the call to `PMIx_Spawn`.
- The job-level directives must include both the `PMIX_SPAWN_TOOL` attribute indicating that the initiator is spawning a tool, and the `PMIX_DEBUG_STOP_IN_INIT` attribute directing the IL to stop during the call to `PMIx_tool_init`. The latter directive allows the initiator to connect to the IL prior to launch of the application.
- The `PMIX_LAUNCHER_RNDZ_URI` and `PMIX_KEEPLIVE_PIPE` environmental variables are provided to the launcher in its environment via the `app.env` field.
- The IL must use `PMIx_Get` with the `PMIX_LAUNCH_DIRECTIVES` key to obtain any initiator-provided directives (e.g., `PMIX_DEBUG_STOP_IN_INIT` or `PMIX_DEBUG_STOP_ON_EXEC`) aimed at the application(s) it will spawn.

Upon return from `PMIx_Spawn`, the initiator must:

- use the `PMIx_tool_set_server` API to set the spawned IL as its primary server
- register with that server to receive the `PMIX_LAUNCH_COMPLETE` event. This allows the initiator to know when the IL has completed launch of the application
- release the IL from its "hold" in `PMIx_tool_init` by issuing the `PMIX_DEBUGGER_RELEASE` event, specifying the IL as the custom range. Upon receipt of the event, the IL is free to parse its command line, apply any provided directives, and execute the application.

Upon receipt of the `PMIX_LAUNCH_COMPLETE` event, the initiator should register to receive notification of completion of the returned namespace of the application. Receipt of the `PMIX_EVENT_JOB_END` event provides a signal that the initiator may itself terminate.

## 18.2.3 Tool spawn-related attributes

Tools are free to utilize the spawn attributes available to applications (see 12.2.4) when constructing a spawn request, but can also utilize the following attributes that are specific to tool-based spawn operations:

**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.

1 **PMIX\_FWD\_STDERR** "`pmix.fwd.stderr`" (**bool**)  
 2 Requests that the ability to forward the `stderr` of the spawned processes be maintained.  
 3 The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and  
 4 other options for delivery of the forwarded output.

5 **PMIX\_FWD\_STDDIAG** "`pmix.fwd.stddiag`" (**bool**)  
 6 Requests that the ability to forward the diagnostic channel (if it exists) of the spawned  
 7 processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify the  
 8 callback function and other options for delivery of the forwarded output.

9 **PMIX\_NOHUP** "`pmix.nohup`" (**bool**)  
 10 Any processes started on behalf of the calling tool (or the specified namespace, if such  
 11 specification is included in the list of attributes) should continue after the tool disconnects  
 12 from its server.

13 **PMIX\_LAUNCHER\_DAEMON** "`pmix.lnch.dmn`" (**char\***)  
 14 Path to executable that is to be used as the backend daemon for the launcher. This replaces  
 15 the launcher's own daemon with the specified executable. Note that the user is therefore  
 16 responsible for ensuring compatibility of the specified executable and the host launcher.

17 **PMIX\_FORKEXEC\_AGENT** "`pmix.frkex.agnt`" (**char\***)  
 18 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
 19 application processes. The fork/exec agent shall connect back (as a `PMIx` tool) to the  
 20 launcher's daemon to receive its spawn instructions, and is responsible for starting the actual  
 21 application process it replaced. See Section 18.4.3 for details.

22 **PMIX\_EXEC\_AGENT** "`pmix.exec.agnt`" (**char\***)  
 23 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
 24 application processes. The launcher's daemon shall pass the full command line of the  
 25 application on the command line of the exec agent, which shall not connect back to the  
 26 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
 27 process in its own place. See Section 18.4.3 for details.

28 **PMIX\_LAUNCH\_DIRECTIVES** "`pmix.lnch.dirs`" (`pmix_data_array_t*`)  
 29 Array of `pmix_info_t` containing directives for the launcher - a convenience attribute for  
 30 retrieving all directives with a single call to `PMIx_Get`.

## 31 18.2.4 Tool rendezvous-related events

32 The following constants refer to events relating to rendezvous of a tool and launcher during spawn  
 33 of the IL.

34 **PMIX\_LAUNCHER\_READY** An application launcher (e.g., `mpixec`) shall generate this event to  
 35 signal a tool that started it that the launcher is ready to receive directives/commands (e.g.,  
 36 `PMIx_Spawn`). This is only used when the initiator is able to parse the command line itself,  
 37 or the launcher is started as a persistent Distributed Virtual Machine (DVM).

## 38 18.3 IO Forwarding

39 Underlying the operation of many tools is a common need to forward `stdin` from the tool to  
 40 targeted processes, and to return `stdout/stderr` from those processes to the tool (e.g., for

1 display on the user’s console). Historically, each tool developer was responsible for creating their  
2 own IO forwarding subsystem. However, the introduction of PMIx as a standard mechanism for  
3 interacting between applications and the host environment has made it possible to relieve tool  
4 developers of this burden.

5 This section defines functions by which tools can request forwarding of input/output to/from other  
6 processes and serves as a design guide to:

- 7 • provide tool developers with an overview of the expected behavior of the PMIx IO forwarding  
8 support;
- 9 • guide RM vendors regarding roles and responsibilities expected of the RM to support IO  
10 forwarding; and
- 11 • provide insight into the thinking of the PMIx community behind the definition of the PMIx IO  
12 forwarding APIs.

13 Note that the forwarding of IO via PMIx requires that both the host environment and the tool  
14 support PMIx, but does not impose any similar requirements on the application itself.

15 The responsibility of the host environment in forwarding of IO falls into the following areas:

- 16 • Capturing output from specified processes.
- 17 • Forwarding that output to the host of the PMIx server library that requested it.
- 18 • Delivering that payload to the PMIx server library via the [PMIx\\_server\\_IOF\\_deliver](#) API  
19 for final dispatch to the requesting tool.

20 It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the  
21 requesting client. This may require caching of output until a forwarding registration is received, as  
22 governed by the corresponding IO forwarding attributes of Section 18.3.5 that are supported by the  
23 implementation.

### 24 18.3.1 Forwarding stdout/stderr

25 At an appropriate point in its operation (usually during startup), a tool will utilize the  
26 [PMIx\\_tool\\_init](#) function to connect to a PMIx server. The PMIx server can be hosted by an  
27 RM daemon or could be embedded in a library-provided starter program such as *mpiexec* - in terms  
28 of IO forwarding, the operations remain the same either way. For purposes of this discussion, we  
29 will assume the server is in an RM daemon and that the application processes are directly launched  
30 by the RM, as shown in Fig 18.4.

31 Once the tool has connected to the target server, it can request that processes be spawned on its  
32 behalf or that output from a specified set of existing processes in a given executing application be  
33 forwarded to it. Requests to spawn processes should include the [PMIX\\_FWD\\_STDIN](#),  
34 [PMIX\\_FWD\\_STDOUT](#), and/or [PMIX\\_FWD\\_STDERR](#) attributes if the tool intends to request that  
35 the corresponding streams be forwarded at some point during execution.

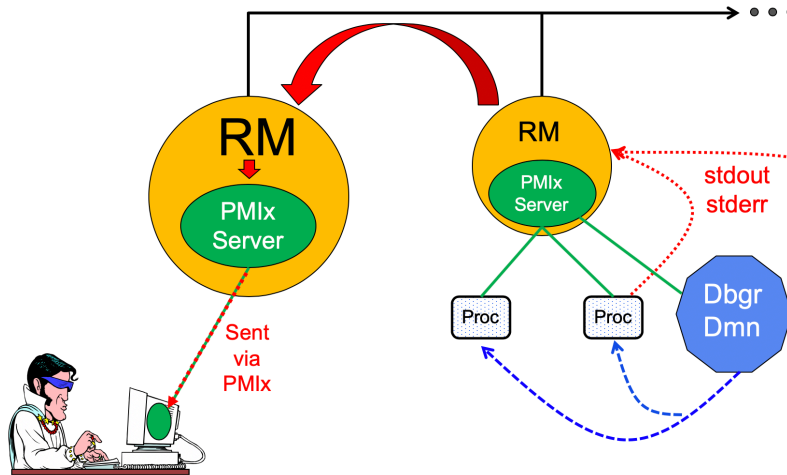


Figure 18.4.: Forwarding stdout/stderr

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:

- `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.
- `PMIX_IOF_REDIRECT` "`pmix.iof.redir`" (`bool`)  
Requests that the host environment intercept the specified output stream(s) and deliver it to the requesting tool instead of its current final destination. This might be used, for example, during a debugging procedure to avoid injection of debugger-related output into the application's results file. The original output stream(s) destination is restored upon termination of the tool. This is the default mode of operation.

When requesting to forward `stdout/stderr`, the tool can specify several formatting options to be used on the resulting output stream. These include:

- `PMIX_IOF_TAG_OUTPUT` "`pmix.iof.tag`" (`bool`)



1 Requests that output be prefixed with the nspace,rank of the source and a string  
2 identifying the channel (**stdout**, **stderr**, etc.).

- 3 • **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "**pmix.iof.ts**" (**bool**)

4 Requests that output be marked with the time at which the data was received by the tool -  
5 note that this will differ from the time at which the data was collected from the source.

- 6 • **PMIX\_IOF\_XML\_OUTPUT** "**pmix.iof.xml**" (**bool**)

7 Requests that output be formatted in XML.

8 The PMIx client in the tool is responsible for formatting the output stream. Note that output from  
9 multiple processes will often be interleaved due to variations in arrival time - ordering of output is  
10 not guaranteed across processes and/or nodes.

## 11 18.3.2 Forwarding stdin

12 A tool is not necessarily a child of the RM as it may have been started directly from the command  
13 line. Thus, provision must be made for the tool to collect its **stdin** and pass it to the host RM (via  
14 the PMIx server) for forwarding. Two methods of support for forwarding of **stdin** are defined:

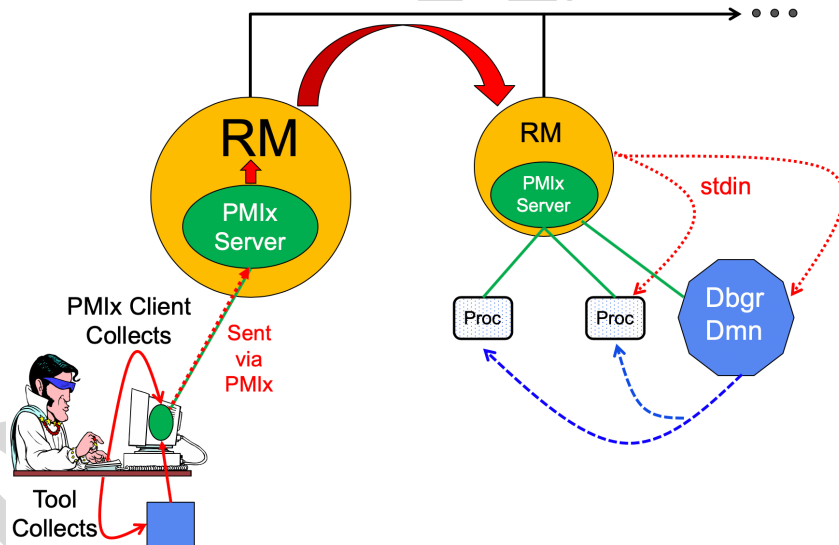


Figure 18.5.: Forwarding stdin

- 15 • internal collection by the PMIx tool library itself. This is requested via the  
16 **PMIX\_IOF\_PUSH\_STDIN** attribute in the **PMIx\_IOF\_push** call. When this mode is  
17 selected, the tool library begins collecting all **stdin** data and internally passing it to the local



server for distribution to the specified target processes. 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.

- external collection directly by the tool. It is assumed that the tool will provide its own code/mechanism for collecting its `stdin` as the tool developers may choose to insert some filtering and/or editing of the stream prior to forwarding it. In addition, the tool can directly control the targets for the data on a per-call basis – i.e., each call to `PMIx_IOF_push` can specify its own set of target recipients for that particular *blob* of data. Thus, this method provides maximum flexibility, but requires that the tool developer provide their own code to capture `stdin`.

Note that it is the responsibility of the RM to forward data to the host where the target process(es) are executing, and for the host daemon on that node to deliver the data to the `stdin` of target process(es). The PMIx server on the remote node is not involved in this process. Systems that do not support forwarding of `stdin` shall return `PMIx_ERR_NOT_SUPPORTED` in response to a forwarding request.

### Advice to users

Scalable forwarding of `stdin` represents a significant challenge. Most environments will at least handle a *send-to-1* model whereby `stdin` is forwarded to a single identified process, and occasionally an additional *send-to-all* model where `stdin` is forwarded to all processes in the application. Users are advised to check their host environment for available support as the distribution method lies outside the scope of PMIx.

`Stdin` buffering by the RM and/or PMIx library can be problematic. If any targeted recipient is slow reading data (or decides never to read data), then the data must be buffered in some intermediate daemon or the PMIx tool library itself. Thus, piping a large amount of data into `stdin` can result in a very large memory footprint in the system management stack or the tool. Best practices, therefore, typically focus on reading of input files by application processes as opposed to forwarding of `stdin`.

## 18.3.3 IO Forwarding Channels

*PMIx v3.0*

The `pmix_iof_channel_t` structure is a `uint16_t` type that defines a set of bit-mask flags for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple channels.

<code>PMIX_FWD_NO_CHANNELS</code>	Forward no channels.
<code>PMIX_FWD_STDIN_CHANNEL</code>	Forward <code>stdin</code> .
<code>PMIX_FWD_STDOUT_CHANNEL</code>	Forward <code>stdout</code> .
<code>PMIX_FWD_STDERR_CHANNEL</code>	Forward <code>stderr</code> .
<code>PMIX_FWD_STDDIAG_CHANNEL</code>	Forward <code>stddiag</code> , if available.
<code>PMIX_FWD_ALL_CHANNELS</code>	Forward all available channels.

## 1 18.3.4 IO Forwarding constants

2 **PMIX\_ERR\_IOF\_FAILURE** An IO forwarding operation failed - the affected channel will be  
3 included in the notification.

4 **PMIX\_ERR\_IOF\_COMPLETE** IO forwarding of the standard input for this process has  
5 completed - i.e., the stdin file descriptor has closed.

## 6 18.3.5 IO Forwarding attributes

7 The following attributes are used to control IO forwarding behavior at the request of tools. Use of  
8 the attributes is optional - any option not provided will revert to some implementation-specific  
9 value.

10 **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csize" (uint32\_t)

11 The requested size of the PMIx server cache in bytes for each specified channel. By default,  
12 the server is allowed (but not required) to drop all bytes received beyond the max size.

13 **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)

14 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
15 cache.

16 **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)

17 In an overflow situation, the PMIx server is to drop any new bytes received until room  
18 becomes available in the cache (default).

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 arrive.

28 **PMIX\_IOF\_COMPLETE** "pmix.iof.cmp" (bool)

29 Indicates that the specified IO channel has been closed by the source.

30 **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (bool)

31 Requests that output be prefixed with the nspace,rank of the source and a string identifying  
32 the channel (`stdout`, `stderr`, etc.).

33 **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (bool)

34 Requests that output be marked with the time at which the data was received by the tool -  
35 note that this will differ from the time at which the data was collected from the source.

36 **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)

37 Requests that output be formatted in XML.

38 **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 18.4 Debugger Support

17 Debuggers are a class of tool that merits special consideration due to their particular requirements  
18 for access to job-related information and control over process execution. The primary advantage of  
19 using PMIx for these purposes lies in the resulting portability of the debugger as it can be used with  
20 any system and/or programming model that supports PMIx. In addition to the general tool support  
21 described above, debugger support includes:

- 22 • Co-location, co-spawn, and communication wireup of debugger daemons for scalable launch.  
23 This includes providing debugger daemons with endpoint connection information across the  
24 daemons themselves.
- 25 • Identification of the job that is to be debugged. This includes automatically providing debugger  
26 daemons with the job-level information for their target job.

27 Debuggers can also utilize the options in the `PMIx_Spawn` API to exercise a degree of control  
28 over spawned jobs for debugging purposes. For example, a debugger can utilize the environmental  
29 parameter attributes of Section 12.2.4 to request `LD_PRELOAD` of a memory interceptor library  
30 prior to spawning an application process, or interject a custom fork/exec agent to shepherd the  
31 application process.

32 A key element of the debugging process is the ability of the debugger to require that processes  
33 *pause* at some well-defined point, thereby providing the debugger with an opportunity to attach and  
34 control execution. The actual implementation of the *pause* lies outside the scope of PMIx - it  
35 typically requires either the launcher or the application itself to implement the necessary  
36 operations. However, PMIx does provide several standard attributes by which the debugger can  
37 specify the desired attach point:

- 38 • **PMIX\_DEBUG\_STOP\_ON\_EXEC** "`pmix.dbg.exec`" (bool)

1 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the  
2 directive applies only to that application) or in the `job_info` array if it applies to all  
3 applications in the given spawn request. Indicates that the application is being spawned  
4 under a debugger, and that the local launch agent is to pause the resulting application  
5 processes on first instruction for debugger attach. The launcher (RM or IL) is to generate  
6 the `PMIX_LAUNCH_COMPLETE` event when all processes are stopped at the exec point.  
7 Launchers that cannot support this operation shall return an error from the `PMIx_Spawn`  
8 API if this behavior is requested.

9 • `PMIX_DEBUG_STOP_IN_INIT` "pmix.dbg.init" (bool)

10 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the  
11 directive applies only to that application) or in the `job_info` array if it applies to all  
12 applications in the given spawn request. Indicates that the specified application is being  
13 spawned under a debugger. The PMIx client library in each resulting application process  
14 shall notify its PMIx server that it is pausing and then pause during `PMIx_Init` of the  
15 spawned processes until either released by debugger modification of an appropriate  
16 variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL)  
17 is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when  
18 all processes have reached the pause point. PMIx implementations that do not support  
19 this operation shall return an error from `PMIx_Init` if this behavior is requested.  
20 Launchers that cannot support this operation shall return an error from the `PMIx_Spawn`  
21 API if this behavior is requested.

22 • `PMIX_DEBUG_WAIT_FOR_NOTIFY` "pmix.dbg.notify" (bool)

23 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the  
24 directive applies only to that application) or in the `job_info` array if it applies to all  
25 applications in the given spawn request. Indicates that the specified application is being  
26 spawned under a debugger. The resulting application processes are to notify their server  
27 (by generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach  
28 some application-determined location and pause at that point until either released by  
29 debugger modification of an appropriate variable or receipt of the  
30 `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for  
31 generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have  
32 indicated they are at the pause point. Launchers that cannot support this operation shall  
33 return an error from the `PMIx_Spawn` API if this behavior is requested.

34 Note that there is no mechanism by which the PMIx library or the launcher can verify that  
35 an application will recognize and support the `PMIX_DEBUG_WAIT_FOR_NOTIFY`  
36 request. Debuggers utilizing this attachment method must, therefore, be prepared to deal  
37 with the case where the application fails to recognize and/or honor the request.

38 If the PMIx implementation and/or the host environment support it, debuggers can utilize the  
39 `PMIx_Query_info` API to determine which features are available via the  
40 `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute.

41 • `PMIX_DEBUG_STOP_IN_INIT` by checking `PMIX_CLIENT_ATTRIBUTES` for the

1 `PMIx_Init` API.

- 2 • `PMIX_DEBUG_STOP_ON_EXEC` by checking `PMIX_HOST_ATTRIBUTES` for the  
3 `PMIx_Spawn` API.

4 The target namespace or process (as given by the debugger in the spawn request) shall be provided  
5 to each daemon in its job-level information via the `PMIX_DEBUG_TARGET` attribute. Debugger  
6 daemons are responsible for self-determining their specific target process(es), and can then utilize  
7 the `PMIx_Query_info` API to obtain information about them (see Fig 18.6) - e.g., to obtain the  
8 PIDs of the local processes to which they need to attach. PMIx provides the  
9 `pmix_proc_info_t` structure for organizing information about a process' PID, location, and  
10 state. Debuggers may request information on a given job at two levels:

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

23 Note that the information provided in the returned proctable represents a snapshot in time. Any  
24 process, regardless of role (tool, client, debugger, etc.) can obtain the proctable of a given  
25 namespace so long as it has the system-determined authorizations to do so. The list of namespaces  
26 available via a given server can be obtained using the `PMIx_Query_info` API with the  
27 `PMIX_QUERY_NAMESPACES` key.

28 Debugger daemons can be started in two ways - either at the same time the application is spawned,  
29 or separately at a later time.

## 30 18.4.1 Co-Location of Debugger Daemons

31 Debugging operations typically require the use of daemons that are located on the same node as the  
32 processes they are attempting to debug. The debugger can, of course, specify its own mapping  
33 method when issuing its spawn request or utilize its own internal launcher to place the daemons.  
34 However, when attaching to a running job, PMIx provides debuggers with a simplified method for  
35 requesting that the launcher associated with the job *co-locate* the required daemons. Debuggers can  
36 request *co-location* of their daemons by adding the following attributes to the `PMIx_Spawn` used  
37 to spawn them:

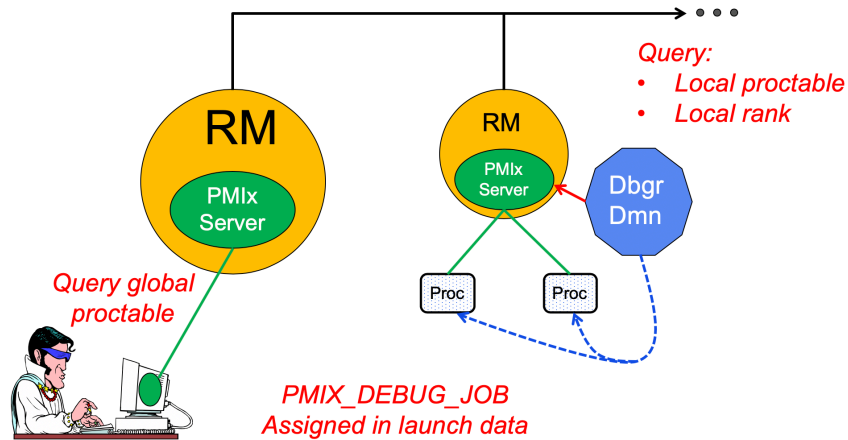


Figure 18.6.: Obtaining proctables

- 1 • **PMIX\_DEBUGGER\_DAEMONS** - indicating that the launcher is being asked to spawn debugger
- 2 daemons.
- 3 • **PMIX\_DEBUG\_TARGET** - indicating the job or process that is to be debugged. This allows the
- 4 launcher to identify the processes to be debugged and their location. Note that the debugger job
- 5 shall be assigned its own namespace (different from that of the job it is being spawned to debug)
- 6 and each daemon will be assigned a unique rank within that namespace.
- 7 • **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** - specifies the number of debugger daemons to be
- 8 co-located per target process.
- 9 • **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** - specifies the number of debugger daemons to be
- 10 co-located per node where at least one target process is executing.

11 Debugger daemons spawned in this manner shall be provided with the typical PMIx information for

12 their own job plus the target they are to debug via the **PMIX\_DEBUG\_TARGET** attribute. The

13 debugger daemons spawned on a given node are responsible for self-determining their specific

14 target process(es) - e.g., by referencing their own **PMIX\_LOCAL\_RANK** in the daemon debugger

15 job versus the corresponding **PMIX\_LOCAL\_RANK** of the target processes on the node. Note that

16 the debugger will be attaching to the application processes at some arbitrary point in the

17 application's execution unless some method for pausing the application (e.g., by providing a PMIx

18 directive at time of launch, or via a tool using the **PMIx\_Job\_control** API to direct that the

19 process be paused) has been employed.

▼ Advice to users ▼

20 Note that the tool calling **PMIx\_Spawn** to request the launch of the debugger daemons is *not*

21 included in the resulting job - i.e., the debugger daemons do not inherit the namespace of the tool.

1 Thus, collective operations and notifications that target the debugger daemon job will not include  
2 the tool unless the namespace/rank of the tool is explicitly included.

## 3 18.4.2 Co-Spawn of Debugger Daemons

4 In the case where a job is being spawned under the control of a debugger, PMIx provides a shortcut  
5 method for spawning the debugger's daemons in parallel with the job. This requires that the  
6 debugger be specified as one of the `pmix_app_t` in the same spawn command used to start the  
7 job. The debugger application must include at least the `PMIX_DEBUGGER_DAEMONS` attribute  
8 identifying itself as a debugger, and may utilize either a mapping option to direct daemon  
9 placement, or one of the `PMIX_DEBUG_DAEMONS_PER_PROC` or  
10 `PMIX_DEBUG_DAEMONS_PER_NODE` directives.

11 The launcher must not include information regarding the debugger daemons in the job-level info  
12 provided to the rest of the `pmix_app_t`s, nor in any calculated rank values (e.g.,  
13 `PMIX_NODE_RANK` or `PMIX_LOCAL_RANK`) in those applications. The debugger job is to be  
14 assigned its own namespace and each debugger daemon shall receive a unique rank - i.e., the  
15 debugger application is to be treated as a completely separate PMIx job that is simply being started  
16 in parallel with the user's applications. The launcher is free to implement the launch as a single  
17 operation for both the applications and debugger daemons (preferred), or may stage the launches as  
18 required. The launcher shall not return from the `PMIx_Spawn` command until all included  
19 applications and the debugger daemons have been started.

20 Attributes that apply to both the debugger daemons and the application processes can be specified  
21 in the `job_info` array passed into the `PMIx_Spawn` API. Attributes that either (a) apply solely to  
22 the debugger daemons or to one of the applications included in the spawn request, or (b) have  
23 values that differ from those provided in the `job_info` array, should be specified in the `info` array in  
24 the corresponding `pmix_app_t`. Note that PMIx job *pause* attributes (e.g.,  
25 `PMIX_DEBUG_STOP_IN_INIT`) do not apply to applications (defined in `pmix_app_t`) where  
26 the `PMIX_DEBUGGER_DAEMONS` attribute is set to `true`.

27 Debugger daemons spawned in this manner shall be provided with the typical PMIx information for  
28 their own job plus the target they are to debug via the `PMIX_DEBUG_TARGET` attribute. The  
29 debugger daemons spawned on a given node are responsible for self-determining their specific  
30 target process(es) - e.g., by referencing their own `PMIX_LOCAL_RANK` in the daemon debugger  
31 job versus the corresponding `PMIX_LOCAL_RANK` of the target processes on the node.



## Advice to users

1 Note that the tool calling `PMIx_Spawn` to request the launch of the debugger daemons is *not*  
2 included in the resulting job - i.e., the debugger daemons do not inherit the namespace of the tool.  
3 Thus, collective operations and notifications that target the debugger daemon job will not include  
4 the tool unless the namespace/rank of the tool is explicitly included.

5 The `PMIx_Spawn` API only supports the return of a single namespace resulting from the spawn  
6 request. In the case where the debugger job is co-spawned with the application, the spawn function  
7 shall return the namespace of the application and not the debugger job. Tools requiring access to  
8 the namespace of the debugger job must query the launcher for the spawned namespaces to find the  
9 one belonging to the debugger job.

### 18.4.3 Debugger Agents

11 Individual debuggers may, depending upon implementation, require varying degrees of control over  
12 each application process when it is started beyond those available via directives to `PMIx_Spawn`.  
13 PMIx offers two mechanisms to help provide a means of meeting these needs.

14 The `PMIX_FORKEXEC_AGENT` attribute allows the debugger to specify an intermediate process  
15 (the Fork/Exec Agent (FEA)) for spawning the actual application process (see Fig. 18.7a), thereby  
16 interposing the debugger daemon between the application process and the launcher's daemon.  
17 Instead of spawning the application process, the launcher will spawn the FEA, which will connect  
18 back to the PMIx server as a tool to obtain the spawn description of the application process it is  
19 to spawn. The PMIx server in the launcher's daemon shall not register the fork/exec agent as a local  
20 client process, nor shall the launcher include the agent in any of the job-level values (e.g.,  
21 `PMIX_RANK` within the job or `PMIX_LOCAL_RANK` on the node) provided to the application  
22 process. The launcher shall treat the collection of FEAs as a debugger job equivalent to the  
23 co-spawn use-case described in Section 18.4.2.

24 In contrast, the `PMIX_EXEC_AGENT` attribute (Fig. 18.7b) allows the debugger to specify an agent  
25 that will perform some preparatory actions and then exec the eventual application process to replace  
26 itself. In this scenario, the exec agent is provided with the application process' command line as  
27 arguments on its command line (e.g., `./agent appargv[0] appargv[1]`) and does not  
28 connect back to the host's PMIx server. It is the responsibility of the exec agent to properly separate  
29 its own command line arguments (if any) from the application description.



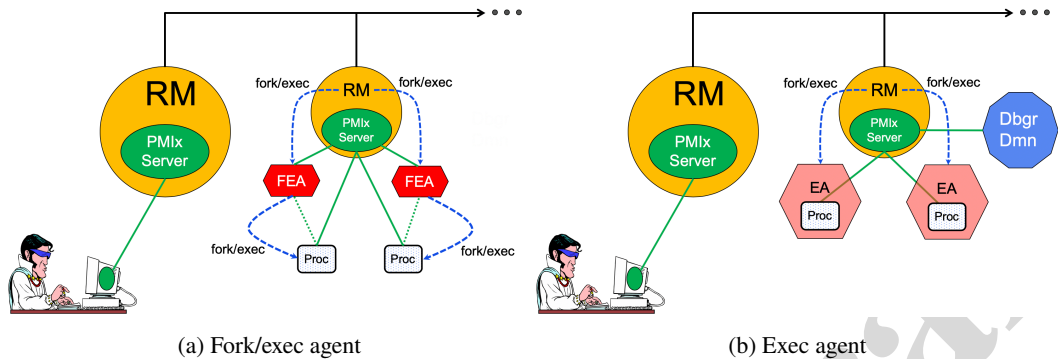


Figure 18.7.: Intermediate agents

## 18.4.4 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.

1 Users are therefore advised to specify the job whose events are of interest by including the  
2 **PMIX\_EVENT\_AFFECTED\_PROC** or **PMIX\_EVENT\_AFFECTED\_PROCS** attribute in the *info*  
3 array passed to the **PMIx\_Register\_event\_handler** API.

#### 4 18.4.4.1 Job lifecycle events

5 **PMIX\_EVENT\_JOB\_START** The first process in the job has been spawned - includes  
6 **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_JOBID** and/or **PMIX\_NAMESPACE** of the job.

7 **PMIX\_LAUNCH\_COMPLETE** All processes in the job have been spawned - includes  
8 **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_JOBID** and/or **PMIX\_NAMESPACE** of the job.

9 **PMIX\_EVENT\_JOB\_END** All processes in the job have terminated - includes  
10 **PMIX\_EVENT\_TIMESTAMP** when the last process terminated as well as the **PMIX\_JOBID**  
11 and/or **PMIX\_NAMESPACE** of the job.

12 **PMIX\_EVENT\_SESSION\_START** The allocation has been instantiated and is ready for use -  
13 includes **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_SESSION\_ID** of the allocation.  
14 This event is issued after any system-controlled prologue has completed, but before any  
15 user-specified actions are taken.

16 **PMIX\_EVENT\_SESSION\_END** The allocation has terminated - includes  
17 **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_SESSION\_ID** of the allocation. This  
18 event is issued after any user-specified actions have completed, but before any  
19 system-controlled epilogue is performed.

20 The following events relate to processes within a job:

21 **PMIX\_EVENT\_PROC\_TERMINATED** The specified process(es) terminated - normal or  
22 abnormal termination will be indicated by the **PMIX\_PROC\_TERM\_STATUS** in the *info*  
23 array of the notification. Note that a request for individual process events can generate a  
24 significant event volume from large-scale jobs.

25 **PMIX\_ERR\_PROC\_TERM\_WO\_SYNC** Process terminated without calling **PMIx\_Finalize**,  
26 or was a member of an assemblage formed via **PMIx\_Connect** and terminated or called  
27 **PMIx\_Finalize** without first calling **PMIx\_Disconnect** (or its non-blocking form)  
28 from that assemblage.

29 The following constants may be included via the **PMIX\_JOB\_TERM\_STATUS** attributed in the  
30 *info* array in the **PMIX\_EVENT\_JOB\_END** event notification to provide more detailed information  
31 regarding the reason for job abnormal termination:

32 **PMIX\_ERR\_JOB\_CANCELED** The job was canceled by the host environment.

33 **PMIX\_ERR\_JOB\_ABORTED** One or more processes in the job called abort, causing the job to  
34 be terminated.

35 **PMIX\_ERR\_JOB\_KILLED\_BY\_CMD** The job was killed by user command.

36 **PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG** The job was aborted due to receipt of an error signal  
37 (e.g., SIGKILL).

1 **PMIX\_ERR\_JOB\_TERM\_WO\_SYNC** The job was terminated due to at least one process  
2 terminating without calling **PMIx\_Finalize**, or was a member of an assemblage formed  
3 via **PMIx\_Connect** and terminated or called **PMIx\_Finalize** without first calling  
4 **PMIx\_Disconnect** (or its non-blocking form) from that assemblage.  
5 **PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED** The job was terminated due to one or more  
6 processes exceeding a specified sensor limit.  
7 **PMIX\_ERR\_JOB\_NON\_ZERO\_TERM** The job was terminated due to one or more processes  
8 exiting with a non-zero status.  
9 **PMIX\_ERR\_JOB\_ABORTED\_BY\_SYS\_EVENT** The job was aborted due to receipt of a  
10 system event.

#### 11 18.4.4.2 Job lifecycle attributes

12 **PMIX\_JOB\_TERM\_STATUS** "pmix.job.term.status" (**pmix\_status\_t**)  
13 Status returned by job upon its termination. The status will be communicated as part of a  
14 PMIx event payload provided by the host environment upon termination of a job. Note that  
15 generation of the **PMIX\_EVENT\_JOB\_END** event is optional and host environments may  
16 choose to provide it only upon request.  
17 **PMIX\_PROC\_STATE\_STATUS** "pmix.proc.state" (**pmix\_proc\_state\_t**)  
18 State of the specified process as of the last report - may not be the actual current state based  
19 on update rate.  
20 **PMIX\_PROC\_TERM\_STATUS** "pmix.proc.term.status" (**pmix\_status\_t**)  
21 Status returned by a process upon its termination. The status will be communicated as part  
22 of a PMIx event payload provided by the host environment upon termination of a process.  
23 Note that generation of the **PMIX\_EVENT\_PROC\_TERMINATED** event is optional and host  
24 environments may choose to provide it only upon request.

#### 25 18.4.5 Debugger-related constants

26 The following constants are used in events used to coordinate applications and the debuggers  
27 attaching to them.

28 **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** All processes in the job to be debugged are paused  
29 waiting for a release at some point within the application. The application shall remain in a  
30 paused state awaiting release until receipt of the **PMIX\_DEBUGGER\_RELEASE**.  
31 **PMIX\_DEBUGGER\_RELEASE** Release processes that are paused at the  
32 **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** point in the target application.

#### 33 18.4.6 Debugger attributes

34 Attributes used to assist debuggers - these are values that can either be passed to the **PMIx\_Spawn**  
35 APIs or accessed by a debugger itself using the **PMIx\_Get** API with the  
36 **PMIX\_RANK\_WILDCARD** rank.

37 **PMIX\_DEBUG\_STOP\_ON\_EXEC** "pmix.dbg.exec" (**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 application is being spawned under a debugger, and  
4 that the local launch agent is to pause the resulting application processes on first instruction  
5 for debugger attach. The launcher (RM or IL) is to generate the  
6 `PMIX_LAUNCH_COMPLETE` event when all processes are stopped at the exec point.

7 **PMIX\_DEBUG\_STOP\_IN\_INIT** "pmix.dbg.init" (bool)

8 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
9 applies only to that application) or in the `job_info` array if it applies to all applications in the  
10 given spawn request. Indicates that the specified application is being spawned under a  
11 debugger. The PMIx client library in each resulting application process shall notify its PMIx  
12 server that it is pausing and then pause during `PMIx_Init` of the spawned processes until  
13 either released by debugger modification of an appropriate variable or receipt of the  
14 `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating  
15 the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have reached the  
16 pause point.

17 **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** "pmix.dbg.notify" (bool)

18 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
19 applies only to that application) or in the `job_info` array if it applies to all applications in the  
20 given spawn request. Indicates that the specified application is being spawned under a  
21 debugger. The resulting application processes are to notify their server (by generating the  
22 `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some  
23 application-determined location and pause at that point until either released by debugger  
24 modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE`  
25 event. The launcher (RM or IL) is responsible for generating the  
26 `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they are  
27 at the pause point.

28 **PMIX\_DEBUG\_TARGET** "pmix.dbg.tgt" (`pmix_proc_t*`)

29 Identifier of process(es) to be debugged - a rank of `PMIX_RANK_WILDCARD` indicates that  
30 all processes in the specified namespace are to be included.

31 **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)

32 Included in the `pmix_info_t` array of a `pmix_app_t`, this attribute declares that the  
33 application consists of debugger daemons and shall be governed accordingly. If used as the  
34 sole `pmix_app_t` in a `PMIx_Spawn` request, then the `PMIX_DEBUG_TARGET` attribute  
35 must also be provided (in either the `job_info` or in the `info` array of the `pmix_app_t`) to  
36 identify the namespace to be debugged so that the launcher can determine where to place the  
37 spawned daemons. If neither `PMIX_DEBUG_DAEMONS_PER_PROC` nor  
38 `PMIX_DEBUG_DAEMONS_PER_NODE` is specified, then the launcher shall default to a  
39 placement policy of one daemon per process in the target job.

40 **PMIX\_COSPAWN\_APP** "pmix.cospawn" (bool)

41 Designated application is to be spawned as a disconnected job - i.e., the launcher shall not  
42 include the application in any of the job-level values (e.g., `PMIX_RANK` within the job)  
43 provided to any other application process generated by the same spawn request. Typically

used to cospawn debugger daemons alongside an application.

**PMIX\_DEBUG\_DAEMONS\_PER\_PROC** "pmix.dbg.dpproc" (uint16\_t)

Number of debugger daemons to be spawned per application process. The launcher is to pass the identifier of the namespace to be debugged by including the **PMIX\_DEBUG\_TARGET** attribute in the daemon's job-level information. 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.

**PMIX\_DEBUG\_DAEMONS\_PER\_NODE** "pmix.dbg.dpnd" (uint16\_t)

Number of debugger daemons to be spawned on each node where the target job is executing. The launcher is to pass the identifier of the namespace to be debugged by including the **PMIX\_DEBUG\_TARGET** attribute in the daemon's job-level information. 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.

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

Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace whose process table is being queried.

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

Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each process in the specified namespace executing on the same node as the requester, ordered by process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME** indicating the host whose local process table is being queried. By default, the query assumes that the host upon which the request was made is to be used.

## 18.5 Tool-Specific APIs

PMIx-based tools automatically have access to all PMIx client functions. Tools designated as a *launcher* or a *server* will also have access to all PMIx server functions. There are, however, an additional set of functions (described in this section) that are specific to a PMIx tool. Access to those functions require use of the tool initialization routine.

### 18.5.1 PMIx\_tool\_init

#### Summary

Initialize the PMIx library for operating as a tool, optionally connecting to a specified PMIx server.

#### Format

PMIx v2.0

```

1  pmix_status_t
2  PMIx_tool_init(pmix_proc_t *proc,
3                pmix_info_t info[], size_t ninfo);

```

**INOUT** `proc`

`pmix_proc_t` structure (handle)

**IN** `info`

Array of `pmix_info_t` structures (array of handles)

**IN** `ninfo`

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

Returns `PMIX_SUCCESS` or a negative value indicating the error.

### Required Attributes

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

**PMIX\_TOOL\_NAMESPACE** "`pmix.tool.namespace`" (`char*`)

Name of the namespace to use for this tool.

**PMIX\_TOOL\_RANK** "`pmix.tool.rank`" (`uint32_t`)

Rank of this tool.

**PMIX\_TOOL\_DO\_NOT\_CONNECT** "`pmix.tool.nocon`" (`bool`)

The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

**PMIX\_TOOL\_ATTACHMENT\_FILE** "`pmix.tool.attach`" (`char*`)

Pathname of file containing connection information to be used for attaching to a specific server.

**PMIX\_SERVER\_URI** "`pmix.srvr.uri`" (`char*`)

URI of the PMIx server to be contacted.

**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\_SERVER\_PIDINFO** "`pmix.srvr.pidinfo`" (`pid_t`)

PID of the target PMIx server for a tool.

**PMIX\_SERVER\_NAMESPACE** "`pmix.srv.namespace`" (`char*`)

Name of the namespace to use for this PMIx server.

**PMIX\_CONNECT\_TO\_SYSTEM** "`pmix.cnct.sys`" (`bool`)

The requester requires that a connection be made only to a local, system-level PMIx server.

**PMIX\_CONNECT\_SYSTEM\_FIRST** "`pmix.cnct.sys.first`" (`bool`)

1 Preferentially, look for a system-level PMIx server first.

## 2 Optional Attributes

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

4 **PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (uint32\_t)

5 Time in seconds between connection attempts to a PMIx server - the default value is  
6 implementation specific.

7 **PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)

8 Maximum number of times to try to connect to PMIx server - the default value is  
9 implementation specific.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



1 **PMIX\_EXTERNAL\_PROGRESS** "pmix.evext" (bool)

2 The host shall progress the PMIx library via calls to **PMIx\_Progress**

3 **PMIX\_EVENT\_BASE** "pmix.evbase" (void\*)

4 Pointer to an **event\_base** to use in place of the internal progress thread. All PMIx library  
5 events are to be assigned to the provided event base. The event base *must* be compatible with  
6 the event library used by the PMIx implementation - e.g., either both the host and PMIx  
7 library must use libevent, or both must use libev. Cross-matches are unlikely to work and  
8 should be avoided - it is the responsibility of the host to ensure that the PMIx  
9 implementation supports (and was built with) the appropriate event library.

---

## 10 **Description**

11 Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided  
12 **pmix\_proc\_t** struct. The *info* array is used to pass user requests pertaining to the initialization  
13 and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives  
14 are desired.

15 If called with the **PMIX\_TOOL\_DO\_NOT\_CONNECT** attribute, the PMIx tool library will fully  
16 initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later  
17 point in time, if desired, by calling the **PMIx\_tool\_attach\_to\_server** function. If provided,  
18 the *proc* structure will be set to a zero-length namespace and a rank of **PMIX\_RANK\_UNDEF** unless  
19 the **PMIX\_TOOL\_NAMESPACE** and **PMIX\_TOOL\_RANK** attributes are included in the *info* array.

20 In all other cases, the PMIx tool library will automatically attempt to connect to a PMIx server  
21 according to the precedence chain described in Section 18.1. If successful, the function will return  
22 **PMIX\_SUCCESS** and will fill the process structure (if provided) with the assigned namespace and  
23 rank of the tool. The server to which the tool connects will be designated its *primary* server. Note  
24 that each connection attempt in the above precedence chain will retry (with delay between each  
25 retry) a number of times according to the values of the corresponding attributes.

26 Note that the PMIx tool library is referenced counted, and so multiple calls to **PMIx\_tool\_init**  
27 are allowed. If the tool is not connected to any server when this API is called, then the tool will  
28 attempt to connect to a server unless the **PMIX\_TOOL\_DO\_NOT\_CONNECT** is included in the call  
29 to API.

## 30 **18.5.2 PMIx\_tool\_finalize**

### 31 **Summary**

32 Finalize the PMIx tool library.



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29

## Format

C

```
pmix_status_t  
PMIx_tool_finalize(void);
```

C

Returns **PMIX\_SUCCESS** or a negative value indicating the error.

## Description

Finalize the PMIx tool library, closing all existing connections to servers. An error code will be returned if, for some reason, a connection cannot be cleanly terminated — in such cases, the connection is dropped. Upon detecting loss of the connection, the PMIx server shall cleanup all associated records of the tool.

## 18.5.3 PMIx\_tool\_disconnect

### Summary

Disconnect the PMIx tool from the specified server connection while leaving the tool library initialized.

### Format

C

*PMIx v4.0*

```
pmix_status_t  
PMIx_tool_disconnect(const pmix_proc_t *server);
```

C

### IN server

**pmix\_proc\_t** structure (handle)

Returns **PMIX\_SUCCESS** or a negative value indicating the error.

### Description

Close the current connection to the specified server, if one has been made, while leaving the PMIx library initialized. An error code will be returned if, for some reason, the connection cannot be cleanly terminated - in this case, the connection is dropped. In either case, the library will remain initialized. Upon detecting loss of the connection, the PMIx server shall cleanup all associated records of the tool.

Note that if the server being disconnected is the current *primary* server, then all operations requiring support from a server will return the **PMIX\_ERR\_UNREACH** error until the tool either designates an existing connection to be the *primary* server or, if no other connections exist, the tool establishes a connection to a PMIx server.

## 1 18.5.4 PMix\_tool\_attach\_to\_server

### 2 Summary

3 Establish a connection to a PMix server.

### 4 Format

```
5 pmix_status_t  
6 PMix_tool_attach_to_server(pmix_proc_t *proc,  
7                             pmix_proc_t *server,  
8                             pmix_info_t info[],  
9                             size_t ninfo);
```

#### 10 INOUT proc

11 Pointer to `pmix_proc_t` structure (handle)

#### 12 INOUT server

13 Pointer to `pmix_proc_t` structure (handle)

#### 14 IN info

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

#### 16 IN ninfo

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

18 Returns `PMIX_SUCCESS` or a negative value indicating the error.

### 19 Required Attributes

20 The following attributes are required to be supported by all PMix libraries:

21 **PMIX\_TOOL\_ATTACHMENT\_FILE** "pmix.tool.attach" (char\*)

22 Pathname of file containing connection information to be used for attaching to a specific server.

23 **PMIX\_SERVER\_URI** "pmix.srvr.uri" (char\*)

24 URI of the PMix server to be contacted.

25 **PMIX\_TCP\_URI** "pmix.tcp.uri" (char\*)

26 The URI of the PMix server to connect to, or a file name containing it in the form of  
27 file:<name of file containing it>.

28 **PMIX\_SERVER\_PIDINFO** "pmix.srvr.pidinfo" (pid\_t)

29 PID of the target PMix server for a tool.

30 **PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (char\*)

31 Name of the namespace to use for this PMix server.

32 **PMIX\_CONNECT\_TO\_SYSTEM** "pmix.cnct.sys" (bool)

33 The requester requires that a connection be made only to a local, system-level PMix server.

34 **PMIX\_CONNECT\_SYSTEM\_FIRST** "pmix.cnct.sys.first" (bool)

1 Preferentially, look for a system-level PMIx server first.

2 **PMIX\_PRIMARY\_SERVER** "pmix.pri.svr" (bool)

3 The server to which the tool is connecting shall be designated the *primary* server once  
4 connection has been accomplished.

### 5 **Description**

6 Establish a connection to a server. This function can be called at any time by a PMIx tool to create a  
7 new connection to a server. If a specific server is given and the tool is already attached to it, then  
8 the API shall return **PMIX\_SUCCESS** without taking any further action. In all other cases, the tool  
9 will attempt to discover a server using the method described in Section 18.1, ignoring all candidates  
10 to which it is already connected. The **PMIX\_ERR\_UNREACH** error shall be returned if no new  
11 connection is made.

12 The process identifier assigned to this tool is returned in the provided *proc* structure. Passing a  
13 value of **NULL** for the *proc* parameter is allowed if the user wishes solely to connect to a PMIx  
14 server and does not require return of the identifier at that time.

15 The process identifier of the server to which the tool attached is returned in the *server* structure.  
16 Passing a value of **NULL** for the *proc* parameter is allowed if the user wishes solely to connect to a  
17 PMIx server and does not require return of the identifier at that time.

18 Note that the **PMIX\_PRIMARY\_SERVER** attribute must be included in the *info* array if the server  
19 being connected to is to become the primary server, or a call to **PMIx\_tool\_set\_server** must  
20 be provided immediately after the call to this function.

### 21 **Advice to PMIx library implementers**

22 When a tool connects to a server that is under a different namespace manager (e.g., host RM) from  
23 the prior server, the namespace in the identifier of the tool must remain unique in the new universe.  
24 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.

### 25 **Advice to users**

26 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.

## 27 **18.5.5 PMIx\_tool\_get\_servers**

### 28 **Summary**

29 Get an array containing the **pmix\_proc\_t** process identifiers of all servers to which the tool is  
30 currently connected.

1 **Format** C

```
2 pmix_status_t  
3 PMIx_tool_get_servers(pmix_proc_t *servers[], size_t *nservers);  
4 C
```

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 indicating the error.

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 18.5.6 PMIx\_tool\_set\_server

14 **Summary**  
15 Designate a server as the tool's *primary* server.

16 *PMIx v4.0* **Format** C

```
17 pmix_status_t  
18 PMIx_tool_set_server(const pmix_proc_t *server pmix_inf  
19 info[], size_t ninfo);  
20 C
```

20 **IN server**  
21 `pmix_proc_t` structure (handle)  
22 **IN info**  
23 Array of `pmix_info_t` structures (array of handles)  
24 **IN ninfo**  
25 Number of elements in the *info* array (`size_t`)

26 Returns `PMIX_SUCCESS` or a negative value indicating the error.

## Required Attributes

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

**PMIX\_WAIT\_FOR\_CONNECTION** "pmix.wait.conn" (bool)

Wait until the specified process has connected to the requesting tool or server, or the operation times out (if the **PMIX\_TIMEOUT** directive is included in the request).

**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

Designate the specified server to be the tool's *primary* server for all subsequent API calls.

## 18.5.7 PMIx\_IOF\_pull

### Summary

Register to receive output forwarded from a set of remote processes.

### Format

PMIx v3.0

C

```
pmix_status_t
```

```
PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,  
              const pmix_info_t directives[], size_t ndirs,  
              pmix_iof_channel_t channel,  
              pmix_iof_cbfunc_t cbfunc,  
              pmix_hdlr_reg_cbfunc_t regcbfunc,  
              void *regcbdata);
```

C

**IN** **procs**

Array of proc structures identifying desired source processes (array of handles)

**IN** **nprocs**

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

**IN** **directives**

Array of **pmix\_info\_t** structures (array of handles)

**IN** **ndirs**

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

**IN** **channel**

Bitmask of IO channels included in the request (**pmix\_iof\_channel\_t**)

**IN** **cbfunc**

Callback function for delivering relevant output (**pmix\_iof\_cbfunc\_t** function reference)

1 **IN regcbfunc**  
2 Function to be called when registration is completed (`pmix_hdlr_reg_cbfunc_t`  
3 function reference)

4 **IN regcbdata**  
5 Data to be passed to the `regcbfunc` callback function (memory reference)

6 Returns `PMIX_SUCCESS` or a negative value indicating the error. In the event the function returns  
7 an error, the `regcbfunc` will *not* be called.

▼----- Required Attributes -----▼

8 The following attributes are required for PMIx libraries that support IO forwarding:

9 **PMIX\_IOF\_CACHE\_SIZE** "`pmix.iof.csize`" (`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 are optional for PMIx libraries that support IO forwarding:

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.

29 **PMIX\_IOF\_TAG\_OUTPUT** "`pmix.iof.tag`" (`bool`)  
30 Requests that output be prefixed with the namespace, rank of the source and a string identifying  
31 the channel (`stdout`, `stderr`, etc.).

32 **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "`pmix.iof.ts`" (`bool`)  
33 Requests that output be marked with the time at which the data was received by the tool -  
34 note that this will differ from the time at which the data was collected from the source.

1 **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)

2 Requests that output be formatted in XML.



### 3 **Description**

4 Register to receive output forwarded from a set of remote processes.

#### ▼ **Advice to users** ▼

5 Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated  
6 channels to be written to their corresponding **stdout/stderr** file descriptors. Use of  
7 **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but should be  
8 used carefully due to bandwidth and memory footprint considerations.



## 9 **18.5.8 PMIx\_IOF\_deregister**

### 10 **Summary**

11 Deregister from output forwarded from a set of remote processes.

### 12 **Format**

PMIx v3.0

```
13 pmix_status_t  
14 PMIx_IOF_deregister(size_t iofhdlr,  
15                     const pmix_info_t directives[], size_t ndirs,  
16                     pmix_op_cbfunc_t cbfunc, void *cbdata);
```

#### 17 **IN iofhdlr**

18 Registration number returned from the **pmix\_hdlr\_reg\_cbfunc\_t** callback from the  
19 call to **PMIx\_IOF\_pull** (**size\_t**)

#### 20 **IN directives**

21 Array of **pmix\_info\_t** structures (array of handles)

#### 22 **IN ndirs**

23 Number of elements in the *directives* array (integer)

#### 24 **IN cbfunc**

25 Callback function to be called when deregistration has been completed. (function reference)

#### 26 **IN cbdata**

27 Data to be passed to the *cbfunc* callback function (memory reference)

28 A successful return indicates that the request is being processed and the result will be returned in  
29 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
30 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

31 Returns **PMIX\_SUCCESS** or one of the following error codes when the condition described occurs:

1 **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed  
2 successfully - the *cbfunc* will *not* be called.

3 If none of the above return codes are appropriate, then an implementation must return either a  
4 general PMIx error code or an implementation defined error code as described in Section 3.1.1.

## 5 **Description**

6 Deregister from output forwarded from a set of remote processes.

### ▼ Advice to PMIx library implementers ▼

7 Any currently buffered IO should be flushed upon receipt of a deregistration request. All received  
8 IO after receipt of the request shall be discarded.



## 9 **18.5.9 PMIx\_IOF\_push**

### 10 **Summary**

11 Push data collected locally (typically from **stdin** or a file) to **stdin** of the target recipients.

### 12 **Format**

*PMIx v3.0*

C

```
13 pmix_status_t  
14 PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,  
15               pmix_byte_object_t *bo,  
16               const pmix_info_t directives[], size_t ndirs,  
17               pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

18 **IN targets**

Array of proc structures identifying desired target processes (array of handles)

20 **IN ntargets**

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

22 **IN bo**

Pointer to [pmix\\_byte\\_object\\_t](#) containing the payload to be delivered (handle)

24 **IN directives**

Array of [pmix\\_info\\_t](#) structures (array of handles)

26 **IN ndirs**

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

28 **IN directives**

Array of [pmix\\_info\\_t](#) structures (array of handles)

30 **IN cbfunc**

Callback function to be called when operation has been completed. ([pmix\\_op\\_cbfunc\\_t](#) function reference)



1 **IN cldata**

2 Data to be passed to the *cbfunc* callback function (memory reference)

3 A successful return indicates that the request is being processed and the result will be returned in  
4 the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning  
5 from the API. The callback function, *cbfunc*, is only called when **PMIX\_SUCCESS** is returned.

6 Returns PMIX\_SUCCESS or one of the following error codes when the condition described occurs:

7 **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed  
8 successfully - the *cbfunc* will *not* be called.

9 If none of the above return codes are appropriate, then an implementation must return either a  
10 general PMIX error code or an implementation defined error code as described in Section 3.1.1.

▼----- Required Attributes -----▼

11 The following attributes are required for PMIX libraries that support IO forwarding:

12 **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csize" (uint32\_t)

13 The requested size of the PMIX server cache in bytes for each specified channel. By default,  
14 the server is allowed (but not required) to drop all bytes received beyond the max size.

15 **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)

16 In an overflow situation, the PMIX server is to drop the oldest bytes to make room in the  
17 cache.

18 **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)

19 In an overflow situation, the PMIX server is to drop any new bytes received until room  
20 becomes available in the cache (default).

▲----- Optional Attributes -----▼

21 The following attributes are optional for PMIX libraries that support IO forwarding:

22 **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)

23 Requests that IO on the specified channel(s) be aggregated in the PMIX tool library until the  
24 specified number of bytes is collected to avoid being called every time a block of IO arrives.  
25 The PMIX tool library will execute the callback and reset the collection counter whenever the  
26 specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
27 the callback upon a call to deregister the respective channel.

28 **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)

29 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
30 size, this prevents IO from being held indefinitely while waiting for another payload to  
31 arrive.

32 **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 Description

7 Called either to:

- 8 • push data collected by the caller themselves (typically from **stdin** or a file) to **stdin** of the  
9 target recipients;
- 10 • request that the PMIx library automatically collect and push the **stdin** of the caller to the target  
11 recipients; or
- 12 • indicate that automatic collection and transmittal of **stdin** is to stop

### Advice to users

13 Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires  
14 the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets.  
15 Use of **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but  
16 should be used carefully due to bandwidth and memory footprint considerations.

# Storage Support Definitions

---

**Provisional**

Distributed and parallel computing systems are increasingly embracing storage hierarchies to meet the diverse data management needs of applications and other systems software in a cost-effective manner. These hierarchies provide access to a number of distinct storage layers, with each potentially composed of different storage hardware (e.g., HDD, SSD, tape, PMEM), deployed at different locations (e.g., on-node, on-switch, on-site, WAN), and designed using different storage paradigms (e.g., file-based, object-based). Each of these systems offers unique performance and usage characteristics that storage system users should carefully consider to ensure the most efficient use of storage resources.

PMIx enables users to better understand storage hierarchies by defining attributes that formalize storage system characteristics, state, and other parameters. These attributes can be queried by applications, I/O libraries and middleware, and workflow systems to discover available storage resources and to inform on which resources are most suitable for different I/O workload requirements.

## 19.1 Storage support constants

**Provisional**

The `pmix_storage_medium_t` is a `uint64_t` type that defines a set of bit-mask flags for specifying different types of storage mediums. These can be bitwise OR'd together to accommodate storage systems that mix storage medium types.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_UNKNOWN** The storage medium type is unknown.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_TAPE** The storage system uses tape media.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_HDD** The storage system uses HDDs with traditional SAS, SATA interfaces.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_SSD** The storage system uses SSDs with traditional SAS, SATA interfaces.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_NVME** The storage system uses SSDs with NVMe interface.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_PMEM** The storage system uses persistent memory.

**Provisional**

**PMIX\_STORAGE\_MEDIUM\_RAM** The storage system is volatile (e.g., tmpfs).

## Advice to PMIx library implementers

PMIx implementations should maintain the same ordering for bit-mask values for `pmix_storage_medium_t` struct as provided in this standard, since these constants are ordered to provide semantic information that may be of use to PMIx users. Namely, `pmix_storage_medium_t` constants are ordered in terms of increasing medium bandwidth.

It is further recommended that implementations should try to allocate empty bits in the mask so that they can be extended to account for new constant definitions corresponding to new storage mediums.

- Provisional* The `pmix_storage_accessibility_t` is a `uint64_t` type that defines a set of bit-mask flags for specifying different levels of storage accessibility (i.e., from where a storage system may be accessed). These can be bitwise OR'd together to accommodate storage systems that are accessible in multiple ways.
- Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_NODE** The storage system resources are accessible within the same node.
- Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_SESSION** The storage system resources are accessible within the same session.
- Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_JOB** The storage system resources are accessible within the same job.
- Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_RACK** The storage system resources are accessible within the same rack.
- Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_CLUSTER** The storage system resources are accessible within the same cluster.
- Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_REMOTE** The storage system resources are remote.
- Provisional* The `pmix_storage_persistence_t` type specifies different levels of persistence for a particular storage system.
- Provisional* **PMIX\_STORAGE\_PERSISTENCE\_TEMPORARY** Data on the storage system is persisted only temporarily (i.e., it does not survive across sessions or node reboots).
- Provisional* **PMIX\_STORAGE\_PERSISTENCE\_NODE** Data on the storage system is persisted on the node.
- Provisional* **PMIX\_STORAGE\_PERSISTENCE\_SESSION** Data on the storage system is persisted for the duration of the session.
- Provisional* **PMIX\_STORAGE\_PERSISTENCE\_JOB** Data on the storage system is persisted for the duration of the job.
- Provisional* **PMIX\_STORAGE\_PERSISTENCE\_SCRATCH** Data on the storage system is persisted according to scratch storage policies (short-term storage, typically persisted for days to weeks).
- Provisional* **PMIX\_STORAGE\_PERSISTENCE\_PROJECT** Data on the storage system is persisted according to project storage policies (long-term storage, typically persisted for the duration of a project).

1	<b>PMIX_STORAGE_PERSISTENCE_ARCHIVE</b>	Data on the storage system is persisted
2		according to archive storage policies (long-term storage, typically persisted indefinitely).
3		The <b>pmix_storage_access_type_t</b> type specifies different storage system access types.
4	<i>Provisional</i> <b>PMIX_STORAGE_ACCESS_RD</b>	Provide information on storage system read operations.
5	<i>Provisional</i> <b>PMIX_STORAGE_ACCESS_WR</b>	Provide information on storage system write operations.
6	<i>Provisional</i> <b>PMIX_STORAGE_ACCESS_RDWR</b>	Provide information on storage system read and write
7		operations.

## 8 19.2 Storage support attributes

9 The following attributes may be returned in response to queries (e.g., **PMIx\_Get** or  
10 **PMIx\_Query\_info**) made by processes or tools.

11	<i>Provisional</i> <b>PMIX_STORAGE_ID</b>	<b>"pmix.strg.id" (char*)</b>
12		An identifier for the storage system (e.g., lustre-fs1, daos-oss1, home-fs)
13	<i>Provisional</i> <b>PMIX_STORAGE_PATH</b>	<b>"pmix.strg.path" (char*)</b>
14		Mount point path for the storage system (valid only for file-based storage systems)
15	<i>Provisional</i> <b>PMIX_STORAGE_TYPE</b>	<b>"pmix.strg.type" (char*)</b>
16		Type of storage system (i.e., "lustre", "gpfs", "daos", "ext4")
17	<i>Provisional</i> <b>PMIX_STORAGE_VERSION</b>	<b>"pmix.strg.ver" (char*)</b>
18		Version string for the storage system
19	<i>Provisional</i> <b>PMIX_STORAGE_MEDIUM</b>	<b>"pmix.strg.medium" (pmix_storage_medium_t)</b>
20		Types of storage mediums utilized by the storage system (e.g., SSDs, HDDs, tape)
21		<b>PMIX_STORAGE_ACCESSIBILITY</b>
22	<i>Provisional</i>	<b>"pmix.strg.access" (pmix_storage_accessibility_t)</b>
23		Accessibility level of the storage system (e.g., within same node, within same session)
24		<b>PMIX_STORAGE_PERSISTENCE</b>
25	<i>Provisional</i>	<b>"pmix.strg.persist" (pmix_storage_persistence_t)</b>
26		Persistence level of the storage system (e.g., scratch storage or archive storage)
27	<i>Provisional</i> <b>PMIX_QUERY_STORAGE_LIST</b>	<b>"pmix.strg.list" (char*)</b>
28		Comma-delimited list of storage identifiers (i.e., <b>PMIX_STORAGE_ID</b> types) for available
29		storage systems
30	<i>Provisional</i> <b>PMIX_STORAGE_CAPACITY_LIMIT</b>	<b>"pmix.strg.caplim" (double)</b>
31		Overall limit on capacity (in bytes) for the storage system
32	<i>Provisional</i> <b>PMIX_STORAGE_CAPACITY_USED</b>	<b>"pmix.strg.capuse" (double)</b>
33		Overall used capacity (in bytes) for the storage system
34	<i>Provisional</i> <b>PMIX_STORAGE_OBJECT_LIMIT</b>	<b>"pmix.strg.objlim" (uint64_t)</b>
35		Overall limit on number of objects (e.g., inodes) for the storage system
36	<i>Provisional</i> <b>PMIX_STORAGE_OBJECTS_USED</b>	<b>"pmix.strg.objuse" (uint64_t)</b>
37		Overall used number of objects (e.g., inodes) for the storage system
38	<i>Provisional</i> <b>PMIX_STORAGE_MINIMAL_XFER_SIZE</b>	<b>"pmix.strg.minxfer" (double)</b>

1 Minimal transfer size (in bytes) for the storage system - this is the storage system's atomic  
2 unit of transfer (e.g., block size)

3 *Provisional* **PMIX\_STORAGE\_SUGGESTED\_XFER\_SIZE** "**pmix.strg.sxfer**" (**double**)

4 Suggested transfer size (in bytes) for the storage system

5 *Provisional* **PMIX\_STORAGE\_BW\_MAX** "**pmix.strg.bwmax**" (**double**)

6 Maximum bandwidth (in bytes/sec) for storage system - provided as the theoretical  
7 maximum or the maximum observed bandwidth value

8 *Provisional* **PMIX\_STORAGE\_BW\_CUR** "**pmix.strg.bwcur**" (**double**)

9 Observed bandwidth (in bytes/sec) for storage system - provided as a recently observed  
10 bandwidth value, with the exact measurement interval depending on the storage system  
11 and/or PMIx library implementation

12 *Provisional* **PMIX\_STORAGE\_IOPS\_MAX** "**pmix.strg.iopsmax**" (**double**)

13 Maximum IOPS (in I/O operations per second) for storage system - provided as the  
14 theoretical maximum or the maximum observed IOPS value

15 *Provisional* **PMIX\_STORAGE\_IOPS\_CUR** "**pmix.strg.iopscur**" (**double**)

16 Observed IOPS (in I/O operations per second) for storage system - provided as a recently  
17 observed IOPS value, with the exact measurement interval depending on the storage system  
18 and/or PMIx library implementation

19 **PMIX\_STORAGE\_ACCESS\_TYPE**

20 *Provisional* "**pmix.strg.atype**" (**pmix\_storage\_access\_type\_t**)

21 Qualifier describing the type of storage access to return information for (e.g., for qualifying  
22 **PMIX\_STORAGE\_BW\_CUR**, **PMIX\_STORAGE\_IOPS\_CUR**, or  
23 **PMIX\_STORAGE\_SUGGESTED\_XFER\_SIZE** attributes)

## 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.

## 11 A.1 Design Considerations

12 Several issues arose during design of the Python bindings:

### 13 A.1.1 Error Codes vs Python Exceptions

14 The C programming language reports errors through the return of the corresponding integer status  
15 codes. PMIx has defined a range of negative values for this purpose. However, Python has the  
16 option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program  
17 appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and  
18 return PMIx status codes in lieu of raising exceptions as this method was considered more  
19 consistent for those working in both domains.

### 20 A.1.2 Representation of Structured Data

21 PMIx utilizes a number of C-language structures to efficiently bundle related information. For  
22 example, the PMIx process identifier is represented as a struct containing a character array for the  
23 namespace and a 32-bit unsigned integer for the process rank. There are several options for  
24 translating such objects to Python – e.g., the PMIx process identifier could be represented as a  
25 two-element tuple (nspace, rank) or as a dictionary ‘nspace’: name, ‘rank’: 0. Exploration found no  
26 discernible benefit to either representation, nor was any clearly identifiable rationale developed that  
27 would lead a user to expect one versus the other for a given PMIx data type. Consistency in the  
28 translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion.  
29 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 13 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
<code>bool</code>	<code>PMIX_BOOL</code>	boolean	
<code>byte</code>	<code>PMIX_BYTE</code>	A single element byte array (i.e., a byte array of length one)	
<code>char*</code>	<code>PMIX_STRING</code>	string	
<code>size_t</code>	<code>PMIX_SIZE</code>	integer	
<code>pid_t</code>	<code>PMIX_PID</code>	integer	value shall be limited to the <code>uint32_t</code> range
<code>int, int8_t, int16_t, int32_t, int64_t</code>	<code>PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64</code>	integer	value shall be limited to its corresponding range
<code>uint, uint8_t, uint16_t, uint32_t, uint64_t</code>	<code>PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64</code>	integer	value shall be limited to its corresponding range
<code>float, double</code>	<code>PMIX_FLOAT, PMIX_DOUBLE</code>	float	value shall be limited to its corresponding range
<code>struct timeval</code>	<code>PMIX_TIMEVAL</code>	{'sec': sec, 'usec': microsec}	each field is an integer value
<code>time_t</code>	<code>PMIX_TIME</code>	integer	limited to positive values
<code>pmix_data_type_t</code>	<code>PMIX_DATA_TYPE</code>	integer	value shall be limited to the <code>uint16_t</code> range
<code>pmix_status_t</code>	<code>PMIX_STATUS</code>	integer	
<code>pmix_key_t</code>	N/A	string	The string's length shall be limited to one less than the size of the <code>pmix_key_t</code> array (to reserve space for the terminating <b>NULL</b> )
<code>pmix_nspace_t</code>	N/A	string	The 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 <b>NULL</b> )

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<code>pmix_rank_t</code>	<code>PMIX_PROC_RANK</code>	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>	<code>PMIX_PROC</code>	{'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>	<code>PMIX_BYTE_OBJECT</code>	{'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>	<code>PMIX_PERSISTENCE</code>	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_scope_t</code>	<code>PMIX_SCOPE</code>	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_data_range_t</code>	<code>PMIX_RANGE</code>	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_proc_state_t</code>	<code>PMIX_PROC_STATE</code>	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_proc_info_t</code>	<code>PMIX_PROC_INFO</code>	{'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>	<code>PMIX_DATA_ARRAY</code>	{'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>	<code>PMIX_INFO_DIRECTIVES</code>	list	list of integer values (defined in Section 3.2.10)
<code>pmix_alloc_directive_t</code>	<code>PMIX_ALLOC_DIRECTIVE</code>	integer	value shall be limited to the <code>uint8_t</code> range
<code>pmix_iof_channel_t</code>	<code>PMIX_IOF_CHANNEL</code>	list	list of integer values (defined in Section 18.3.3)
<code>pmix_envar_t</code>	<code>PMIX_ENVAR</code>	{'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>	<code>PMIX_VALUE</code>	{'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>	<code>PMIX_INFO</code>	{'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>	<code>PMIX_PDATA</code>	{'proc': {'nspc': nspc, '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>	<b>PMIX_APP</b>	{'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 <i>lists</i> containing Python strings; <i>maxprocs</i> is an integer; and <i>info</i> is a Python <i>list</i> of <b>info</b> values
<code>pmix_query_t</code>	<b>PMIX_QUERY</b>	{'keys': [keys], 'qualifiers': [info]}	<i>keys</i> is a Python <i>list</i> of Python strings, and <i>qualifiers</i> is a Python <i>list</i> of <b>info</b> values
<code>pmix_regattr_t</code>	<b>PMIX_REGATTR</b>	{'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 <i>list</i> of <b>info</b> values; and <i>description</i> is a list of Python strings describing the attribute
<code>pmix_job_state_t</code>	<b>PMIX_JOB_STATE</b>	integer	value shall be limited to the <b>uint8_t</b> range
<code>pmix_link_state_t</code>	<b>PMIX_LINK_STATE</b>	integer	value shall be limited to the <b>uint8_t</b> range
<code>pmix_cpuset_t</code>	<b>PMIX_PROC_CPUSSET</b>	{'source': source, 'cpus': bitmap}	<i>source</i> is a string name of the library that created the cpuset; and <i>cpus</i> is a list of string ranges identifying the PUs to which the process is bound (e.g., [1, 3-5, 7])
<code>pmix_locality_t</code>	<b>PMIX_LOCTYPE</b>	list	list of integer values (defined in Section 12.4.2.3) describing 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 <b>info</b> describing the fabric
<code>pmix_endpoint_t</code>	<b>PMIX_ENDPOINT</b>	{'uuid': uuid, 'osname': osname, 'endpt': endpt}	<i>uuid</i> is the string system-unique identifier assigned to the device; <i>osname</i> is the operating system name assigned to the device; <i>endpt</i> is a <b>byteobject</b> containing the endpoint information

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
<code>pmix_device_distance_t</code>	<code>PMIX_DEVICE_DIST</code>	{'uuid': uuid, 'osname': osname, 'mindist': mindist, 'maxdist': maxdist}	<i>uuid</i> is the string system-unique identifier assigned to the device; <i>osname</i> is the operating system name assigned to the device; and <i>mindist</i> and <i>maxdist</i> are Python integers
<code>pmix_coord_t</code>	<code>PMIX_COORD</code>	{'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>	<code>PMIX_GEOMETRY</code>	{'fabric': idx, 'uuid': uuid, 'osname': osname, '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; <i>osname</i> is the operating system name assigned to the device; and <i>coordinates</i> is a list of <code>coord</code> containing the coordinates for the device across all views
<code>pmix_device_type_t</code>	<code>PMIX_DEVTYPE</code>	list	list of integer values (defined in Section 12.4.8)
<code>pmix_bind_envelope_t</code>	N/A	integer	one of the values defined in Section 12.4.4.1

## 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:

```
▼ C ───────────────────────────────────────────────────────────────────────────────────────────────────▶
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':PMIX_STRING},
26         {'key':PMIX_MODEL_LIBRARY_NAME,
27         'value':'PMIX', 'val_type':PMIX_STRING}]
28 (rc,myproc) = myclient.init(info)
▶ 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 *PMIx v4.0*

#### Format

Python

```
6 def iofcbfunc(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_iof_cbfunc_t` for details

### 21 A.3.2 Event Handler

#### 22 Summary

23 Callback function for event handlers

#### 24 *PMIx v4.0* Format

## Python

```
1 def evhandler(evhdlr:integer, status:integer,  
2             source:dict, info:list, results:list)
```

## Python

3 **IN** **iofhdlr**

4 Registration number of the handler being invoked (integer)

5 **IN** **status**

6 Status associated with the operation (integer)

7 **IN** **source**

8 Python **proc** identifying the namespace/rank of the process that generated the event (dict)

9 **IN** **info**

10 List of Python **info** provided by the source containing metadata about the event (list)

11 **IN** **results**

12 List of Python **info** containing the aggregated results of all prior evhandlers (list)

13 Returns:

- 14 • **rc** - Status returned by the event handler's operation (integer)
- 15 • **results** - List of Python **info** containing results from this event handler's operation on the event  
16 (list)

17 See [pmix\\_notification\\_fn\\_t](#) for details

### 18 A.3.3 Server Module Functions

19 The following definitions represent functions that may be provided to the PMIx server library at  
20 time of initialization for servicing of client requests. Module functions that are not provided default  
21 to returning "not supported" to the caller.

#### 22 A.3.3.1 Client Connected

##### 23 Summary

24 Notify the host server that a client connected to this server.

##### 25 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**

*PMIx v4.0*

Python

13 `def clientfinalized(proc:dict is not None):`

Python

14 **IN** `proc`

15 Python `proc` identifying the namespace/rank of the process that finalized (dict)

16 Returns: nothing

17 See `pmix_server_client_finalized_fn_t` for details

### 18 **A.3.3.3 Client Aborted**

#### 19 **Summary**

20 Notify the host environment that a local client called `PMIx_Abort`.

1 **Format** Python

2 `def clientaborted(args:dict is not None)` Python

3 **IN args**  
4 Python dictionary containing:

- 5 • 'caller': Python **proc** identifying the namespace/rank of the process calling abort (dict)
- 6 • 'status': PMIx status to be returned on exit (integer)
- 7 • 'msg': Optional string message to be printed (string)
- 8 • 'targets': Optional list of Python **proc** identifying the namespace/rank of the processes to
- 9 be aborted (list)

10 Returns:

- 11 • *rc* - **PMIx\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

12 See [pmix\\_server\\_abort\\_fn\\_t](#) for details

### 13 A.3.3.4 Fence

#### 14 Summary

15 At least one client called either [PMIx\\_Fence](#) or [PMIx\\_Fence\\_nb](#)

16 *PMIx v4.0* **Format** Python

17 `def fence(args:dict is not None)` Python

18 **IN args**  
19 Python dictionary containing:

- 20 • 'procs': List of Python **proc** identifying the namespace/rank of the participating processes
- 21 (list)
- 22 • 'directives': Optional list of Python **info** containing directives controlling the operation
- 23 (list)
- 24 • 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)

25 Returns:

- 26 • *rc* - **PMIx\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

- 27 • *data* - Python bytearray containing the aggregated data from all participants (bytearray)

28 See [pmix\\_server\\_fence\\_nb\\_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**

*PMIx v4.0*

Python

```
6 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**

*PMIx v4.0*

Python

```
20 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**

*PMIx v4.0*

Python

```
5 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**

*PMIx v4.0*

Python

```
19 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**

*PMIx v4.0*

Python

```
5 def spawn(args:dict is not None)
```

Python

#### 6 **IN args**

7 Python dictionary containing:

- 8 • 'proc': Python [proc](#) of process making the request (dict)
- 9 • 'jobinfo': Optional list of Python [info](#) job-level directives and information (list)
- 10 • 'apps': List of Python [app](#) describing applications to be spawned (list)

11 Returns:

- 12 • *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- 13 • *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**

*PMIx v4.0*

Python

```
19 def connect(args:dict is not None)
```

Python

#### 20 **IN args**

21 Python dictionary containing:

- 22 • 'procs': List of Python [proc](#) identifying the namespace/rank of the participating processes (list)
- 23 • 'directives': Optional list of Python [info](#) containing directives controlling the operation (list)

26 Returns:

- 27 • *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

28 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**

*PMIx v4.0*

Python

```
5 def disconnect(args:dict is not None)
```

Python

#### 6 **IN args**

7 Python dictionary containing:

- 8 • 'procs': List of Python **proc** identifying the namespace/rank of the participating processes (list)
- 9 • 'directives': Optional list of Python **info** containing directives controlling the operation (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**

*PMIx v4.0*

Python

```
19 def register_events(args:dict is not None)
```

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 (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 *PMIx v4.0* Format

Python

```
5 def deregister_events(args:dict is not None)
```

Python

#### 6 IN args

7 Python dictionary containing:

- 8 • 'codes': List of Python integers (list)

9 Returns:

- 10 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

11 See [pmix\\_server\\_deregister\\_events\\_fn\\_t](#) for details

### 12 A.3.3.14 Notify Event

#### 13 Summary

14 Notify the specified range of processes of an event.

#### 15 *PMIx v4.0* Format

Python

```
16 def notify_event(args:dict is not None)
```

Python

#### 17 IN args

18 Python dictionary containing:

- 19 • 'code': Python integer [pmix\\_status\\_t](#) (integer)
- 20 • 'source': Python [proc](#) of process that generated the event (dict)
- 21 • 'range': Python [range](#) in which the event is to be reported (integer)
- 22 • 'directives': Optional list of Python [info](#) directives (list)

23 Returns:

- 24 • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

25 See [pmix\\_server\\_notify\\_event\\_fn\\_t](#) for details

### 26 A.3.3.15 Query

#### 27 Summary

28 Query information from the resource manager.

## Format

Python

```
def query(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'queries': List of Python **query** directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *info* - List of Python **info** containing the returned results (list)

See [pmix\\_server\\_query\\_fn\\_t](#) for details

## A.3.3.16 Tool Connected

### Summary

Register that a tool has connected to the server.

## Format

Python

```
def tool_connected(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'directives': Optional list of Python **info** info on the connecting tool (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *proc* - Python **proc** containing the assigned namespace:rank for the tool (dict)

See [pmix\\_server\\_tool\\_connection\\_fn\\_t](#) for details

## A.3.3.17 Log

### Summary

Log data on behalf of a client.



## Format

Python

```
def log(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'data': Optional list of Python **info** containing data to be logged (list)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See [pmix\\_server\\_log\\_fn\\_t](#) for details.

### A.3.3.18 Allocate Resources

#### Summary

Request allocation operations on behalf of a client.

## Format

Python

*PMIx v4.0*

```
def allocate(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'action': Python **allocdir** specifying requested action (integer)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *refarginfo* - List of Python **info** containing results of requested operation (list)

See [pmix\\_server\\_alloc\\_fn\\_t](#) for details.

### A.3.3.19 Job Control

#### Summary

Execute a job control action on behalf of a client.

## Format

Python

```
def job_control(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'targets': List of Python **proc** specifying target processes (list)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See [pmix\\_server\\_job\\_control\\_fn\\_t](#) for details.

## A.3.3.20 Monitor

### Summary

Request that a client be monitored for activity.

## Format

Python

```
def monitor(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'monitor': Python **info** attribute indicating the type of monitor being requested (dict)
- 'error': Status code to be used when generating an event notification (integer) alerting that the monitor has been triggered.
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See [pmix\\_server\\_monitor\\_fn\\_t](#) for details.

## A.3.3.21 Get Credential

### Summary

Request a credential from the host environment.

## Format

Python

```
def get_credential(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *cred* - Python **byteobject** containing returned credential (dict)
- *info* - List of Python **info** containing any additional info about the credential (list)

See [pmix\\_server\\_get\\_cred\\_fn\\_t](#) for details.

## A.3.3.22 Validate Credential

### Summary

Request validation of a credential

## Format

Python

```
def validate_credential(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'credential': Python **byteobject** containing credential (dict)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *info* - List of Python **info** containing any additional info from the credential (list)

See [pmix\\_server\\_validate\\_cred\\_fn\\_t](#) for details.

## A.3.3.23 IO Forward

### Summary

Request the specified IO channels be forwarded from the given array of processes.

## Format

Python

```
def iof_pull(args:dict is not None)
```

Python

### IN args

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)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See [pmix\\_server\\_iof\\_fn\\_t](#) for details.

### A.3.3.24 IO Push

#### Summary

Pass standard input data to the host environment for transmission to specified recipients.

## Format

Python

```
def iof_push(args:dict is not None)
```

Python

### IN args

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)

Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See [pmix\\_server\\_stdin\\_fn\\_t](#) for details.

### A.3.3.25 Group Operations

#### Summary

Request group operations (construct, destruct, etc.) on behalf of a set of processes.

## Format

Python

```
def group(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'op': Operation host is to perform on the specified group (integer)
- 'group': String identifier of target group (str)
- 'procs': List of Python **proc** of participating processes (dict)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- rc - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python **info** containing results of requested operation (list)

See [pmix\\_server\\_grp\\_fn\\_t](#) for details.

## A.3.3.26 Fabric Operations

### Summary

Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

## Format

Python

```
def fabric(args:dict is not None)
```

Python

### IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'index': Identifier of the fabric being operated upon (integer)
- 'op': Operation host is to perform on the specified fabric (integer)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- rc - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python **info** containing results of requested operation (list)

See [pmix\\_server\\_fabric\\_fn\\_t](#) for details.

## 1 A.4 PMIxClient

2 The client Python class is by far the richest in terms of APIs as it houses all the APIs that an  
3 application might utilize. Due to the datatype translation requirements of the C-Python interface,  
4 only the blocking form of each API is supported – providing a Python callback function directly to  
5 the C interface underlying the bindings was not a supportable option.

### 6 A.4.1 Client.init

#### 7 Summary

8 Initialize the PMIx client library after obtaining a new PMIxClient object.

#### 9 *PMIx v4.0* Format

Python

```
10 rc, proc = myclient.init(info:list)
```

Python

#### 11 IN info

12 List of Python **info** dictionaries (list)

13 Returns:

- 14 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 15 • *proc* - a Python **proc** dictionary (dict)

16 See **PMIx\_Init** for description of all relevant attributes and behaviors.

### 17 A.4.2 Client.initialized

#### 18 *PMIx v4.0* Format

Python

```
19 rc = myclient.initialized()
```

Python

20 Returns:

- 21 • *rc* - a value of **1** (true) will be returned if the PMIx library has been initialized, and **0** (false)  
22 otherwise (integer)

23 See **PMIx\_Initialized** for description of all relevant attributes and behaviors.

### 1 A.4.3 Client.get\_version

#### 2 Format

Python

3 `vers = myclient.get_version()`

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

Python

*PMIx v4.0*

11 `rc = myclient.finalize(info:list)`

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.

## Format

Python

```
rc = myclient.abort(status:integer, msg:str, targets:list)
```

Python

### IN status

PMIx status to be returned on exit (integer)

### IN msg

String message to be printed (string)

### IN targets

List of Python `proc` dictionaries (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_Abort` for description of all relevant attributes and behaviors.

## A.4.6 Client.store\_internal

### Summary

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

## Format

Python

```
rc = myclient.store_internal(proc:dict, key:str, value:dict)
```

Python

### IN proc

Python `proc` dictionary of the process being referenced (dict)

### IN key

String key of the data (string)

### IN value

Python `value` dictionary (dict)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_Store_internal` for details.

## A.4.7 Client.put

### Summary

Push a key/value pair into the client's namespace.



1

### Format

Python

2

```
rc = myclient.put(scope:integer, key:str, value:dict)
```

Python

3

**IN** `scope`

Scope of the data being posted (integer)

4

**IN** `key`

String key of the data (string)

5

6

**IN** `value`

Python `value` dictionary (dict)

7

8

9

Returns:

10

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

11

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

*PMIx v4.0*

### Format

Python

16

```
rc = myclient.commit()
```

Python

17

Returns:

18

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

19

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.

## Format

Python

```
rc = myclient.fence(peers:list, directives:list)
```

Python

### IN peers

List of Python **proc** dictionaries (list)

### IN directives

List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx\_Fence** for description of all relevant attributes and behaviors.

## A.4.10 Client.get

### Summary

Retrieve a key/value pair.

## Format

Python

*PMIx v4.0*

```
rc, val = myclient.get(proc:dict, key:str, directives:list)
```

Python

### IN proc

Python **proc** whose data is being requested (dict)

### IN key

Python string key of the data to be returned (str)

### IN directives

List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *val* - Python **value** containing the returned data (dict)

See **PMIx\_Get** for description of all relevant attributes and behaviors.

## A.4.11 Client.publish

### Summary

Publish data for later access via **PMIx\_Lookup**.

1 **Format** Python

2 `rc = myclient.publish(directives:list)`  
3 Python

4 **IN directives**  
5 List of Python **info** dictionaries containing data to be published and directives (list)

6 Returns:  
7 

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

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 *PMIx v4.0* **Format** Python

12 `rc,info = myclient.lookup(pdata:list, directives:list)`  
13 Python

14 **IN pdata**  
15 List of Python **pdata** dictionaries identifying data to be retrieved (list)

16 **IN directives**  
17 List of Python **info** dictionaries (list)

18 Returns:  
19 

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - Python list of **info** containing the returned data (list)

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  
2  
3  
4  
5  
6  
7  
8  
9

**Format**

Python

```
rc = myclient.unpublish(keys:list, directives:list)
```

Python

**IN keys**

List of Python string keys identifying data to be deleted (list)

**IN directives**

List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- See **PMIx\_Unpublish** for description of all relevant attributes and behaviors.

**A.4.14 Client.spawn**

**Summary**

Spawn a new job.

13 *PMIx v4.0*

**Format**

Python

```
rc, nspace = myclient.spawn(jobinfo:list, apps:list)
```

Python

**IN jobinfo**

List of Python **info** dictionaries (list)

**IN apps**

List of Python **app** dictionaries (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *nspace* - Python **nspace** of the new job (dict)

See **PMIx\_Spawn** for description of all relevant attributes and behaviors.

**A.4.15 Client.connect**

**Summary**

Connect namespaces.

24  
25

1 **Format** Python

2 `rc = myclient.connect(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\_Connect** for description of all relevant attributes and behaviors.

### 10 A.4.16 Client.disconnect

11 **Summary**  
12 Disconnect namespaces.

13 *PMIx v4.0* **Format** Python

14 `rc = myclient.disconnect(peers:list, directives:list)` Python

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 **nspace** on the given node.

## Format

Python

```
rc,procs = myclient.resolve_peers(node:str, nspace:str)
```

Python

### IN node

Name of node whose processes are being requested (str)

### IN nspace

Python **nspace** whose processes are to be returned (str)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *procs* - List of Python **proc** dictionaries (list)

See **PMIx\_Resolve\_peers** for description of all relevant attributes and behaviors.

## A.4.18 Client.resolve\_nodes

### Summary

Return list of nodes hosting processes within the specified **nspace**.

## Format

Python

```
rc,nodes = myclient.resolve_nodes(nspace:str)
```

Python

### IN nspace

Python **nspace** (str)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *nodes* - List of Python string node names (list)

See **PMIx\_Resolve\_nodes** for description of all relevant attributes and behaviors.

## A.4.19 Client.query

### Summary

Query information about the system in general.

1 **Format** Python

2 `rc, info = myclient.query(queries:list)`  
3 Python

4 **IN queries**  
5 List of Python [query](#) dictionaries (list)

6 Returns:

- 7 • *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- 8 • *info* - List of Python [info](#) containing results of the query (list)

9 See [PMIx\\_Query\\_info](#) 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 *PMIx v4.0* **Format** Python

13 `rc = myclient.log(data:list, directives:list)`  
14 Python

15 **IN data**  
16 List of Python [info](#) (list)

17 **IN directives**  
18 Optional list of Python [info](#) (list)

19 Returns:

- 20 • *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

21 See [PMIx\\_Log](#) for description of all relevant attributes and behaviors.

## 21 **A.4.21 Client.allocation\_request**

22 **Summary**  
23 Request an allocation operation from the host resource manager.

## Format

Python

```
rc, info = myclient.allocation_request(request:integer, directives:list)
```

Python

### IN request

Python [allocdir](#) specifying requested operation (integer)

### IN directives

List of Python [info](#) describing request (list)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python [info](#) containing results of the request (list)

See [PMIx\\_Allocation\\_request](#) for description of all relevant attributes and behaviors.

## A.4.22 Client.job\_ctrl

### Summary

Request a job control action.

## Format

Python

```
rc, info = myclient.job_ctrl(targets:list, directives:list)
```

Python

### IN targets

List of Python [proc](#) specifying targets of requested operation (integer)

### IN directives

List of Python [info](#) describing operation to be performed (list)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python [info](#) containing results of the request (list)

See [PMIx\\_Job\\_control](#) for description of all relevant attributes and behaviors.

## A.4.23 Client.monitor

### Summary

Request that something be monitored.



## Format

Python

```
rc, info = myclient.monitor(monitor:dict, error_code:integer, directives:list)
```

Python

### IN monitor

Python **info** specifying specifying the type of monitor being requested (dict)

### IN error\_code

Status code to be used when generating an event notification alerting that the monitor has been triggered (integer)

### IN directives

List of Python **info** describing request (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **info** containing results of the request (list)

See **PMIx\_Process\_monitor** for description of all relevant attributes and behaviors.

## A.4.24 Client.get\_credential

### Summary

Request a credential from the PMIx server/SMS.

## Format

Python

*PMIx v4.0*

```
rc, cred = myclient.get_credential(directives:list)
```

Python

### IN directives

Optional list of Python **info** describing request (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *cred* - Python **byteobject** containing returned credential (dict)

See **PMIx\_Get\_credential** for description of all relevant attributes and behaviors.

## A.4.25 Client.validate\_credential

### Summary

Request validation of a credential by the PMIx server/SMS.

## Format

Python

```
rc, info = myclient.validate_credential(cred:dict, directives:list)
```

Python

### IN cred

Python **byteobject** containing credential (dict)

### IN directives

Optional list of Python **info** describing request (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **info** containing additional results of the request (list)

See **PMIx\_Validate\_credential** for description of all relevant attributes and behaviors.

## A.4.26 Client.group\_construct

### Summary

Construct a new group composed of the specified processes and identified with the provided group identifier.

## Format

Python

```
rc, info = myclient.construct_group(grp:string,  
                                   members:list, directives:list)
```

Python

### IN grp

Python string identifier for the group (str)

### IN members

List of Python **proc** dictionaries identifying group members (list)

### IN directives

Optional list of Python **info** describing request (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **info** containing results of the request (list)

See **PMIx\_Group\_construct** for description of all relevant attributes and behaviors.

## A.4.27 Client.group\_invite

### Summary

Explicitly invite specified processes to join a group.

## Format

Python

```
rc, info = myclient.group_invite(grp:string,  
                                members:list, directives:list)
```

Python

### IN grp

Python string identifier for the group (str)

### IN members

List of Python [proc](#) dictionaries identifying processes to be invited (list)

### IN directives

Optional list of Python [info](#) describing request (list)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python [info](#) containing results of the request (list)

See [PMIx\\_Group\\_invite](#) for description of all relevant attributes and behaviors.

## A.4.28 Client.group\_join

### Summary

Respond to an invitation to join a group that is being asynchronously constructed.

## Format

Python

```
rc, info = myclient.group_join(grp:string,  
                               leader:dict, opt:integer,  
                               directives:list)
```

Python

### IN grp

Python string identifier for the group (str)

### IN leader

Python [proc](#) dictionary identifying process leading the group (dict)

### IN opt

One of the [pmix\\_group\\_opt\\_t](#) values indicating decline/accept (integer)

### IN directives

Optional list of Python [info](#) describing request (list)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python [info](#) containing results of the request (list)

See [PMIx\\_Group\\_join](#) for description of all relevant attributes and behaviors.

## 1 A.4.29 Client.group\_leave

### 2 Summary

3 Leave a PMIx Group.

### 4 *PMIx v4.0* Format

Python

5 `rc = myclient.group_leave(grp:string, directives:list)`

Python

6 **IN** `grp`

7 Python string identifier for the group (str)

8 **IN** `directives`

9 Optional list of Python [info](#) describing request (list)

10 Returns:

- 11 • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

12 See [PMIx\\_Group\\_leave](#) for description of all relevant attributes and behaviors.

## 13 A.4.30 Client.group\_destruct

### 14 Summary

15 Destruct a PMIx Group.

### 16 *PMIx v4.0* Format

Python

17 `rc = myclient.group_destruct(grp:string, directives:list)`

Python

18 **IN** `grp`

19 Python string identifier for the group (str)

20 **IN** `directives`

21 Optional list of Python [info](#) describing request (list)

22 Returns:

- 23 • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

24 See [PMIx\\_Group\\_destruct](#) for description of all relevant attributes and behaviors.

## 25 A.4.31 Client.register\_event\_handler

### 26 Summary

27 Register an event handler to report events.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26

## Format

Python

```
rc, id = myclient.register_event_handler(codes:list,  
                                       directives:list, cbfunc)
```

Python

### IN codes

List of Python integer status codes that should be reported to this handler (l1ist)

### IN directives

Optional list of Python [info](#) describing request (list)

### IN cbfunc

Python [evhandler](#) to be called when event is received (func)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *id* - PMIx reference identifier for handler (integer)

See [PMIx\\_Register\\_event\\_handler](#) for description of all relevant attributes and behaviors.

## A.4.32 Client.deregister\_event\_handler

### Summary

Deregister an event handler.

## Format

Python

```
myclient.deregister_event_handler(id:integer)
```

Python

### IN id

PMIx reference identifier for handler (integer)

Returns: None

See [PMIx\\_Deregister\\_event\\_handler](#) for description of all relevant attributes and behaviors.

## A.4.33 Client.notify\_event

### Summary

Report an event for notification via any registered handler.

## Format

Python

```
rc = myclient.notify_event(status:integer, source:dict,  
                           range:integer, directives:list)
```

Python

### IN status

PMIx status code indicating the event being reported (integer)

### IN source

Python **proc** of the process that generated the event (dict)

### IN range

Python **range** in which the event is to be reported (integer)

### IN directives

Optional list of Python **info** dictionaries describing the event (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx\_Notify\_event** for description of all relevant attributes and behaviors.

## A.4.34 Client.fabric\_register

### Summary

Register for access to fabric-related information, including communication cost matrix.

## Format

Python

```
rc, idx, fabricinfo = myclient.fabric_register(directives:list)
```

Python

### IN directives

Optional list of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *idx* - Index of the registered fabric (integer)
- *fabricinfo* - List of Python **info** containing fabric info (list)

See **PMIx\_Fabric\_register** for details.

## A.4.35 Client.fabric\_update

### Summary

Update fabric-related information, including communication cost matrix.

1 **Format** Python

```
2 rc, fabricinfo = myclient.fabric_update(idx:integer)
```

3 **IN** `idx`  
4 Index of the registered fabric (list)

5 Returns:  
6 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
7 • `fabricinfo` - List of Python **info** containing updated fabric info (list)  
8 See **PMIx\_Fabric\_update** for details.

### 9 **A.4.36 Client.fabric\_deregister**

10 **Summary**  
11 Deregister fabric.

12 *PMIx v4.0* **Format** Python

```
13 rc = myclient.fabric_deregister(idx:integer)
```

14 **IN** `idx`  
15 Index of the registered fabric (list)

16 Returns:  
17 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
18 See **PMIx\_Fabric\_deregister** for details.

### 19 **A.4.37 Client.load\_topology**

20 **Summary**  
21 Load the local hardware topology into the PMIx library.

22 *PMIx v4.0* **Format** Python

```
23 rc = myclient.load_topology()
```

24 Returns:  
25 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
26 See **PMIx\_Load\_topology** for details - note that the topology loaded into the PMIx library may  
27 be utilized by PMIx and other libraries, but is not directly accessible by Python.

## 1 A.4.38 Client.get\_relative\_locality

### 2 Summary

3 Get the relative locality of two local processes.

### 4 *PMIx v4.0* Format

Python

5 `rc, locality = myclient.get_relative_locality(loc1:str, loc2:str)`

Python

6 **IN** `loc1`

7 Locality string of a process (str)

8 **IN** `loc2`

9 Locality string of a process (str)

10 Returns:

- 11 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 12 • `locality` - **locality** list containing the relative locality of the two processes (list)

13 See **PMIx\_Get\_relative\_locality** for details.

## 14 A.4.39 Client.get\_cpuset

### 15 Summary

16 Get the PU binding bitmap of the current process.

### 17 *PMIx v4.0* Format

Python

18 `rc, cpuset = myclient.get_cpuset(ref:integer)`

Python

19 **IN** `ref`

20 **bindenv** binding envelope to be used (integer)

21 Returns:

- 22 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 23 • `cpuset` - **cpuset** containing the source and bitmap of the cpuset (dict)

24 See **PMIx\_Get\_cpuset** for details.

## 25 A.4.40 Client.parse\_cpuset\_string

### 26 Summary

27 Parse the PU binding bitmap from its string representation.



## Format

Python

```
rc, cpuset = myclient.parse_cpuset_string(cpuset:string)
```

Python

**IN** `cpuset`

String returned by `PMIxServer.generate_cpuset_string` (string)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `cpuset` - `cpuset` containing the source and bitmap of the cpuset (dict)

See `PMIx_Parse_cpuset_string` for details.

### A.4.41 Client.compute\_distances

#### Summary

Compute distances from specified process location to local devices.

## Format

Python

```
rc, distances = myclient.compute_distances(cpuset:dict, info:list)
```

Python

**IN** `cpuset`

`cpuset` describing the location of the process (dict)

**IN** `info`

List of `info` dictionaries describing the devices whose distance is to be computed (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `distances` - List of `devdist` structures containing the distances from the caller to the specified devices (list)

See `PMIx_Compute_distances` for details. Note that distances can only be computed against the local topology.

### A.4.42 Client.error\_string

#### Summary

Pretty-print string representation of `pmix_status_t`.

1 **Format** Python

2 `rep = myclient.error_string(status:integer)` Python

3 **IN status**  
4 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.4.43 Client.proc\_state\_string

9 **Summary**  
10 Pretty-print string representation of [pmix\\_proc\\_state\\_t](#).

11 *PMIx v4.0* **Format** Python  
12 `rep = myclient.proc_state_string(state:integer)` 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.4.44 Client.scope\_string

19 **Summary**  
20 Pretty-print string representation of [pmix\\_scope\\_t](#).

21 *PMIx v4.0* **Format** Python  
22 `rep = myclient.scope_string(scope:integer)` 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.4.45 Client.persistence\_string

### 2 Summary

3 Pretty-print string representation of [pmix\\_persistence\\_t](#).

### 4 *PMIx v4.0* Format

Python

5 `rep = myclient.persistence_string(persistence:integer)`

Python

### 6 IN persistence

7 PMIx persistence value (integer)

8 Returns:

- 9 • *rep* - String representation of the provided persistence (str)

10 See [PMIx\\_Persistence\\_string](#) for further details.

## 11 A.4.46 Client.data\_range\_string

### 12 Summary

13 Pretty-print string representation of [pmix\\_data\\_range\\_t](#).

### 14 *PMIx v4.0* Format

Python

15 `rep = myclient.data_range_string(range:integer)`

Python

### 16 IN range

17 PMIx data range value (integer)

18 Returns:

- 19 • *rep* - String representation of the provided data range (str)

20 See [PMIx\\_Data\\_range\\_string](#) for further details.

## 21 A.4.47 Client.info\_directives\_string

### 22 Summary

23 Pretty-print string representation of [pmix\\_info\\_directives\\_t](#).

1 **Format** Python

2 `rep = myclient.info_directives_string(directives:bitarray)`  
3 Python

3 **IN directives**  
4 PMIx **info directives** value (bitarray)

5 Returns:

- 6 • *rep* - String representation of the provided info directives (str)  
7 See [PMIx\\_Info\\_directives\\_string](#) for further details.

## 8 **A.4.48 Client.data\_type\_string**

### 9 **Summary**

10 Pretty-print string representation of [pmix\\_data\\_type\\_t](#).

11 *PMIx v4.0* **Format** Python

12 `rep = myclient.data_type_string(dtype:integer)`  
13 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.

## 18 **A.4.49 Client.alloc\_directive\_string**

### 19 **Summary**

20 Pretty-print string representation of [pmix\\_alloc\\_directive\\_t](#).

21 *PMIx v4.0* **Format** Python

22 `rep = myclient.alloc_directive_string(adir:integer)`  
23 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.4.50 Client.iof\_channel\_string**

### 2 **Summary**

3 Pretty-print string representation of [pmix\\_iof\\_channel\\_t](#).

### 4 **Format**

*PMIx v4.0*

Python

5 `rep = myclient.iof_channel_string(channel:bitarray)`

Python

6 **IN** `channel`

7 `PMIx IOF channel value (bitarray)`

8 Returns:

- 9 • `rep` - String representation of the provided IOF channel (str)

10 See [PMIx\\_IOF\\_channel\\_string](#) for further details.

## 11 **A.4.51 Client.job\_state\_string**

### 12 **Summary**

13 Pretty-print string representation of [pmix\\_job\\_state\\_t](#).

### 14 **Format**

*PMIx v4.0*

Python

15 `rep = myclient.job_state_string(state:integer)`

Python

16 **IN** `state`

17 `PMIx job state value (integer)`

18 Returns:

- 19 • `rep` - String representation of the provided job state (str)

20 See [PMIx\\_Job\\_state\\_string](#) for further details.

## 21 **A.4.52 Client.get\_attribute\_string**

### 22 **Summary**

23 Pretty-print string representation of a `PMIx` attribute.

## Format

Python

```
rep = myclient.get_attribute_string(attribute:str)
```

Python

### IN attribute

PMIx attribute name (string)

Returns:

- *rep* - String representation of the provided attribute (str)
- See [PMIx\\_Get\\_attribute\\_string](#) for further details.

## A.4.53 Client.get\_attribute\_name

### Summary

Pretty-print name of a PMIx attribute corresponding to the provided string.

## Format

Python

*PMIx v4.0*

```
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.4.54 Client.link\_state\_string

### Summary

Pretty-print string representation of [pmix\\_link\\_state\\_t](#).

## Format

Python

*PMIx v4.0*

```
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.4.55 Client.device\_type\_string**

### 2 **Summary**

3 Pretty-print string representation of [pmix\\_device\\_type\\_t](#).

### 4 *PMIx v4.0* **Format**

Python

5 `rep = myclient.device_type_string(type:bitarray)`

Python

### 6 **IN type**

7 PMIx device type value (bitarray)

8 Returns:

- 9 • *rep* - String representation of the provided device type (str)

10 See [PMIx\\_Device\\_type\\_string](#) for further details.

## 11 **A.4.56 Client.progress**

### 12 **Summary**

13 Progress the PMIx library.

### 14 *PMIx v4.0* **Format**

Python

15 `myclient.progress()`

Python

16 See [PMIx\\_Progress](#) for further details.

## 17 **A.5 PMIxServer**

18 The server Python class inherits the Python "client" class as its parent. Thus, it includes all client  
19 functions in addition to the ones defined in this section.

### 20 **A.5.1 Server.init**

#### 21 **Summary**

22 Initialize the PMIx server library after obtaining a new PMIxServer object.

1

## Format

Python

2

```
rc = myserver.init(directives:list, map:dict)
```

Python

3

### IN directives

List of Python [info](#) dictionaries (list)

4

5

### IN map

Python dictionary key-function pairs that map [server module](#) callback functions to provided implementations (see [pmix\\_server\\_module\\_t](#)) (dict)

6

7

8

Returns:

9

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

10

See [PMIx\\_server\\_init](#) for description of all relevant attributes and behaviors.

## 11 A.5.2 Server.finalize

12

### Summary

Finalize the PMIx server library.

13

14

*PMIx v4.0*

### Format

Python

15

```
rc = myserver.finalize()
```

Python

16

Returns:

17

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

18

See [PMIx\\_server\\_finalize](#) for details.

## 19 A.5.3 Server.generate\_regex

20

### Summary

Generate a regular expression representation of the input strings.

21



## Format

Python

```
rc, regex = myserver.generate_regex(input:list)
```

Python

### IN input

List of Python strings (e.g., node names) (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *regex* - Python **bytearray** containing regular expression representation of the input list (**bytearray**)

See [PMIx\\_generate\\_regex](#) for details.

## A.5.4 Server.generate\_ppn

### Summary

Generate a regular expression representation of the input strings.

## Format

Python

```
rc, regex = myserver.generate_ppn(input:list)
```

Python

### IN input

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)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *regex* - Python **bytearray** containing regular expression representation of the input list (**bytearray**)

See [PMIx\\_generate\\_ppn](#) for details.

## A.5.5 Server.generate\_locality\_string

### Summary

Generate a PMIx locality string from a given cpuset.

## Format

Python

```
rc, locality = myserver.generate_locality_string(cpuset:dict)
```

Python

### IN cset

**cpuset** containing the bitmap of assigned PUs (dict)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *locality* - String representation of the PMIx locality corresponding to the input bitmap (string)

See [PMIx\\_server\\_generate\\_locality\\_string](#) for details.

## A.5.6 Server.generate\_cpuset\_string

### Summary

Generate a PMIx string representation of the provided cpuset.

## Format

Python

```
rc, cpustr = myserver.generate_cpuset_string(cpuset:dict)
```

Python

### IN cset

**cpuset** containing the bitmap of assigned PUs (dict)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *cpustr* - String representation of the input bitmap (string)

See [PMIx\\_server\\_generate\\_cpuset\\_string](#) for details.

## A.5.7 Server.register\_namespace

### Summary

Setup the data about a particular namespace.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

### Format

Python

```
rc = myserver.register_namespace(namespace: str,  
                                nlocalprocs: integer,  
                                directives: list)
```

Python

#### IN namespace

Python string containing the namespace (str)

#### IN nlocalprocs

Number of local processes (integer)

#### IN directives

List of Python [info](#) dictionaries (list)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- See [PMIx\\_server\\_register\\_namespace](#) for description of all relevant attributes and behaviors.

## A.5.8 Server.deregister\_namespace

### Summary

Deregister a namespace.

### Format

Python

*PMIx v4.0*

```
myserver.deregister_namespace(namespace: str)
```

Python

#### IN namespace

Python string containing the namespace (str)

Returns: None

See [PMIx\\_server\\_deregister\\_namespace](#) for details.

## A.5.9 Server.register\_resources

### Summary

Register non-namespace related information with the local PMIx library

1 **Format** Python

2 `myserver.register_resources(directives:list)` Python

3 **IN directives**  
4 List of Python **info** dictionaries (list)

5 Returns: None

6 See [PMIx\\_server\\_register\\_resources](#) for details.

### 7 **A.5.10 Server.deregister\_resources**

8 **Summary**  
9 Remove non-namespace related information from the local PMIx library

10 *PMIx v4.0* **Format** Python

11 `myserver.deregister_resources(directives:list)` Python

12 **IN directives**  
13 List of Python **info** dictionaries (list)

14 Returns: None

15 See [PMIx\\_server\\_deregister\\_resources](#) for details.

### 16 **A.5.11 Server.register\_client**

17 **Summary**  
18 Register a client process with the PMIx server library.

19 *PMIx v4.0* **Format** Python

20 `rc = myserver.register_client(proc:dict, uid:integer, gid:integer)` Python

21 **IN proc**  
22 Python **proc** dictionary identifying the client process (dict)

23 **IN uid**  
24 Linux uid value for user executing client process (integer)

25 **IN gid**  
26 Linux gid value for user executing client process (integer)

27 Returns:

- 28 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

29 See [PMIx\\_server\\_register\\_client](#) for details.

## 1 A.5.12 Server.deregister\_client

### 2 Summary

3 Deregister a client process and purge all data relating to it.

### 4 *PMIx v4.0* Format

Python

5 `myserver.deregister_client(proc:dict)`

Python

6 **IN** `proc`

7 Python `proc` dictionary identifying the client process (dict)

8 Returns: None

9 See [PMIx\\_server\\_deregister\\_client](#) for details.

## 10 A.5.13 Server.setup\_fork

### 11 Summary

12 Setup the environment of a child process that is to be forked by the host.

### 13 *PMIx v4.0* Format

Python

14 `rc = myserver.setup_fork(proc:dict, environ:dict)`

Python

15 **IN** `proc`

16 Python `proc` dictionary identifying the client process (dict)

17 **INOUT** `environ`

18 Python dictionary containing the environment to be passed to the client (dict)

19 Returns:

- 20 • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

21 See [PMIx\\_server\\_setup\\_fork](#) for details.

## 22 A.5.14 Server.dmodex\_request

### 23 Summary

24 Function by which the host server can request modex data from the local PMIx server.

## Format

Python

```
rc, data = myserver.dmodex_request(proc:dict)
```

Python

### IN `proc`

Python `proc` dictionary identifying the process whose data is requested (dict)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `data` - Python `byteobject` containing the returned data (dict)

See `PMIx_server_dmodex_request` for details.

## A.5.15 `Server.setup_application`

### Summary

Function by which the resource manager can request application-specific setup data prior to launch of a *job*.

## Format

Python

```
rc, info = myserver.setup_application(nspace:str, directives:list)
```

Python

### IN `nspace`

Namespace whose setup information is being requested (str)

### IN `directives`

Python list of `info` directives

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `info` - Python list of `info` dictionaries containing the returned data (list)

See `PMIx_server_setup_application` for details.

## A.5.16 `Server.register_attributes`

### Summary

Register host environment attribute support for a function.

1 **Format** Python

2 `rc = myserver.register_attributes(function:str, attrs:list)`  
3 Python

4 **IN function**  
5 Name of the function (str)

6 **IN attrs**  
7 Python list of [regattr](#) describing the supported attributes

8 Returns:

- 9 • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- 10 See [PMIx\\_Register\\_attributes](#) for details.

## 10 A.5.17 Server.setup\_local\_support

### 11 Summary

12 Function by which the local PMIx server can perform any application-specific operations prior to  
13 spawning local clients of a given application.

14 *PMIx v4.0* **Format** Python

15 `rc = myserver.setup_local_support(nspace:str, info:list)`  
16 Python

17 **IN nspace**  
18 Namespace whose setup information is being requested (str)

19 **IN info**  
20 Python list of [info](#) containing the setup data (list)

21 Returns:

- 22 • `rc` - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- 23 See [PMIx\\_server\\_setup\\_local\\_support](#) for details.

## 23 A.5.18 Server.iof\_deliver

### 24 Summary

25 Function by which the host environment can pass forwarded IO to the PMIx server library for  
26 distribution to its clients.

## Format

Python

```
rc = myserver.iof_deliver(source:dict, channel:integer,  
                          data:dict, directives:list)
```

Python

### IN source

Python **proc** dictionary identifying the process who generated the data (dict)

### IN channel

Python **channel** bitmask identifying IO channel of the provided data (integer)

### IN data

Python **byteobject** containing the data (dict)

### IN directives

Python list of **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIX error constant (integer)

See [PMIX\\_server\\_IOF\\_deliver](#) for details.

## A.5.19 Server.collect\_inventory

### Summary

Collect inventory of resources on a node.

## Format

Python

```
rc,info = myserver.collect_inventory(directives:list)
```

Python

### IN directives

Optional Python list of **info** containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIX error constant (integer)
- *info* - Python list of **info** containing the returned data (list)

See [PMIX\\_server\\_collect\\_inventory](#) for details.

## A.5.20 Server.deliver\_inventory

### Summary

Pass collected inventory to the PMIX server library for storage.



1  
2  
3  
4  
5  
6  
7  
8  
9

**Format**

Python

```
rc = myserver.deliver_inventory(info:list, directives:list)
```

Python

**IN info**

- Python list of **info** dictionaries containing the inventory data (list)

**IN directives**

Python list of **info** dictionaries containing directives (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See [PMIx\\_server\\_deliver\\_inventory](#) for details.

10 **A.5.21 Server.define\_process\_set**

11 **Summary**

12 Add members to a PMIx process set.

13 *PMIx v4.0*

**Format**

Python

```
rc = myserver.define_process_set(members:list, name:str)
```

Python

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:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

22 See [PMIx\\_server\\_define\\_process\\_set](#) for details.

23 **A.5.22 Server.delete\_process\_set**

24 **Summary**

25 Delete a PMIx process set.

1 **Format** Python

2 `rc = myserver.delete_process_set(name:str)`  
3 Python

4 **IN** `name`  
5 - Name of the process set (str)

6 Returns:  
7 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
8 See [PMIx\\_server\\_delete\\_process\\_set](#) for details.

### 8 **A.5.23 Server.register\_resources**

9 **Summary**  
10 Register non-namespace related information with the local PMIx server library.

11 *PMIx v4.0* **Format** Python

12 `rc = myserver.register_resources(info:list)`  
13 Python

14 **IN** `info`  
15 - List of Python `info` dictionaries list)

16 Returns:  
17 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
18 See [PMIx\\_server\\_register\\_resources](#) for details.

### 18 **A.5.24 Server.deregister\_resources**

19 **Summary**  
20 Deregister non-namespace related information with the local PMIx server library.

21 *PMIx v4.0* **Format** Python

22 `rc = myserver.deregister_resources(info:list)`  
23 Python

24 **IN** `info`  
25 - List of Python `info` dictionaries list)

26 Returns:  
27 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
28 See [PMIx\\_server\\_deregister\\_resources](#) for details.

## 1 A.6 PMIxTool

2 The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and  
3 server functions in addition to the ones defined in this section.

### 4 A.6.1 Tool.init

#### 5 Summary

6 Initialize the PMIx tool library after obtaining a new PMIxTool object.

#### 7 *PMIx v4.0* Format

Python

8 `rc,proc = mytool.init(info:list)`

Python

#### 9 IN info

10 List of Python `info` directives (list)

11 Returns:

- 12 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- 13 • `proc` - a Python `proc` (dict)

14 See `PMIx_tool_init` for description of all relevant attributes and behaviors.

### 15 A.6.2 Tool.finalize

#### 16 Summary

17 Finalize the PMIx tool library, closing the connection to the server.

#### 18 *PMIx v4.0* Format

Python

19 `rc = mytool.finalize()`

Python

20 Returns:

- 21 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

22 See `PMIx_tool_finalize` for description of all relevant attributes and behaviors.

### 23 A.6.3 Tool.disconnect

#### 24 Summary

25 Disconnect the PMIx tool from the specified server connection while leaving the tool library  
26 initialized.

1  
2  
3  
4  
5  
6  
7

**Format**

Python

```
rc = mytool.disconnect(server:dict)
```

Python

**IN server**

Process identifier of server from which the tool is to be disconnected (**proc**)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- See **PMIx\_tool\_disconnect** for details.

8 **A.6.4 Tool.attach\_to\_server**

9  
10

**Summary**

Establish a connection to a PMIx server.

11 *PMIx v4.0*

**Format**

Python

```
rc,proc,server = mytool.connect_to_server(info:list)
```

Python

**IN info**

List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *proc* - a Python **proc** containing the tool's identifier (dict)
- *server* - a Python **proc** containing the identifier of the server to which the tool attached (dict)

See **PMIx\_tool\_attach\_to\_server** for details.

20 **A.6.5 Tool.get\_servers**

21  
22  
23

**Summary**

Get a list containing the **proc** process identifiers of all servers to which the tool is currently connected.

1 **Format** Python

```
2 rc, servers = mytool.get_servers()
```

3 Returns:

- 4 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 5 • *servers* - a list of Python **proc** containing the identifiers of the servers to which the tool is
- 6 currently attached (dict)

7 See **PMIx\_tool\_get\_servers** for details.

## 8 **A.6.6 Tool.set\_server**

### 9 **Summary**

10 Designate a server as the tool's *primary* server.

11 *PMIx v4.0* **Format** Python

```
12 rc = mytool.set_server(proc:dict, info:list)
```

- 13 **IN** **proc**  
14 Python **proc** containing the identifier of the servers to which the tool is to attach (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 See **PMIx\_tool\_set\_server** for details.

## 20 **A.6.7 Tool.iof\_pull**

### 21 **Summary**

22 Register to receive output forwarded from a remote process.

## Format

Python

```
rc, id = mytool.iof_pull(sources:list, channel:integer,  
                        directives:list, cbfunc)
```

Python

### IN sources

List of Python [proc](#) dictionaries of processes whose IO is being requested (list)

### IN channel

Python [channel](#) bitmask identifying IO channels to be forwarded (integer)

### IN directives

List of Python [info](#) dictionaries describing request (list)

### IN cbfunc

Python [iofcbfunc](#) to receive IO payloads (func)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *id* - PMIx reference identifier for request (integer)

See [PMIx\\_IOF\\_pull](#) for description of all relevant attributes and behaviors.

## A.6.8 Tool.iof\_deregister

### Summary

Deregister from output forwarded from a remote process.

## Format

Python

```
rc = mytool.iof_deregister(id:integer, directives:list)
```

Python

### IN id

PMIx reference identifier returned by pull request (list)

### IN directives

List of Python [info](#) dictionaries describing request (list)

Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

See [PMIx\\_IOF\\_deregister](#) for description of all relevant attributes and behaviors.

## A.6.9 Tool.iof\_push

### Summary

Push data collected locally (typically from stdin) to stdin of target recipients.

## Format

Python

```
rc = mytool.iof_push(targets:list, data:dict, directives:list)
```

Python

### IN sources

List of Python `proc` of target processes (list)

### IN data

Python `byteobject` containing data to be delivered (dict)

### IN directives

Optional list of Python `info` describing request (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_IOF_push` for description of all relevant attributes and behaviors.

## A.7 Example Usage

The following examples are provided to illustrate the use of the Python bindings.

### A.7.1 Python Client

The following example contains a client program that illustrates a fairly common usage pattern. The program instantiates and initializes the `PMIxClient` class, posts some data that is to be shared across all processes in the job, executes a “fence” that circulates the data, and then retrieves a value posted by one of its peers. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *

def main():
    # Instantiate a client object
    myclient = PMIxClient()
    print("Testing PMIx ", myclient.get_version())

    # Initialize the PMIx client library, declaring the programming model
    # as "TEST" and the library name as "PMIX", just for the example
    info = ['key':PMIX_PROGRAMMING_MODEL,
            'value':'TEST', 'val_type':PMIX_STRING,
            'key':PMIX_MODEL_LIBRARY_NAME,
            'value':'PMIX', 'val_type':PMIX_STRING]
    rc,myname = myclient.init(info)
```

```

1     if PMIX_SUCCESS != rc:
2         print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
3         exit(1)
4
5     # try posting a value
6     rc = myclient.put(PMIX_GLOBAL, "mykey",
7                       'value':1, 'val_type':PMIX_INT32)
8     if PMIX_SUCCESS != rc:
9         print("PMIx_Put FAILED WITH ERROR", myclient.error_string(rc))
10        # cleanly finalize
11        myclient.finalize()
12        exit(1)
13
14    # commit it
15    rc = myclient.commit()
16    if PMIX_SUCCESS != rc:
17        print("PMIx_Commit FAILED WITH ERROR",
18              myclient.error_string(rc))
19        # cleanly finalize
20        myclient.finalize()
21        exit(1)
22
23    # execute fence across all processes in my job
24    procs = []
25    info = []
26    rc = myclient.fence(procs, info)
27    if PMIX_SUCCESS != rc:
28        print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
29        # cleanly finalize
30        myclient.finalize()
31        exit(1)
32
33    # Get a value from a peer
34    if 0 != myname['rank']:
35        info = []
36        rc, get_val = myclient.get('nspace':"testnspace", 'rank': 0,
37                                  "mykey", info)
38        if PMIX_SUCCESS != rc:
39            print("PMIx_Commit FAILED WITH ERROR",
40                  myclient.error_string(rc))
41            # cleanly finalize
42            myclient.finalize()
43            exit(1)

```



```

1         print("Get value returned: ", get_val)
2
3     # test a fence that should return not_supported because
4     # we pass a required attribute that the server is known
5     # not to support
6     procs = []
7     info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
8            'value':10, 'val_type':PMIX_INT]
9     rc = myclient.fence(procs, info)
10    if PMIX_SUCCESS == rc:
11        print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
12        # cleanly finalize
13        myclient.finalize()
14        exit(1)
15
16    # Publish something
17    info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
18    rc = myclient.publish(info)
19    if PMIX_SUCCESS != rc:
20        print("PMIx_Publish FAILED WITH ERROR",
21            myclient.error_string(rc))
22        # cleanly finalize
23        myclient.finalize()
24        exit(1)
25
26    # finalize
27    info = []
28    myclient.finalize(info)
29    print("Client finalize complete")
30
31    # Python main program entry point
32    if __name__ == '__main__':
33        main()

```

Python

## 34 A.7.2 Python Server

35 The following example contains a minimum-level server host program that instantiates and  
36 initializes the `PMIxServer` class. The program illustrates passing several server module functions to  
37 the bindings and includes code to setup and spawn a simple client application, waiting until the  
38 spawned client terminates before finalizing and exiting itself. Note that the example has been  
39 formatted to fit the document layout.

```

1  from pmix import *
2  import signal, time
3  import os
4  import select
5  import subprocess
6
7  def clientconnected(proc:tuple is not None):
8      print("CLIENT CONNECTED", proc)
9      return PMIX_OPERATION_SUCCEEDED
10
11 def clientfinalized(proc:tuple is not None):
12     print("CLIENT FINALIZED", proc)
13     return PMIX_OPERATION_SUCCEEDED
14
15 def clientfence(procs:list, directives:list, data:bytearray):
16     # check directives
17     if directives is not None:
18         for d in directives:
19             # these are each an info dict
20             if "pmix" not in d['key']:
21                 # we do not support such directives - see if
22                 # it is required
23                 try:
24                     if d['flags'] & PMIX_INFO_REQD:
25                         # return an error
26                         return PMIX_ERR_NOT_SUPPORTED
27                 except:
28                     #it can be ignored
29                     pass
30     return PMIX_OPERATION_SUCCEEDED
31
32 def main():
33     try:
34         myserver = PMIXServer()
35     except:
36         print("FAILED TO CREATE SERVER")
37         exit(1)
38     print("Testing server version ", myserver.get_version())
39
40     args = ['key':PMIX_SERVER_SCHEDULER,
41            'value':'T', 'val_type':PMIX_BOOL]
42     map = 'clientconnected': clientconnected,

```

```

1         'clientfinalized': clientfinalized,
2         'fencenb': clientfence
3 my_result = myserver.init(args, map)
4
5     # get our environment as a base
6     env = os.environ.copy()
7
8     # register an nspace for the client app
9     (rc, regex) = myserver.generate_regex("test000,test001,test002")
10    (rc, ppn) = myserver.generate_ppn("0")
11    kvals = ['key':PMIX_NODE_MAP,
12            'value':regex, 'val_type':PMIX_STRING,
13            'key':PMIX_PROC_MAP,
14            'value':ppn, 'val_type':PMIX_STRING,
15            'key':PMIX_UNIV_SIZE,
16            'value':1, 'val_type':PMIX_UINT32,
17            'key':PMIX_JOB_SIZE,
18            'value':1, 'val_type':PMIX_UINT32]
19    rc = foo.register_nspace("testnspace", 1, kvals)
20    print("RegNspace ", rc)
21
22    # register a client
23    uid = os.getuid()
24    gid = os.getgid()
25    rc = myserver.register_client('nspace':"testnspace", 'rank':0,
26                                uid, gid)
27    print("RegClient ", rc)
28    # setup the fork
29    rc = myserver.setup_fork('nspace':"testnspace", 'rank':0, env)
30    print("SetupFrk", rc)
31
32    # setup the client argv
33    args = ["/client.py"]
34    # open a subprocess with stdout and stderr
35    # as distinct pipes so we can capture their
36    # output as the process runs
37    p = subprocess.Popen(args, env=env,
38                          stdout=subprocess.PIPE, stderr=subprocess.PIPE)
39    # define storage to catch the output
40    stdout = []
41    stderr = []
42    # loop until the pipes close
43    while True:

```

```

1     reads = [p.stdout.fileno(), p.stderr.fileno()]
2     ret = select.select(reads, [], [])
3
4     stdout_done = True
5     stderr_done = True
6
7     for fd in ret[0]:
8         # if the data
9         if fd == p.stdout.fileno():
10            read = p.stdout.readline()
11            if read:
12                read = read.decode('utf-8').rstrip()
13                print('stdout: ' + read)
14                stdout_done = False
15            elif fd == p.stderr.fileno():
16                read = p.stderr.readline()
17                if read:
18                    read = read.decode('utf-8').rstrip()
19                    print('stderr: ' + read)
20                    stderr_done = False
21
22            if stdout_done and stderr_done:
23                break
24            print("FINALIZING")
25            myserver.finalize()
26
27
28 if __name__ == '__main__':
29     main()

```

Python

## APPENDIX B

# Use-Cases

---

1 The PMIx standard provides many generic interfaces that can be composed into higher-level use  
2 cases in a variety of ways. While the specific interfaces and attributes are standardized, the use  
3 cases themselves are not (and should not) be standardized. Common use cases are included here as  
4 examples of how PMIx's generic interfaces *might* be composed together for a higher-level purpose.  
5 The use cases are intended for both PMIx interface users and library implementors. Whereby a  
6 better understanding of the general usage model within the community can help users picking up  
7 PMIx for the first and help implementors optimize their implementation for the common cases.

8 Each use case is structured to provide background information about the high-level use case as well  
9 as specific details about how the PMIx interfaces are used within the use case. Some use cases even  
10 provide code snippets. These code snippets are apart of larger code examples located within the  
11 standard's source code repository, and each complete code example is fully compilable and  
12 runnable. The related interfaces and attributes collected at the bottom of each use case are mainly  
13 for convenience and link to the full standardized definitions.

## 14 B.1 Business Card Exchange for Process-to- 15 Process Wire-up

### 16 B.1.1 Use Case Summary

17 Multi-process communication libraries, such as MPI, need to establish communication channels  
18 between a set of those processes. In this scenario, each process needs to share connectivity  
19 information (a.k.a. Business Cards) with all other processes before communication channels can be  
20 established. This connectivity information may take the form of one or more unique strings that  
21 allow a different process to establish a communication channel with the originator. The runtime  
22 environment must provide a mechanism for the efficient exchange of this connectivity information.  
23 Additional information about the current state of the job (e.g., number of processes globally and  
24 locally) and of how the process was started (e.g., process binding) is also helpful.

25 Note: The Instant-On wire-up mechanism is a separate, related use case.

## 1 B.1.2 Use Case Details

2 Each process provides their business card to PMIx via one or more **PMIx\_Put** operations to store  
3 the tuple of {**UID**, **key**, **value**}. The **UID** is the unique name for this process in the PMIx  
4 universe (i.e., **namespace** and **rank**). The **key** is a unique key that other processes can reference  
5 generically (note that since the **UID** is also associated with the **key** there is no need to make the  
6 **key** uniquely named per process). The **value** is the string representation of the connectivity  
7 information.

8 Some business card information is meant for remote processes (e.g., TCP or InfiniBand addresses)  
9 while others are meant only for local processes (e.g., shared memory information). As such a  
10 **scope** should be associated with the **PMIx\_Put** operation to differentiate this intention.

11 The **PMIx\_Put** operations may be cached local to the process. Once all **PMIx\_Put** operations  
12 have been called each process should call **PMIx\_Commit** to push those values to the local PMIx  
13 server. Note that in a multi-library configuration each library may **PMIx\_Put** then  
14 **PMIx\_Commit** values - so there may be multiple **PMIx\_Commit** calls before a Business Card  
15 Exchange is activated.

16 After calling **PMIx\_Commit** a process can activate the Business Card Exchange collective  
17 operation by calling **PMIx\_Fence**. The **PMIx\_Fence** operation is collective over the set of  
18 processes specified in the argument set. That allows for the collective to span a subset of a  
19 namespace or multiple namespaces. After the completion of the **PMIx\_Fence** operation, the data  
20 stored by other processes via **PMIx\_Put** is available to the local process through a call to  
21 **PMIx\_Get** which returns the key/value pairs necessary to establish the connection(s) with the  
22 other processes.

23 The **PMIx\_Fence** operation has a "Synchronize Only" mode that works as a barrier operation.  
24 This is helpful if the communication library requires a synchronization before leaving initialization  
25 or starting finalization, for example.

26 The **PMIx\_Fence** operation has a "Sparse" mode in addition to a "Full" mode for the data  
27 exchange. The "Full" mode will fully exchange all Business Card information with all other  
28 processes. This is helpful for tightly communicating applications. The "Sparse" mode will  
29 dynamically pull the connectivity information on-demand from inside of **PMIx\_Get** (if it is not  
30 already available locally). This is helpful for sparsely communicating applications. Since which  
31 mode is best for an application cannot be inferred by the PMIx library the caller must specify which  
32 mode works best for their application. The **PMIx\_Fence** operation has an option for the end user  
33 to specify which mode they desire for this operation.

34 Additional information about the current state of the job (e.g., number of processes globally and  
35 locally) and of how the process was started (e.g., process binding) is also helpful. This "job level"  
36 information is available immediately after **PMIx\_Init** without the need for any explicit  
37 synchronization.

38 The number of processes globally in the namespace and this process's rank within that namespace  
39 is important to know before establishing the Business Card information to best allocate resources.

1 The number of processes local to the node and this process's local rank is important to know before  
2 establishing the Business Card information to help the caller determine the scope of the put  
3 operation. For example, to designate a leader to set up a shared memory segment of the proper size  
4 before putting that information into the locally scoped Business Card information.

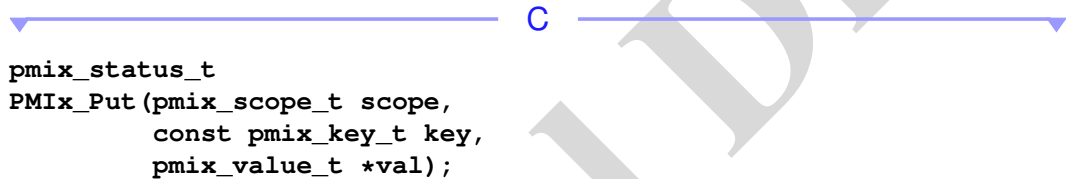
5 The number of processes local to a remote node is also helpful to know before establishing the  
6 Business Card information. This information is useful to pre-establish local resources before that  
7 remote node starts to initiate a connection or to determine the number of connections that need to  
8 be advertised in the Business Card when it is sent out.

9 Note that some of the job level information may change over the course of the job in a dynamic  
10 application.

## 11 Related Interfaces

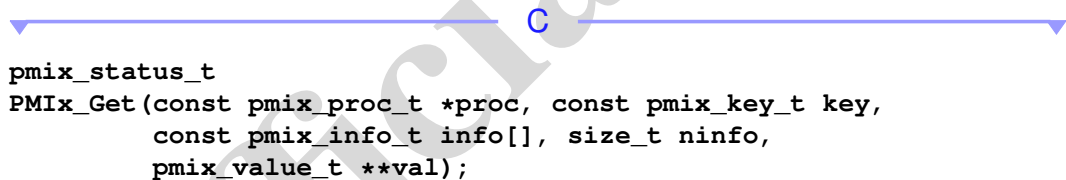
### 12 *PMIx v1.0* **PMIx\_Put**

```
13 pmix_status_t  
14 PMIx_Put (pmix_scope_t scope,  
15           const pmix_key_t key,  
16           pmix_value_t *val);
```



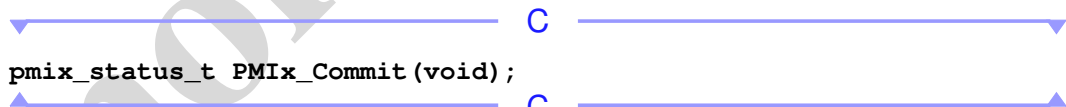
### 17 *PMIx v1.0* **PMIx\_Get**

```
18 pmix_status_t  
19 PMIx_Get (const pmix_proc_t *proc, const pmix_key_t key,  
20           const pmix_info_t info[], size_t ninfo,  
21           pmix_value_t **val);
```



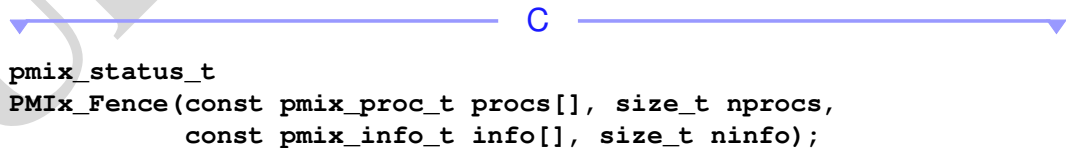
### 22 *PMIx v1.0* **PMIx\_Commit**

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



### 24 *PMIx v1.0* **PMIx\_Fence**

```
25 pmix_status_t  
26 PMIx_Fence (const pmix_proc_t procs[], size_t nprocs,  
27             const pmix_info_t info[], size_t ninfo);
```



---

## PMIx\_Init

```
pmix_status_t
PMIx_Init (pmix_proc_t *proc,
           pmix_info_t info[], size_t ninfo)
```

---

### Related Attributes

The following job level information is useful to have before establishing Business Card information:

**PMIX\_NODE\_LIST** "pmix.nlist" (char\*)

Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults to the *job* realm.

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

Number of nodes currently hosting processes in the specified realm. Defaults to the *job* realm.

**PMIX\_NODEID** "pmix.nodeid" (uint32\_t)

Node identifier expressed as the node's index (beginning at zero) in an array of nodes within the active session. The value must be unique and directly correlate to the **PMIX\_HOSTNAME** of the node - i.e., users can interchangeably reference the same location using either the **PMIX\_HOSTNAME** or corresponding **PMIX\_NODEID**.

**PMIX\_JOB\_SIZE** "pmix.job.size" (uint32\_t)

Total number of processes in the specified job across all contained applications. Note that this value can be different from **PMIX\_MAX\_PROCS**. For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running *job* on-the-fly. In the latter case, PMIx events may be used to notify processes within the job that the job size has changed.

**PMIX\_PROC\_MAP** "pmix.pmap" (char\*)

Regular expression describing processes on each node in the specified realm - see [17.2.3.2](#) for an explanation of its generation. Defaults to the *job* realm.

**PMIX\_LOCAL\_PEERS** "pmix.lpeers" (char\*)

Comma-delimited list of ranks that are executing on the local node within the specified namespace – shortcut for **PMIx\_Resolve\_peers** for the local node.

**PMIX\_LOCAL\_SIZE** "pmix.local.size" (uint32\_t)

Number of processes in the specified job or application realm on the caller's node. Defaults to job realm unless the **PMIX\_APP\_INFO** and the **PMIX\_APPNUM** qualifiers are given.



1 For each process this information is also useful (note that any one process may want to access this  
2 list of information about any other process in the system):

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

4 Process rank within the job, starting from zero.

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

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

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

10 Rank of the specified process spanning across all jobs in this session, starting with zero.  
11 Note that no ordering of the jobs is implied when computing this value. As jobs can start and  
12 end at random times, this is defined as a continually growing number - i.e., it is not  
13 dynamically adjusted as individual jobs and processes are started or terminated.

14 **PMIX\_LOCALITY\_STRING** "pmix.locstr" (char\*)

15 String describing a process's bound location - referenced using the process's rank. The string  
16 is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon. The  
17 remainder of the string represents the corresponding locality as expressed by the underlying  
18 implementation. The entire string must be passed to **PMIx\_Get\_relative\_locality**  
19 for processing. Note that hosts are only required to provide locality strings for local client  
20 processes - thus, a call to **PMIx\_Get** for the locality string of a process that returns  
21 **PMIX\_ERR\_NOT\_FOUND** indicates that the process is not executing on the same node.

22 **PMIX\_HOSTNAME** "pmix.hname" (char\*)

23 Name of the host, as returned by the **gethostname** utility or its equivalent.

24 There are other keys that are helpful to have before a synchronization point. This is not meant to be  
25 a comprehensive list.

## 26 B.2 Debugging

### 27 B.2.1 Terminology

#### 28 B.2.1.1 Tools vs Debuggers

29 A *tool* is a process designed to monitor, record, analyze, or control the execution of another  
30 process. Typically used for the purposes of profiling and debugging. A *first-party tool* runs within  
31 the address space of the application process while a *third-party tool* run within its own process. A  
32 *debugger* is a third-party tool that inspects and controls an application process's execution using  
33 system-level debug APIs (e.g., **ptrace**).

## 1 B.2.1.2 Parallel Launching Methods

2 A *starter* program is a program responsible for launching a parallel runtime, such as MPI. PMIx  
3 supports two primary methods for launching parallel applications under tools and debuggers:  
4 indirect and direct. In the indirect launching method (Section 18.2.2, the tool is attached to the  
5 starter. In the direct launching method (Section 18.2.1, the tool takes the place of the starter. PMIx  
6 also supports attaching to already running programs via the *Process Acquisition* interfaces  
7 (Section B.2.1.4).

## 8 B.2.1.3 Process Synchronization

9 Process Synchronization is a technique tools use to start the processes of a parallel application such  
10 that the tools can still attach to the process early in its lifetime. Said another way, the tool must be  
11 able to start the application processes without them “running away” from the tool. In the case of  
12 MPI (Version 3.1 [4] or the MPI World Process in future versions), this means stopping the  
13 applications processes before they return from `MPI_Init` or `MPI_Init_thread`.

## 14 B.2.1.4 Process Acquisition

15 Process Acquisition is a technique tools use to locate all of the processes, local and remote, of a  
16 given parallel application. This typically boils down to collecting the following information for  
17 every process in the parallel application: the hostname or IP of the machine running the process,  
18 the executable name, and the process ID.

## 19 B.2.2 Use Case Details

### 20 B.2.2.1 Direct-Launch Debugger Tool

21 PMIx can support the tool itself using the PMIx spawn options to control the app’s startup,  
22 including directing the RM/application as to when to block and wait for tool attachment, or  
23 stipulating that an interceptor library be preloaded. However, this means that the user is restricted to  
24 whatever command line options the tool vendor has provided for operations such as process  
25 placement and binding, which places a significant burden on the tool vendor. An example might  
26 look like the following: `dbgr -n 3 ./myapp`.

27 Assuming it is supported, co-launch of debugger daemons in this use-case is supported by adding a  
28 `pmix_app_t` to the `PMIx_Spawn` command, indicating that the resulting processes are  
29 debugger daemons by setting the `PMIX_DEBUGGER_DAEMONS` attribute.

### 30 Related Interfaces

31 *PMIx v2.0*

[PMIx\\_tool\\_init](#)

C

```
32 pmix_status_t  
33 PMIx_tool_init(pmix_proc_t *proc,  
34               pmix_info_t info[], size_t ninfo);
```

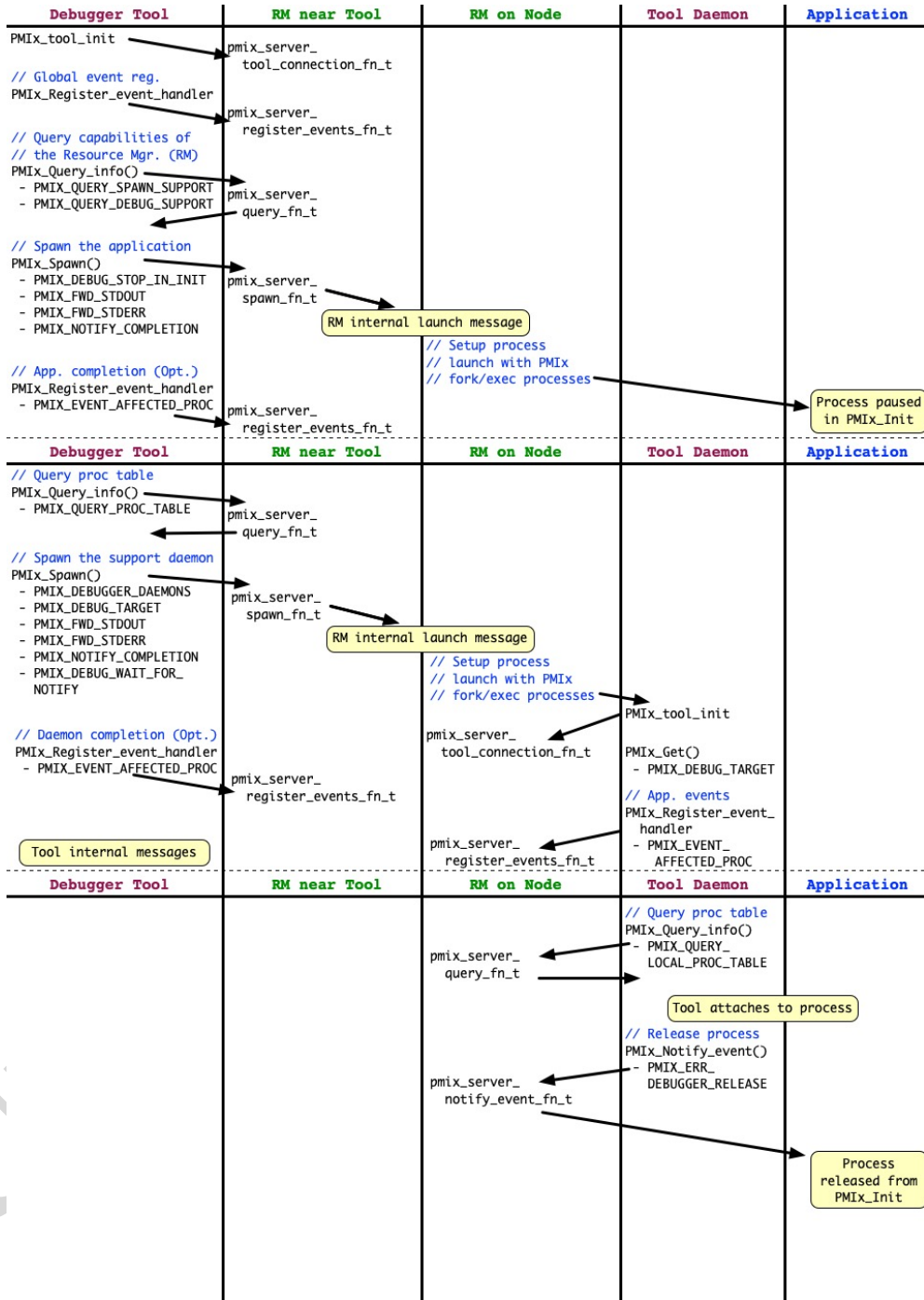


Figure B.1.: Interaction diagram showing an example of the Direct Launch mechanism

```

1 PMIx v2.0 PMIx_Register_event_handler
2
3 pmix_status_t
4 PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
5 pmix_info_t info[], size_t ninfo,
6 pmix_notification_fn_t evhdlr,
7 pmix_hdlr_reg_cbfunc_t cbfunc,
8 void *cbdata);
9
10 PMIx v4.0 PMIx_Query_info
11
12 pmix_status_t
13 PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
14 pmix_info_t *info[], size_t *ninfo);
15
16 PMIx v1.0 PMIx_Spawn
17
18 pmix_status_t
19 PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
20 const pmix_app_t apps[], size_t napps,
21 char nspace[])
22
23 PMIx v1.0 PMIx_Get
24
25 pmix_status_t
26 PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
27 const pmix_info_t info[], size_t ninfo,
28 pmix_value_t **val);
29
30 PMIx v2.0 PMIx_Notify_event

```

```

1 pmix_status_t
2 PMIx_Notify_event(pmix_status_t status,
3                 const pmix_proc_t *source,
4                 pmix_data_range_t range,
5                 pmix_info_t info[], size_t ninfo,
6                 pmix_op_cbfunc_t cbfunc, void *cbdata);

```

## Related Attributes

**PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (bool)

Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

**PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (bool)

Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

**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 either released by debugger modification of an appropriate variable or 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\_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.

**PMIX\_DEBUG\_DAEMONS\_PER\_PROC** "pmix.dbg.dpproc" (uint16\_t)

Number of debugger daemons to be spawned per application process. The launcher is to pass the identifier of the namespace to be debugged by including the `PMIX_DEBUG_TARGET` attribute in the daemon's job-level information. 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.

**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\_COSPAWN\_APP** "pmix.cospawn" (bool)

8 Designated application is to be spawned as a disconnected job - i.e., the launcher shall not  
9 include the application in any of the job-level values (e.g., **PMIX\_RANK** within the job)  
10 provided to any other application process generated by the same spawn request. Typically  
11 used to cospawn debugger daemons alongside an application.

12 **PMIX\_MAPBY** "pmix.mapby" (char\*)

13 Process mapping policy - when accessed using **PMIx\_Get**, use the  
14 **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
15 provided namespace. Supported values are launcher specific.

16 **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)

17 Requests that the ability to forward the **stdout** 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\_FWD\_STDERR** "pmix.fwd.stderr" (bool)

21 Requests that the ability to forward the **stderr** of the spawned processes be maintained.  
22 The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
23 other options for delivery of the forwarded output.

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\_SETUP\_APP\_ENVARS** "pmix.setup.env" (bool)

32 Harvest and include relevant environmental variables.

33 **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (pmix\_proc\_t)

34 The single process that was affected.

35 **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)

36 Included in the **pmix\_info\_t** array of a **pmix\_app\_t**, this attribute declares that the  
37 application consists of debugger daemons and shall be governed accordingly. If used as the  
38 sole **pmix\_app\_t** in a **PMIx\_Spawn** request, then the **PMIX\_DEBUG\_TARGET** attribute  
39 must also be provided (in either the *job\_info* or in the *info* array of the **pmix\_app\_t**) to

1 identify the namespace to be debugged so that the launcher can determine where to place the  
2 spawned daemons. If neither `PMIX_DEBUG_DAEMONS_PER_PROC` nor  
3 `PMIX_DEBUG_DAEMONS_PER_NODE` is specified, then the launcher shall default to a  
4 placement policy of one daemon per process in the target job.

5 `PMIX_DEBUG_TARGET` "pmix.dbg.tgt" (pmix\_proc\_t\*)

6 Identifier of process(es) to be debugged - a rank of `PMIX_RANK_WILDCARD` indicates that  
7 all processes in the specified namespace are to be included.

8 `PMIX_DEBUG_WAIT_FOR_NOTIFY` "pmix.dbg.notify" (bool)

9 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
10 applies only to that application) or in the `job_info` array if it applies to all applications in the  
11 given spawn request. Indicates that the specified application is being spawned under a  
12 debugger. The resulting application processes are to notify their server (by generating the  
13 `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some  
14 application-determined location and pause at that point until either released by debugger  
15 modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE`  
16 event. The launcher (RM or IL) is responsible for generating the  
17 `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they are  
18 at the pause point.

19 `PMIX_QUERY_LOCAL_PROC_TABLE` "pmix.qry.lptable" (char\*)

20 Returns a (`pmix_data_array_t`) array of `pmix_proc_info_t`, one entry for each  
21 process in the specified namespace executing on the same node as the requester, ordered by  
22 process job rank. REQUIRED QUALIFIER: `PMIX_NAMESPACE` indicating the namespace  
23 whose local process table is being queried. OPTIONAL QUALIFIER: `PMIX_HOSTNAME`  
24 indicating the host whose local process table is being queried. By default, the query assumes  
25 that the host upon which the request was made is to be used.

## 26 Related Constants

27 `PMIX_DEBUG_WAITING_FOR_NOTIFY`

28 `PMIX_DEBUGGER_RELEASE`

### 29 B.2.2.2 Indirect-Launch Debugger Tool

30 Executing a program under a tool using an intermediate launcher such as `mpiexec` can also be  
31 made possible. This requires some degree of coordination between the tool and the launcher.  
32 Ultimately, it is the launcher that is going to launch the application, and the tool must somehow  
33 inform the launcher (and the application) that this is being done in a debug session so that the  
34 application knows to “block” until the tool attaches to it.

35 In this operational mode, the user invokes a tool (typically on a non-compute, or “head”, node) that  
36 in turn uses `mpiexec` to launch their application – a typical command line might look like the  
37 following: `dbgr -dbgoption mpiexec -n 32 ./myapp`.



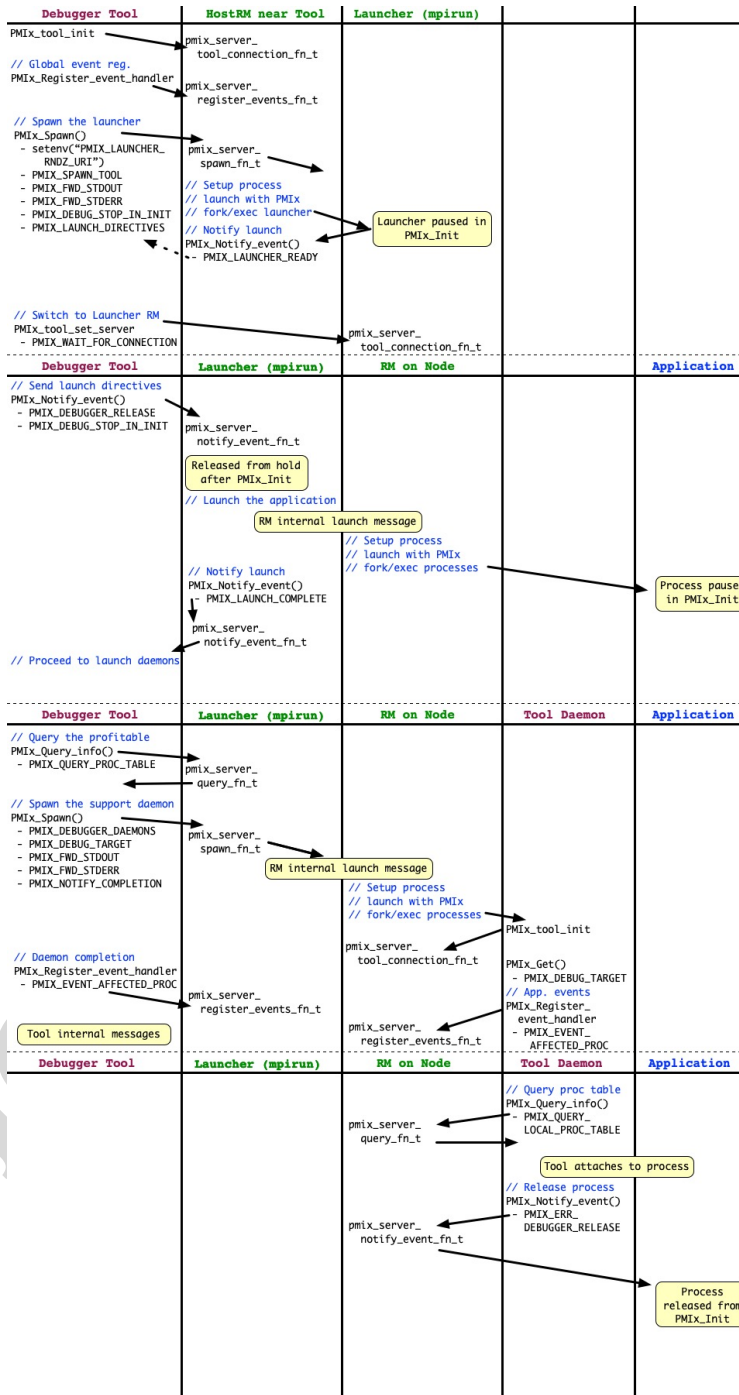


Figure B.2.: Interaction diagram showing an example of the Indirect Launch mechanism



1 **Related Interfaces**

2 *PMIx v2.0* **PMIx\_tool\_init**

▼ C ▼

3 `pmix_status_t`

4 `PMIx_tool_init(pmix_proc_t *proc,`  
5 `pmix_info_t info[], size_t ninfo);`

▲ C ▲

6 *PMIx v2.0* **PMIx\_Register\_event\_handler**

▼ C ▼

7 `pmix_status_t`

8 `PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,`  
9 `pmix_info_t info[], size_t ninfo,`  
10 `pmix_notification_fn_t evhdlr,`  
11 `pmix_hdlr_reg_cbfunc_t cbfunc,`  
12 `void *cbdata);`

▲ C ▲

13 *PMIx v1.0* **PMIx\_Spawn**

▼ C ▼

14 `pmix_status_t`

15 `PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,`  
16 `const pmix_app_t apps[], size_t napps,`  
17 `char nspace[])`

▲ C ▲

18 *PMIx v2.0* **PMIx\_Notify\_event**

▼ C ▼

19 `pmix_status_t`

20 `PMIx_Notify_event(pmix_status_t status,`  
21 `const pmix_proc_t *source,`  
22 `pmix_data_range_t range,`  
23 `pmix_info_t info[], size_t ninfo,`  
24 `pmix_op_cbfunc_t cbfunc, void *cbdata);`

▲ C ▲

25 *PMIx v4.0* **PMIx\_tool\_attach\_to\_server**

```

1  pmix_status_t
2  PMIx_tool_attach_to_server(pmix_proc_t *proc,
3                             pmix_proc_t *server,
4                             pmix_info_t info[],
5                             size_t ninfo);

```

6 *PMIx v4.0* **PMIx\_Query\_info**

```

7  pmix_status_t
8  PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
9                 pmix_info_t *info[], size_t *ninfo);

```

10 *PMIx v1.0* **PMIx\_Get**

```

11 pmix_status_t
12 PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
13          const pmix_info_t info[], size_t ninfo,
14          pmix_value_t **val);

```

## 15 Related Attributes

16 **PMIX\_LAUNCH\_DIRECTIVES** "pmix.lnch.dirs" (pmix\_data\_array\_t\*)

17 Array of **pmix\_info\_t** containing directives for the launcher - a convenience attribute for  
18 retrieving all directives with a single call to **PMIx\_Get**.

19 **PMIX\_SPAWN\_TOOL** "pmix.spwn.tool" (bool)

20 Indicate that the job being spawned is a tool.

21 **PMIX\_COSPAWN\_APP** "pmix.cospawn" (bool)

22 Designated application is to be spawned as a disconnected job - i.e., the launcher shall not  
23 include the application in any of the job-level values (e.g., **PMIX\_RANK** within the job)  
24 provided to any other application process generated by the same spawn request. Typically  
25 used to cospawn debugger daemons alongside an application.

26 **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)

27 Requests that the ability to forward the **stdout** of the spawned processes be maintained.  
28 The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
29 other options for delivery of the forwarded output.

30 **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)

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_SETUP_APP_ENVARS` "`pmix.setup.env`" (`bool`)

5 Harvest and include relevant environmental variables.

6 `PMIX_DEBUG_STOP_IN_INIT` "`pmix.dbg.init`" (`bool`)

7 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
8 applies only to that application) or in the `job_info` array if it applies to all applications in the  
9 given spawn request. Indicates that the specified application is being spawned under a  
10 debugger. The PMIx client library in each resulting application process shall notify its PMIx  
11 server that it is pausing and then pause during `PMIx_Init` of the spawned processes until  
12 either released by debugger modification of an appropriate variable or receipt of the  
13 `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating  
14 the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have reached the  
15 pause point.

16 `PMIX_DEBUG_STOP_ON_EXEC` "`pmix.dbg.exec`" (`bool`)

17 Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive  
18 applies only to that application) or in the `job_info` array if it applies to all applications in the  
19 given spawn request. Indicates that the application is being spawned under a debugger, and  
20 that the local launch agent is to pause the resulting application processes on first instruction  
21 for debugger attach. The launcher (RM or IL) is to generate the  
22 `PMIX_LAUNCH_COMPLETE` event when all processes are stopped at the exec point.

23 `PMIX_DEBUG_DAEMONS_PER_PROC` "`pmix.dbg.dpproc`" (`uint16_t`)

24 Number of debugger daemons to be spawned per application process. The launcher is to pass  
25 the identifier of the namespace to be debugged by including the `PMIX_DEBUG_TARGET`  
26 attribute in the daemon's job-level information. The debugger daemons spawned on a given  
27 node are responsible for self-determining their specific target process(es) - e.g., by  
28 referencing their own `PMIX_LOCAL_RANK` in the daemon debugger job versus the  
29 corresponding `PMIX_LOCAL_RANK` of the target processes on the node.

30 `PMIX_DEBUG_DAEMONS_PER_NODE` "`pmix.dbg.dpnd`" (`uint16_t`)

31 Number of debugger daemons to be spawned on each node where the target job is executing.  
32 The launcher is to pass the identifier of the namespace to be debugged by including the  
33 `PMIX_DEBUG_TARGET` attribute in the daemon's job-level information. The debugger  
34 daemons spawned on a given node are responsible for self-determining their specific target  
35 process(es) - e.g., by referencing their own `PMIX_LOCAL_RANK` in the daemon debugger  
36 job versus the corresponding `PMIX_LOCAL_RANK` of the target processes on the node.

37 `PMIX_MAPBY` "`pmix.mapby`" (`char*`)

38 Process mapping policy - when accessed using `PMIx_Get`, use the  
39 `PMIX_RANK_WILDCARD` value for the rank to discover the mapping policy used for the  
40 provided namespace. Supported values are launcher specific.

1 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.phtable" (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 QUALIFIER:  
4 **PMIX\_NAMESPACE** indicating the namespace whose process table is being queried.

5 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lhtable" (char\*)

6 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
7 process in the specified namespace executing on the same node as the requester, ordered by  
8 process job rank. REQUIRED QUALIFIER: **PMIX\_NAMESPACE** indicating the namespace  
9 whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME**  
10 indicating the host whose local process table is being queried. By default, the query assumes  
11 that the host upon which the request was made is to be used.

12 **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)

13 Included in the **pmix\_info\_t** array of a **pmix\_app\_t**, this attribute declares that the  
14 application consists of debugger daemons and shall be governed accordingly. If used as the  
15 sole **pmix\_app\_t** in a **PMIX\_Spawn** request, then the **PMIX\_DEBUG\_TARGET** attribute  
16 must also be provided (in either the *job\_info* or in the *info* array of the **pmix\_app\_t**) to  
17 identify the namespace to be debugged so that the launcher can determine where to place the  
18 spawned daemons. If neither **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** nor  
19 **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** is specified, then the launcher shall default to a  
20 placement policy of one daemon per process in the target job.

21 **PMIX\_NOTIFY\_COMPLETION** "pmix.notecomp" (bool)

22 Requests that the launcher generate the **PMIX\_EVENT\_JOB\_END** event for normal or  
23 abnormal termination of the spawned job. The event shall include the returned status code  
24 (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job; the identity (**PMIX\_PROCID**)  
25 and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if applicable; and a  
26 **PMIX\_EVENT\_TIMESTAMP** indicating the time the termination occurred. Note that the  
27 requester must register for the event or capture and process it within a default event handler.

28 **PMIX\_DEBUG\_TARGET** "pmix.dbg.tgt" (**pmix\_proc\_t\***)

29 Identifier of process(es) to be debugged - a rank of **PMIX\_RANK\_WILDCARD** indicates that  
30 all processes in the specified namespace are to be included.

31 **PMIX\_WAIT\_FOR\_CONNECTION** "pmix.wait.conn" (bool)

32 Wait until the specified process has connected to the requesting tool or server, or the  
33 operation times out (if the **PMIX\_TIMEOUT** directive is included in the request).

## 34 **Related Constants**

35 **PMIX\_LAUNCHER\_READY**

36 **PMIX\_LAUNCH\_COMPLETE**

37 **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY**

38 **PMIX\_DEBUGGER\_RELEASE**

39 **PMIX\_LAUNCHER\_RNDZ\_URI**

### 1 B.2.2.3 Attaching to a Running Job

2 PMIx supports attaching to an already running parallel job in two ways. In the first way, the main  
3 process of a tool calls `PMIx_Query_info` with the `PMIX_QUERY_PROC_TABLE` attribute.  
4 This returns an array of structs containing the information required for [process acquisition](#). This  
5 includes remote hostnames, executable names, and process IDs. In the second way, every tool  
6 daemon calls `PMIx_Query_info` with the `PMIX_QUERY_LOCAL_PROC_TABLE` attribute.  
7 This returns a similar array of structs but only for processes on the same node.

8 An example of this use-case may look like the following: `mpirun -n 32 ./myApp &&`  
9 `dbgr attach $!`

10 *PMIx v2.0*

#### `PMIx_tool_init`

```
▼ C ▼  
11 pmix_status_t  
12 PMIx_tool_init(pmix_proc_t *proc,  
13 pmix_info_t info[], size_t ninfo);  
▲ C ▲
```

14 *PMIx v2.0*

#### `PMIx_Register_event_handler`

```
▼ C ▼  
15 pmix_status_t  
16 PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,  
17 pmix_info_t info[], size_t ninfo,  
18 pmix_notification_fn_t evhdlr,  
19 pmix_hdlr_reg_cbfunc_t cbfunc,  
20 void *cbdata);  
▲ C ▲
```

21 *PMIx v4.0*

#### `PMIx_Query_info`

```
▼ C ▼  
22 pmix_status_t  
23 PMIx_Query_info(pmix_query_t queries[], size_t nqueries,  
24 pmix_info_t *info[], size_t *ninfo);  
▲ C ▲
```

25 *PMIx v1.0*

#### `PMIx_Spawn`

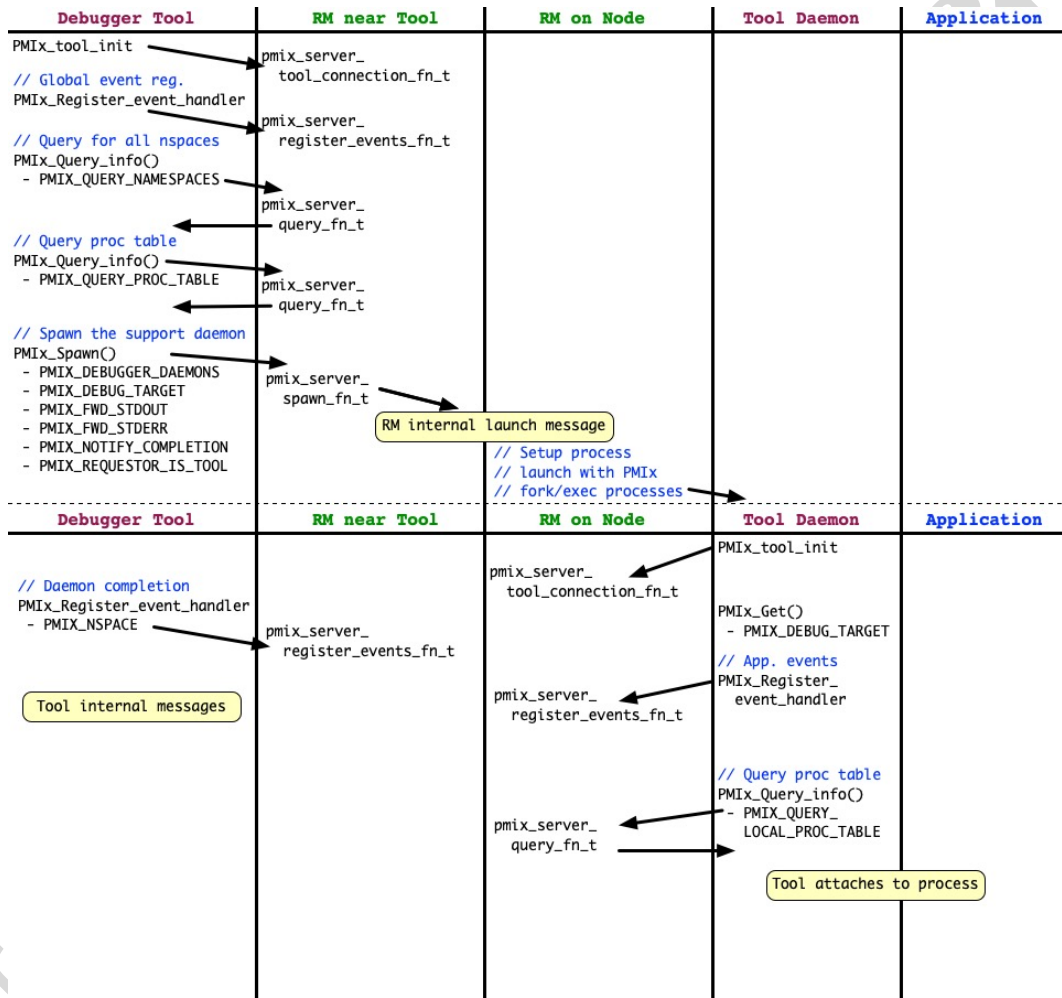


Figure B.3.: Interaction diagram showing an example of the attaching to a running job

```

1 pmix_status_t
2 PMix_Spawn(const pmix_info_t job_info[], size_t ninfo,
3           const pmix_app_t apps[], size_t napps,
4           char nspace[])

```

## Related Attributes

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

Returns a ([pmix\\_data\\_array\\_t](#)) array of [pmix\\_proc\\_info\\_t](#), one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: [PMIX\\_NAMESPACE](#) indicating the namespace whose process table is being queried.

**PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)

Included in the [pmix\\_info\\_t](#) array of a [pmix\\_app\\_t](#), this attribute declares that the application consists of debugger daemons and shall be governed accordingly. If used as the sole [pmix\\_app\\_t](#) in a [PMix\\_Spawn](#) request, then the [PMIX\\_DEBUG\\_TARGET](#) attribute must also be provided (in either the *job\_info* or in the *info* array of the [pmix\\_app\\_t](#)) to identify the namespace to be debugged so that the launcher can determine where to place the spawned daemons. If neither [PMIX\\_DEBUG\\_DAEMONS\\_PER\\_PROC](#) nor [PMIX\\_DEBUG\\_DAEMONS\\_PER\\_NODE](#) is specified, then the launcher shall default to a placement policy of one daemon per process in the target job.

**PMIX\_DEBUG\_TARGET** "pmix.dbg.tgt" ([pmix\\_proc\\_t\\*](#))

Identifier of process(es) to be debugged - a rank of [PMIX\\_RANK\\_WILDCARD](#) indicates that all processes in the specified namespace are to be included.

**PMIX\_DEBUG\_DAEMONS\_PER\_PROC** "pmix.dbg.dpproc" ([uint16\\_t](#))

Number of debugger daemons to be spawned per application process. The launcher is to pass the identifier of the namespace to be debugged by including the [PMIX\\_DEBUG\\_TARGET](#) attribute in the daemon's job-level information. 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.

**PMIX\_DEBUG\_DAEMONS\_PER\_NODE** "pmix.dbg.dpnd" ([uint16\\_t](#))

Number of debugger daemons to be spawned on each node where the target job is executing. The launcher is to pass the identifier of the namespace to be debugged by including the [PMIX\\_DEBUG\\_TARGET](#) attribute in the daemon's job-level information. 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.

**PMIX\_MAPBY** "pmix.mapby" (char\*)

1 Process mapping policy - when accessed using `PMIx_Get`, use the  
2 `PMIX_RANK_WILDCARD` value for the rank to discover the mapping policy used for the  
3 provided namespace. Supported values are launcher specific.

4 `PMIX_FWD_STDOUT` "pmix.fwd.stdout" (bool)

5 Requests that the ability to forward the `stdout` of the spawned processes be maintained.  
6 The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and  
7 other options for delivery of the forwarded output.

8 `PMIX_FWD_STDERR` "pmix.fwd.stderr" (bool)

9 Requests that the ability to forward the `stderr` of the spawned processes be maintained.  
10 The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and  
11 other options for delivery of the forwarded output.

12 `PMIX_NOTIFY_COMPLETION` "pmix.notecomp" (bool)

13 Requests that the launcher generate the `PMIX_EVENT_JOB_END` event for normal or  
14 abnormal termination of the spawned job. The event shall include the returned status code  
15 (`PMIX_JOB_TERM_STATUS`) for the corresponding job; the identity (`PMIX_PROCID`)  
16 and exit status (`PMIX_EXIT_CODE`) of the first failed process, if applicable; and a  
17 `PMIX_EVENT_TIMESTAMP` indicating the time the termination occurred. Note that the  
18 requester must register for the event or capture and process it within a default event handler.

19 `PMIX_REQUESTOR_IS_TOOL` "pmix.req.tool" (bool)

20 The requesting process is a PMIx tool.

21 `PMIX_QUERY_NAMESPACES` "pmix.qry.ns" (char\*)

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

### 23 B.2.2.4 Tool Interaction with RM

24 Tools can benefit from a mechanism by which they may interact with a local PMIx server that has  
25 opted to accept such connections along with support for tool connections to system-level PMIx  
26 servers, and a logging feature. To add support for tool connections to a specified system-level,  
27 PMIx server environments could choose to launch a set of PMIx servers to support a given  
28 allocation - these servers will (if so instructed) provide a tool rendezvous point that is tagged with  
29 their pid and typically placed in an allocation-specific temporary directory to allow for possible  
30 multi-tenancy scenarios. Supporting such operations requires that a system-level PMIx connection  
31 be provided which is not associated with a specific user or allocation. A new key has been added to  
32 direct the PMIx server to expose a rendezvous point specifically for this purpose.

33 *PMIx v2.0* `PMIx_Query_info_nb`



```

1      pmix_status_t
2      PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,
3                          pmix_info_cbfunc_t cbfunc, void *cbdata);
4 PMIx v2.0 PMIx_Register_event_handler
5
6      pmix_status_t
7      PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
8                                  pmix_info_t info[], size_t ninfo,
9                                  pmix_notification_fn_t evhdlr,
10                                 pmix_hdlr_reg_cbfunc_t cbfunc,
11                                 void *cbdata);
12 PMIx v2.0 PMIx_Deregister_event_handler
13
14      pmix_status_t
15      PMIx_Deregister_event_handler(size_t evhdlr_ref,
16                                    pmix_op_cbfunc_t cbfunc,
17                                    void *cbdata);
18 PMIx v2.0 PMIx_Notify_event
19
20      pmix_status_t
21      PMIx_Notify_event(pmix_status_t status,
22                        const pmix_proc_t *source,
23                        pmix_data_range_t range,
24                        pmix_info_t info[], size_t ninfo,
25                        pmix_op_cbfunc_t cbfunc, void *cbdata);
26 PMIx v1.0 PMIx_server_init

```

```

1  pmix_status_t
2  PMIx_server_init(pmix_server_module_t *module,
3                  pmix_info_t info[], size_t ninfo);

```

## B.2.2.5 Environmental Parameter Directives for Applications and Launchers

It is sometimes desirable or required that standard environmental variables (e.g., `PATH`, `LD_LIBRARY_PATH`, `LD_PRELOAD`) be modified prior to executing an application binary or a starter such as `mpirun` - this is particularly true when tools/debuggers are used to start the application.

### Related Interfaces

[PMIx\\_Spawn](#)

```

12 pmix_status_t
13 PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
14            const pmix_app_t apps[], size_t napps,
15            char nspace[])

```

### Related Structs

[pmix\\_env\\_t](#)

### Related Attributes

[PMIX\\_SET\\_ENVAR](#) "pmix.envar.set" (pmix\_env\_t\*)

Set the envar to the given value, overwriting any pre-existing one

[PMIX\\_ADD\\_ENVAR](#) "pmix.envar.add" (pmix\_env\_t\*)

Add the environment variable, but do not overwrite any pre-existing one

[PMIX\\_UNSET\\_ENVAR](#) "pmix.envar.unset" (char\*)

Unset the environment variable specified in the string.

[PMIX\\_PREPEND\\_ENVAR](#) "pmix.envar.prepnd" (pmix\_env\_t\*)

Prepend the given value to the specified environmental value using the given separator character, creating the variable if it doesn't already exist

[PMIX\\_APPEND\\_ENVAR](#) "pmix.envar.appnd" (pmix\_env\_t\*)

Append the given value to the specified environmental value using the given separator character, creating the variable if it doesn't already exist

Resource managers and launchers must scan for relevant directives, modifying environmental parameters as directed. Directives are to be processed in the order in which they were given, starting with job-level directives (applied to each app) followed by app-level directives.

## 1 **B.3 Hybrid Applications**

### 2 **B.3.1 Use Case Summary**

3 Hybrid applications (i.e., applications that utilize more than one programming model or runtime  
4 system, such as an application using MPI that also uses OpenMP or UPS) are growing in  
5 popularity, especially as processors with increasingly large numbers of cores and/or hardware  
6 threads proliferate. Unfortunately, the various corresponding runtime systems currently operate  
7 under the assumption that they alone control execution. This leads to conflicts in hybrid  
8 applications. Deadlock of parallel applications can occur when one runtime system prevents the  
9 other from making progress due to lack of coordination between them [3]. Sub-optimal  
10 performance can also occur due to uncoordinated division of hardware resources between the  
11 runtime systems implementing the different programming models or systems [5, 6]. This use-case  
12 offers potential solutions to this problem by providing a pathway for parallel runtime systems to  
13 coordinate their actions.

### 14 **B.3.2 Use Case Details**

#### 15 **B.3.2.1 Identifying Active Parallel Runtime Systems**

16 The current state-of-the-practice for concurrently used runtime systems in a single application to  
17 detect one another is via set environment variables. For example, some OpenMP implementations  
18 look for environment variables to indicate that an MPI library is active. Unfortunately, this  
19 technique is not completely reliable as environment variables change over time and with new  
20 software versions, and this detection is implementation specific. Also, the fact that an environment  
21 variable is present doesn't guarantee that a particular runtime system is in active use since Resource  
22 Managers routinely set environment variables "just in case" the application needs them. PMIx  
23 provides a reliable mechanism by which each library can determine that another runtime library is  
24 in operation.

25 When initializing PMIx, runtime libraries implementing a parallel programming model can register  
26 themselves, including their name, the library version, the version of the API they implement, and  
27 the threading model. This information is then cached locally and can then be read asynchronously  
28 by other runtime systems using PMIx's Event Notification system.

29 This initialization mechanism also allows runtime libraries to share knowledge of each other's  
30 resources and intended resource utilization. For example, if an OpenMP implementation knows  
31 which hardware threads an MPI library is using it could potentially avoid core and cache contention.

#### 32 **Code Example**

```
1 pmix_proc_t myproc;  
2 pmix_info_t *info;  
3 volatile bool wearedone = false;  
4
```

```

5 PMIX_INFO_CREATE(info, 4);
6 PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "MPI", PMIX_STRING);
7 PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "FooMPI",
  ↪ PMIX_STRING);
8 PMIX_INFO_LOAD(&info[2], PMIX_MODEL_LIBRARY_VERSION, "1.0.0",
  ↪ PMIX_STRING);
9 PMIX_INFO_LOAD(&info[3], PMIX_THREADING_MODEL, "posix", PMIX_STRING);
10 pmix_status_t rc = PMIx_Init(&myproc, info, 4);
11 PMIX_INFO_FREE(info, 4);

```

1

## 2 Related Interfaces

### 3 *PMIx v1.2* [PMIx\\_Init](#)

4

```

pmix_status_t
PMIx_Init(pmix_proc_t *proc,
          pmix_info_t info[], size_t ninfo)

```

6

C

4

5

```

pmix_status_t
PMIx_Init(pmix_proc_t *proc,
          pmix_info_t info[], size_t ninfo)

```

6

## 7 Related Attributes

8 [PMIX\\_PROGRAMMING\\_MODEL](#) "pmix.pgm.model" (char\*)  
 9 Programming model being initialized (e.g., "MPI" or "OpenMP").

10 [PMIX\\_MODEL\\_LIBRARY\\_NAME](#) "pmix.mdl.name" (char\*)  
 11 Programming model implementation ID (e.g., "OpenMPI" or "MPICH").

12 [PMIX\\_MODEL\\_LIBRARY\\_VERSION](#) "pmix.mdl.vrs" (char\*)  
 13 Programming model version string (e.g., "2.1.1").

14 [PMIX\\_THREADING\\_MODEL](#) "pmix.threads" (char\*)  
 15 Threading model used (e.g., "pthreads").

16 [PMIX\\_MODEL\\_NUM\\_THREADS](#) "pmix.mdl.nthrds" (uint64\_t)  
 17 Number of active threads being used by the model.

18 [PMIX\\_MODEL\\_NUM\\_CPUS](#) "pmix.mdl.ncpu" (uint64\_t)  
 19 Number of cpus being used by the model.

20 [PMIX\\_MODEL\\_CPU\\_TYPE](#) "pmix.mdl.cputype" (char\*)  
 21 Granularity - "hwthread", "core", etc.

22 [PMIX\\_MODEL\\_PHASE\\_NAME](#) "pmix.mdl.phase" (char\*)  
 23 User-assigned name for a phase in the application execution (e.g., "cfd reduction").

24 [PMIX\\_MODEL\\_PHASE\\_TYPE](#) "pmix.mdl.ptype" (char\*)  
 25 Type of phase being executed (e.g., "matrix multiply").

1 **PMIX\_MODEL\_AFFINITY\_POLICY** "pmix.mdl.tap" (char\*)  
2 Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread  
3 located on cpu close to master thread), "spread" (threads load-balanced across available  
4 cpus).

### 5 B.3.2.2 Coordinating at Runtime

6 The PMIx Event Notification system provides a mechanism by which the resource manager can  
7 communicate system events to applications, thus providing applications with an opportunity to  
8 generate an appropriate response. Hybrid applications can leverage these events for cross-library  
9 coordination.

10 Runtime libraries can access the information provided by other runtime libraries during their  
11 initialization using the event notification system. In this case, runtime libraries should register a  
12 callback for the **PMIX\_MODEL\_DECLARED** event.

13 Applications, runtime libraries, and resource managers can also use the PMIx event notification  
14 system to communicate dynamic information, such as entering a new application phase  
15 (**PMIX\_MODEL\_PHASE\_NAME**) or a change in resources used (**PMIX\_MODEL\_RESOURCES**).  
16 This dynamic information can be broadcast using the **PMIx\_Notify\_event** function. Runtime  
17 libraries can register callback functions to run when these events occur using  
18 **PMIx\_Register\_event\_handler**.

#### 19 Code Example

20 Registering a callback to run when another runtime library initializes:

```
1 static void model_declared_cb(size_t evhdlr_registration_id,  
2                               pmix_status_t status, const pmix_proc_t  
3                               ↪ *source,  
4                               pmix_info_t info[], size_t ninfo,  
5                               pmix_info_t results[], size_t nresults,  
6                               pmix_event_notification_cbfunc_fn_t  
7                               ↪ cbfunc,  
8                               void *cbdata) {  
9     printf("Entered %s\n", __FUNCTION__);  
10    int n;  
11    for (n = 0; n < ninfo; n++) {  
12        if (PMIX_CHECK_KEY(&info[n], PMIX_PROGRAMMING_MODEL) &&  
13            strcmp(info[n].value.data.string, "MPI") == 0) {  
14            /* ignore our own declaration */  
15            break;  
16        } else {  
17            /* actions to perform when another model registers */  
18        }  
19    }  
20    if (NULL != cbfunc) {  
21        /* tell the event handler that we are only a partial step */
```

```

20     cbfunc(PMIX_EVENT_PARTIAL_ACTION_TAKEN, NULL, 0, NULL, NULL,
21           ↪ cbdata);
22 }
23
24 pmix_status_t code = PMIX_MODEL_DECLARED;
25 rc = PMIx_Register_event_handler(&code, 1, NULL, 0, model_declared_cb,
26   ↪ NULL, NULL);

```

1

2

Notifying an event:

```

1 PMIX_INFO_CREATE(info, 1);
2 PMIX_INFO_LOAD(&info[0], PMIX_EVENT_NON_DEFAULT, NULL, PMIX_BOOL);
3 rc = PMIx_Notify_event(PMIX_OPENMP_PARALLEL_ENTERED, &myproc,
4   ↪ PMIX_RANGE_PROC_LOCAL, info, 1, notify_complete, (void*)&wearedone);

```

3

4

## Related Interfaces

5 *PMIx v2.0*

### [PMIx\\_Notify\\_event](#)

▼ C ▼

6

`pmix_status_t`

7

`PMIx_Notify_event`(`pmix_status_t` status,

8

const `pmix_proc_t` \*source,

9

`pmix_data_range_t` range,

10

`pmix_info_t` info[], `size_t` ninfo,

11

`pmix_op_cbfunc_t` cbfunc, void \*cbdata);

▲ C ▲

12 *PMIx v2.0*

### [PMIx\\_Register\\_event\\_handler](#)

▼ C ▼

13

`pmix_status_t`

14

`PMIx_Register_event_handler`(`pmix_status_t` codes[], `size_t` ncodes,

15

`pmix_info_t` info[], `size_t` ninfo,

16

`pmix_notification_fn_t` evhdlr,

17

`pmix_hdlr_reg_cbfunc_t` cbfunc,

18

void \*cbdata);

▲ C ▲

19 *PMIx v2.0*

### [pmix\\_event\\_notification\\_cbfunc\\_fn\\_t](#)

```

1 typedef void (*pmix_event_notification_cbfunc_fn_t)
2     (pmix_status_t status,
3      pmix_info_t *results, size_t nresults,
4      pmix_op_cbfunc_t cbfunc, void *thiscbdata,
5      void *notification_cbdata);

```

## Related Constants

```

6 PMIX\_MODEL\_DECLARED
7 PMIX\_MODEL\_RESOURCES
8 PMIX\_OPENMP\_PARALLEL\_ENTERED
9 PMIX\_OPENMP\_PARALLEL\_EXITED
10 PMIX\_EVENT\_ACTION\_COMPLETE

```

### B.3.2.3 Coordinating at Runtime with Multiple Event Handlers

Coordinating with a threading library such as an OpenMP runtime library creates the need for separate event handlers for threads of the same process. For example in an MPI+OpenMP hybrid application, the MPI main thread and the OpenMP primary thread may both want to be notified anytime an OpenMP thread starts executing in a parallel region. This requires support for multiple threads to potentially register different event handlers against the same status code.

Multiple event handlers registered against the same event are processed in a chain-like manner based on the order in which they were registered, as modified by any directives. Registrations against specific event codes are processed first, followed by registrations against multiple event codes and then any default registrations. At each point in the chain, an event handler is called by the PMIx progress thread and given a function to call when that handler has completed its operation. The handler callback notifies PMIx that the handler is done, returning a status code to indicate the result of its work. The results are appended to the array of prior results, with the returned values combined into an array within a single `pmix_info_t` as follows:

- `array[0]`: the event handler name provided at registration (may be an empty field if a string name was not given) will be in the key, with the `pmix_status_t` value returned by the handler
- `array[*]`: the array of results returned by the handler, if any.

The current PMIx standard does not actually specify a default ordering for event handlers as they are being registered. However, it does include an inherent ordering for invocation. Specifically, PMIx stipulates that handlers be called in the following categorical order:

- single status event handlers - handlers that were registered against a single specific status.
- multi status event handlers - those registered against more than one specific status.
- default event handlers - those registered against no specific status.

1  
2

## Code Example

From the OpenMP primary thread:

```
1 static void parallel_region_OMP_cb(size_t evhdlr_registration_id,
2     pmix_status_t status,
3     const pmix_proc_t *source,
4     pmix_info_t info[], size_t ninfo,
5     pmix_info_t results[], size_t
6     ↪ nresults,
7     pmix_event_notification_cbfunc_fn_t
8     ↪ cbfunc,
9     void *cbdata) {
10 printf("Entered %s\n", __FUNCTION__);
11 /* do what we need OpenMP to do on entering a parallel region */
12 if (NULL != cbfunc) {
13     /* tell the event handler that we are only a partial step */
14     cbfunc(PMIX_EVENT_PARTIAL_ACTION_TAKEN, NULL, 0, NULL, NULL,
15     ↪ cbdata);
16 }
17 }
18
19 bool is_true = true;
20 pmix_status_t code = PMIX_OPENMP_PARALLEL_ENTERED;
21 PMIX_INFO_CREATE(info, 2);
22 PMIX_INFO_LOAD(&info[0], PMIX_EVENT_HDLR_NAME, "OpenMP-Primary",
23     ↪ PMIX_STRING);
24 PMIX_INFO_LOAD(&info[1], PMIX_EVENT_HDLR_FIRST, &is_true, PMIX_BOOL);
25 rc = PMIX_Register_event_handler(&code, 1, info, 2,
26     ↪ parallel_region_OMP_cb, NULL, NULL);
27 if (rc < 0)
28     fprintf(stderr, "%s: Failed to register event handler for OpenMP
29     ↪ region entrance\n", __FUNCTION__);
30 PMIX_INFO_FREE(info, 2);
```

3  
4

From the MPI process:

```
1 static void parallel_region_MPI_cb(size_t evhdlr_registration_id,
2     pmix_status_t status,
3     const pmix_proc_t *source,
4     pmix_info_t info[], size_t ninfo,
5     pmix_info_t results[], size_t
6     ↪ nresults,
7     pmix_event_notification_cbfunc_fn_t
8     ↪ cbfunc,
9     void *cbdata) {
```



```

8  printf("Entered %s\n", __FUNCTION__);
9  /* do what we need the MPI library to do on entering a parallel region
   ↪ */
10 if (NULL != cbfunc) {
11     /* tell the event handler that we are the last step */
12     cbfunc(PMIX_EVENT_ACTION_COMPLETE, NULL, 0, NULL, NULL, cbdata);
13 }
14 }
15
16 pmix_status_t code = PMIX_OPENMP_PARALLEL_ENTERED;
17 PMIX_INFO_CREATE(info, 2);
18 PMIX_INFO_LOAD(&info[0], PMIX_EVENT_HDLR_NAME, "MPI-Thread",
   ↪ PMIX_STRING);
19 PMIX_INFO_LOAD(&info[1], PMIX_EVENT_HDLR_AFTER, "OpenMP-Primary",
   ↪ PMIX_STRING);
20 rc = PMIx_Register_event_handler(&code, 1, info, 2,
   ↪ parallel_region_MPI_cb, NULL, NULL);
21 if (rc < 0)
22     fprintf(stderr, "%s: Failed to register event handler for OpenMP
   ↪ region entrance\n", __FUNCTION__);
23 PMIX_INFO_FREE(info, 2);

```

1

## 2 Related Interfaces

3 *PMIx v2.0* [PMIx\\_Register\\_event\\_handler](#)



4

```

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);

```



10 *PMIx v2.0* [pmix\\_event\\_notification\\_cbfunc\\_fn\\_t](#)



11

```

typedef void (*pmix_event_notification_cbfunc_fn_t)
    (pmix_status_t status,
     pmix_info_t *results, size_t nresults,
     pmix_op_cbfunc_t cbfunc, void *thiscbdata,
     void *notification_cbdata);

```



15

## Related Attributes

**PMIX\_EVENT\_HDLR\_NAME** "pmix.evname" (char\*)

String name identifying this handler.

**PMIX\_EVENT\_HDLR\_FIRST** "pmix.evfirst" (bool)

Invoke this event handler before any other handlers.

**PMIX\_EVENT\_HDLR\_LAST** "pmix.evlast" (bool)

Invoke this event handler after all other handlers have been called.

**PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY** "pmix.evfirstcat" (bool)

Invoke this event handler before any other handlers in this category.

**PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY** "pmix.evlastcat" (bool)

Invoke this event handler after all other handlers in this category have been called.

**PMIX\_EVENT\_HDLR\_BEFORE** "pmix.evbefore" (char\*)

Put this event handler immediately before the one specified in the (char\*) value.

**PMIX\_EVENT\_HDLR\_AFTER** "pmix.evafter" (char\*)

Put this event handler immediately after the one specified in the (char\*) value.

**PMIX\_EVENT\_HDLR\_APPEND** "pmix.evappend" (bool)

Append this handler to the precedence list within its category.

## Related Constants

**PMIX\_EVENT\_NO\_ACTION\_TAKEN**

**PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN**

**PMIX\_EVENT\_ACTION\_DEFERRED**

## B.4 MPI Sessions

### B.4.1 Use Case Summary

MPI Sessions addresses a number of the limitations of the current MPI programming model.

Among the immediate problems MPI Sessions is intended to address are the following:

- MPI cannot be initialized within an MPI process from different application components without a priori knowledge or coordination,
- MPI cannot be initialized more than once, and MPI cannot be reinitialized after MPI finalize has been called.
- With MPI Sessions, an application no longer needs to explicitly call **MPI\_Init** to make use of MPI, but rather can use a Session to only initialize MPI resources for specific communication needs.

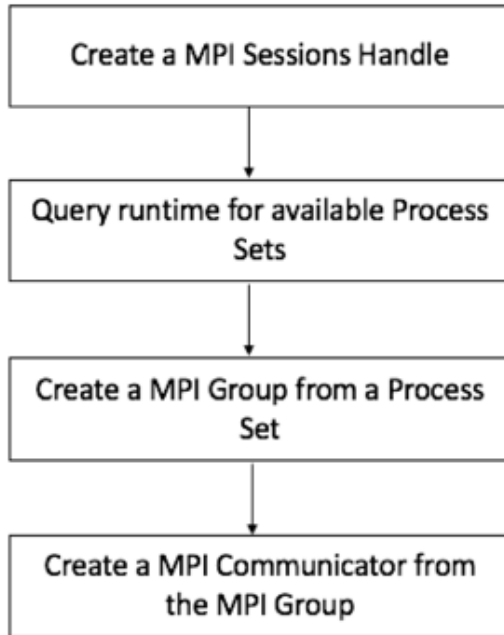


Figure B.4.: MPI Communicator from MPI Session Handle using PMIx

- Unless the MPI process explicitly calls `MPI_Init`, there is also no explicit `MPI_COMM_WORLD` communicator. Sessions can be created and destroyed multiple times in an MPI process.

### B.4.2 Use Case Details

A PMIx Process Set (PSET) is a user-provided or host environment assigned label associated with a given set of application processes. Processes can belong to multiple process sets at a time. Definition of a PMIx process set typically occurs at time of application execution - e.g., on a command line: `prun -n 4 -pset ocean myoceanapp : -n 3 -pset ice myiceapp`

PMIx PSETs are used for query functions (`MPI_SESSION_GET_NUM_PSETS`, `MPI_SESSION_GET_NTH_PSET`) and to create `MPI_GROUP` from a process set name.

In OpenMPI's MPI Sessions prototype, PMIx groups are used during creation of `MPI_COMM` from an `MPI_GROUP`. The PMIx group constructor returns a 64-bit PMIx Group Context Identifier (PGCID) that is guaranteed to be unique for the duration of an allocation (in the case of a batch managed environment). This PGCID could be used as a direct replacement for the existing unique identifiers for communicators in MPI (E.g. Communicator Identifiers (CIDs) in Open MPI), but may have performance implications.

1 There is an important distinction between process sets and process groups. The process set  
2 identifiers are set by the host environment and currently there are no PMIx APIs provided by which  
3 an application can change a process set membership. In contrast, PMIx process groups can only be  
4 defined dynamically by the application.

## 5 Related Interfaces

### 6 *PMIx v1.0* **PMIx\_Get**

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

### 11 *PMIx v4.0* **PMIx\_Group\_construct**

12 `pmix_status_t`  
13 `PMIx_Group_construct(const char grp[],`  
14 `const pmix_proc_t procs[], size_t nprocs,`  
15 `const pmix_info_t directives[],`  
16 `size_t ndirs,`  
17 `pmix_info_t **results,`  
18 `size_t *nresults);`

## 19 Related Attributes

20 **PMIX\_PSET\_NAMES** "pmix.pset.nms" (`pmix_data_array_t*`)

21 Returns an array of `char*` string names of the process sets in which the given process is a  
22 member.

23 **PMIX\_QUERY\_NUM\_GROUPS** "pmix.qry.pgrpnum" (`size_t`)

24 Return the number of process groups defined in the specified range (defaults to session).  
25 OPTIONAL QUALIFIERS: **PMIX\_RANGE**.

26 **PMIX\_QUERY\_GROUP\_NAMES** "pmix.qry.pgrp" (`pmix_data_array_t*`)

27 Return a `pmix_data_array_t` containing an array of string names of the process groups  
28 defined in the specified range (defaults to session). OPTIONAL QUALIFIERS:

29 **PMIX\_RANGE**.

30 **PMIX\_QUERY\_GROUP\_MEMBERSHIP**

31 "pmix.qry.pgrpmems" (`pmix_data_array_t*`)

32 Return a `pmix_data_array_t` of `pmix_proc_t` containing the members of the  
33 specified process group. REQUIRED QUALIFIERS: **PMIX\_GROUP\_ID**.

1  
2

## Related Constants

`PMIX_SUCCESS` `PMIX_ERR_NOT_SUPPORTED`

Unofficial Draft

## APPENDIX C

# Revision History

---

## 1 C.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 C.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 C.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  
24 `PMIx_Deregister_errhandler` APIs were replaced. This pre-dated official adoption of  
25 PMIx as a Standard.

### 26 C.2.2 Deprecated constants

27 The following constants were deprecated in v2.0:

- 28 `PMIX_MODEX`
- 29 `PMIX_INFO_ARRAY`

## 1 C.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 C.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\\_namespace](#) 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 C.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 C.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 C.5.1 Removed constants

6 The following constants were removed in v3.0:

7 `PMIX_MODEX`  
8 `PMIX_INFO_ARRAY`

## 9 C.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 C.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 C.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
- 14 (see 6.1)
- 15 • Add new attributes to assemble information by its level for storage where ambiguity may exist
- 16 (see 17.2.3.1)
- 17 • Add flag and `PMIX_INFO_IS_END` macro for marking and detecting the end of a
- 18 `pmix_info_t` array
- 19 • Clarify that `PMIX_NUM_SLOTS` is duplicative of (a) `PMIX_UNIV_SIZE` when used at the
- 20 *session* level and (b) `PMIX_MAX_PROCS` when used at the *job* and *application* levels, but leave
- 21 it in for backward compatibility.
- 22 • Clarify difference between `PMIX_JOB_SIZE` and `PMIX_MAX_PROCS`
- 23 • Clarify that `PMIx_server_setup_application` must be called per-*job* instead of
- 24 per-*application* as the name implies. Unfortunately, this is a historical artifact. Note that both
- 25 `PMIX_NODE_MAP` and `PMIX_PROC_MAP` must be included as input in the *info* array provided
- 26 to that function. Further descriptive explanation of the “instant on” procedure will be provided in
- 27 the next version of the PMIx Standard.
- 28 • Clarify how the PMIx server expects data passed to the host by
- 29 `pmix_server_fencenb_fn_t` should be aggregated across nodes, and provide a code
- 30 snippet example

## 31 C.7 Version 3.2: Oct. 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.

## C.7.1 Deprecated constants

The following constants were deprecated in v3.2:

<code>PMIX_ERR_DATA_VALUE_NOT_FOUND</code>	Data value not found
<code>PMIX_ERR_HANDSHAKE_FAILED</code>	Connection handshake failed
<code>PMIX_ERR_IN_ERRNO</code>	Error defined in <code>errno</code>
<code>PMIX_ERR_INVALID_ARG</code>	Invalid argument
<code>PMIX_ERR_INVALID_ARGS</code>	Invalid arguments
<code>PMIX_ERR_INVALID_KEY</code>	Invalid key
<code>PMIX_ERR_INVALID_KEY_LENGTH</code>	Invalid key length
<code>PMIX_ERR_INVALID_KEYVALP</code>	Invalid key/value pair
<code>PMIX_ERR_INVALID_LENGTH</code>	Invalid argument length
<code>PMIX_ERR_INVALID_NAMESPACE</code>	Invalid namespace
<code>PMIX_ERR_INVALID_NUM_ARGS</code>	Invalid number of arguments
<code>PMIX_ERR_INVALID_NUM_PARSED</code>	Invalid number parsed
<code>PMIX_ERR_INVALID_SIZE</code>	Invalid size
<code>PMIX_ERR_INVALID_VAL</code>	Invalid value
<code>PMIX_ERR_INVALID_VAL_LENGTH</code>	Invalid value length
<code>PMIX_ERR_NOT_IMPLEMENTED</code>	Not implemented
<code>PMIX_ERR_PACK_MISMATCH</code>	Pack mismatch
<code>PMIX_ERR_PROC_ENTRY_NOT_FOUND</code>	Process not found
<code>PMIX_ERR_PROC_REQUESTED_ABORT</code>	Process is already requested to abort
<code>PMIX_ERR_READY_FOR_HANDSHAKE</code>	Ready for handshake
<code>PMIX_ERR_SERVER_FAILED_REQUEST</code>	Failed to connect to the server
<code>PMIX_ERR_SERVER_NOT_AVAIL</code>	Server is not available
<code>PMIX_ERR_SILENT</code>	Silent error
<code>PMIX_GDS_ACTION_COMPLETE</code>	The Global Data Storage (GDS) action has completed
<code>PMIX_NOTIFY_ALLOC_COMPLETE</code>	Notify that a requested allocation operation is complete - the result of the request will be included in the <i>info</i> array

## 1 C.7.2 Deprecated attributes

2 The following attributes were deprecated in v3.2:

3 **PMIX\_ARCH** "pmix.arch" (uint32\_t)

4 Architecture flag.

5 **PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (char\*)

6 Comma-delimited list of algorithms to use for the collective operation. PMIx does not  
7 impose any requirements on a host environment's collective algorithms. Thus, the  
8 acceptable values for this attribute will be environment-dependent - users are encouraged to  
9 check their host environment for supported values.

10 **PMIX\_DSTPATH** "pmix.dstpath" (char\*)

11 Path to shared memory data storage (dstore) files. Deprecated from Standard as being  
12 implementation specific.

13 **PMIX\_HWLOC\_HOLE\_KIND** "pmix.hwlocholek" (char\*)

14 Kind of VM "hole" HWLOC should use for shared memory

15 **PMIX\_HWLOC\_SHARE\_TOPO** "pmix.hwlocsh" (bool)

16 Share the HWLOC topology via shared memory

17 **PMIX\_HWLOC\_SHMEM\_ADDR** "pmix.hwlocaddr" (size\_t)

18 Address of the HWLOC shared memory segment.

19 **PMIX\_HWLOC\_SHMEM\_FILE** "pmix.hwlocfile" (char\*)

20 Path to the HWLOC shared memory file.

21 **PMIX\_HWLOC\_SHMEM\_SIZE** "pmix.hwlocsize" (size\_t)

22 Size of the HWLOC shared memory segment.

23 **PMIX\_HWLOC\_XML\_V1** "pmix.hwlocxml1" (char\*)

24 XML representation of local topology using HWLOC's v1.x format.

25 **PMIX\_HWLOC\_XML\_V2** "pmix.hwlocxml2" (char\*)

26 XML representation of local topology using HWLOC's v2.x format.

27 **PMIX\_LOCAL\_TOPO** "pmix.ltopo" (char\*)

28 XML representation of local node topology.

29 **PMIX\_MAPPER** "pmix.mapper" (char\*)

30 Mapping mechanism to use for placing spawned processes - when accessed using  
31 [PMIx\\_Get](#), use the [PMIX\\_RANK\\_WILDCARD](#) value for the rank to discover the mapping  
32 mechanism used for the provided namespace.

33 **PMIX\_MAP\_BLOB** "pmix.mblob" (pmix\_byte\_object\_t)

34 Packed blob of process location.

35 **PMIX\_NON\_PMI** "pmix.nonpmi" (bool)

36 Spawned processes will not call [PMIx\\_Init](#).

37 **PMIX\_PROC\_BLOB** "pmix.pblob" (pmix\_byte\_object\_t)

38 Packed blob of process data.

39 **PMIX\_PROC\_URI** "pmix.puri" (char\*)

40 URI containing contact information for the specified process.

41 **PMIX\_TOPOLOGY\_FILE** "pmix.topo.file" (char\*)

42 Full path to file containing XML topology description

1 **PMIX\_TOPOLOGY\_SIGNATURE** "pmix.toposig" (char\*)

2 Topology signature string.

3 **PMIX\_TOPOLOGY\_XML** "pmix.topo.xml" (char\*)

4 XML-based description of topology

## 5 C.8 Version 4.0: Dec. 2020

6 NOTE: The PMIx Standard document has undergone significant reorganization in an effort to  
7 become more user-friendly. Highlights include:

- 8 • Moving all added, deprecated, and removed items to this revision log section to make them more  
9 visible
- 10 • Co-locating constants and attribute definitions with the primary API that uses them - citations  
11 and hyperlinks are retained elsewhere
- 12 • Splitting the Key-Value Management chapter into separate chapters on the use of reserved keys,  
13 non-reserved keys, and non-process-related key-value data exchange
- 14 • Creating a new chapter on synchronization and data access methods
- 15 • Removing references to specific implementations of PMIx and to implementation-specific  
16 features and/or behaviors

17 In addition to the reorganization, the following changes were introduced in v4.0 of the PMIx  
18 Standard:

- 19 • Clarified that the **PMIx\_Fence\_nb** operation can immediately return  
20 **PMIX\_OPERATION\_SUCCEEDED** in lieu of passing the request to a PMIx server if only the  
21 calling process is involved in the operation
- 22 • Added the **PMIx\_Register\_attributes** API by which a host environment can register the  
23 attributes it supports for each server-to-host operation
- 24 • Added the ability to query supported attributes from the PMIx tool, client and server libraries, as  
25 well as the host environment via the new **pmix\_regattr\_t** structure. Both human-readable  
26 and machine-parsable output is supported. New attributes to support this operation include:
  - 27 – **PMIX\_CLIENT\_ATTRIBUTES**, **PMIX\_SERVER\_ATTRIBUTES**,
  - 28 **PMIX\_TOOL\_ATTRIBUTES**, and **PMIX\_HOST\_ATTRIBUTES** to identify which library  
29 supports the attribute; and
  - 30 – **PMIX\_MAX\_VALUE**, **PMIX\_MIN\_VALUE**, and **PMIX\_ENUM\_VALUE** to provide  
31 machine-parsable description of accepted values
- 32 • Add **PMIX\_APP\_WILDCARD** to reference all applications within a given job
- 33 • Fix signature of blocking APIs **PMIx\_Allocation\_request**, **PMIx\_Job\_control**,  
34 **PMIx\_Process\_monitor**, **PMIx\_Get\_credential**, and  
35 **PMIx\_Validate\_credential** to allow return of results
- 36 • Update description to provide an option for blocking behavior of the  
37 **PMIx\_Register\_event\_handler**, **PMIx\_Deregister\_event\_handler**,  
38 **PMIx\_Notify\_event**, **PMIx\_IOF\_pull**, **PMIx\_IOF\_deregister**, and  
39 **PMIx\_IOF\_push** APIs. The need for blocking forms of these functions was not initially

1 anticipated but has emerged over time. For these functions, the return value is sufficient to  
2 provide the caller with information otherwise returned via callback. Thus, use of a **NULL** value  
3 as the callback function parameter was deemed a minimal disruption method for providing the  
4 desired capability

- 5 • Added a chapter on fabric support that includes new APIs, datatypes, and attributes
- 6 • Added a chapter on process sets and groups that includes new APIs and attributes
- 7 • Added APIs and a new datatypes to support generation and parsing of PMIx locality and cpuset  
8 strings
- 9 • Added a new chapter on tools that provides deeper explanation on their operation and collecting  
10 all tool-relevant definitions into one location. Also introduced two new APIs and removed  
11 restriction that limited tools to being connected to only one server at a time.
- 12 • Extended behavior of `PMIx_server_init` to scalably expose the topology description to the  
13 local clients. This includes creating any required shared memory backing stores and/or XML  
14 representations, plus ensuring that all necessary key-value pairs for clients to access the  
15 description are included in the job-level information provided to each client.
- 16 • Added a new API by which the host can manually progress the PMIx library in lieu of the  
17 library's own progress thread. s

18 The above changes included introduction of the following APIs and data types:

19 • Client APIs

- 20 – `PMIx_Group_construct`, `PMIx_Group_construct_nb`
- 21 – `PMIx_Group_destruct`, `PMIx_Group_destruct_nb`
- 22 – `PMIx_Group_invite`, `PMIx_Group_invite_nb`
- 23 – `PMIx_Group_join`, `PMIx_Group_join_nb`
- 24 – `PMIx_Group_leave`, `PMIx_Group_leave_nb`
- 25 – `PMIx_Get_relative_locality`, `PMIx_Load_topology`
- 26 – `PMIx_Parse_cpuset_string`, `PMIx_Get_cpuset`
- 27 – `PMIx_Link_state_string`, `PMIx_Job_state_string`
- 28 – `PMIx_Device_type_string`
- 29 – `PMIx_Fabric_register`, `PMIx_Fabric_register_nb`
- 30 – `PMIx_Fabric_update`, `PMIx_Fabric_update_nb`
- 31 – `PMIx_Fabric_deregister`, `PMIx_Fabric_deregister_nb`
- 32 – `PMIx_Compute_distances`, `PMIx_Compute_distances_nb`
- 33 – `PMIx_Get_attribute_string`, `PMIx_Get_attribute_name`
- 34 – `PMIx_Progress`

35 • Server APIs

- 36 – `PMIx_server_generate_locality_string`
- 37 – `PMIx_Register_attributes`
- 38 – `PMIx_server_define_process_set`, `PMIx_server_delete_process_set`
- 39 – `pmix_server_grp_fn_t`, `pmix_server_fabric_fn_t`
- 40 – `pmix_server_client_connected2_fn_t`

- 1       - `PMIx_server_generate_cpuset_string`
- 2       - `PMIx_server_register_resources`, `PMIx_server_deregister_resources`
- 3       • Tool APIs
- 4       - `PMIx_tool_disconnect`
- 5       - `PMIx_tool_set_server`
- 6       - `PMIx_tool_attach_to_server`
- 7       - `PMIx_tool_get_servers`
- 8       • Data types
- 9       - `pmix_regattr_t`
- 10      - `pmix_cpuset_t`
- 11      - `pmix_topology_t`
- 12      - `pmix_locality_t`
- 13      - `pmix_bind_envelope_t`
- 14      - `pmix_group_opt_t`
- 15      - `pmix_group_operation_t`
- 16      - `pmix_fabric_t`
- 17      - `pmix_device_distance_t`
- 18      - `pmix_coord_t`
- 19      - `pmix_coord_view_t`
- 20      - `pmix_geometry_t`
- 21      - `pmix_link_state_t`
- 22      - `pmix_job_state_t`
- 23      - `pmix_device_type_t`
- 24      • Callback functions
- 25      - `pmix_device_dist_cbfunc_t`

## 26 **C.8.1 Added Constants**

### 27 **General error constants**

- 28 `PMIX_ERR_EXISTS_OUTSIDE_SCOPE`
- 29 `PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED`
- 30 `PMIX_ERR_EMPTY`
- 31



1           **Data type constants**

2           PMIX\_COORD  
3           PMIX\_REGATTR  
4           PMIX\_REGEX  
5           PMIX\_JOB\_STATE  
6           PMIX\_LINK\_STATE  
7           PMIX\_PROC\_CPuset  
8           PMIX\_GEOMETRY  
9           PMIX\_DEVICE\_DIST  
10          PMIX\_ENDPOINT  
11          PMIX\_TOPO  
12          PMIX\_DEVTYPE  
13          PMIX\_LOCTYPE  
14          PMIX\_DATA\_TYPE\_MAX  
15          PMIX\_COMPRESSED\_BYTE\_OBJECT

17          **Info directives**

18          PMIX\_INFO\_REQD\_PROCESSED  
19

20          **Server constants**

21          PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION  
22

23          **Job-Mgmt constants**

24          PMIX\_ERR\_CONFLICTING\_CLEANUP\_DIRECTIVES  
25

26          **Publish constants**

27          PMIX\_ERR\_DUPLICATE\_KEY  
28

29          **Tool constants**

30          PMIX\_LAUNCHER\_READY  
31          PMIX\_ERR\_IOF\_FAILURE  
32          PMIX\_ERR\_IOF\_COMPLETE  
33          PMIX\_EVENT\_JOB\_START  
34          PMIX\_LAUNCH\_COMPLETE  
35          PMIX\_EVENT\_JOB\_END  
36          PMIX\_EVENT\_SESSION\_START  
37          PMIX\_EVENT\_SESSION\_END  
38          PMIX\_ERR\_PROC\_TERM\_WO\_SYNC  
39          PMIX\_ERR\_JOB\_CANCELED  
40          PMIX\_ERR\_JOB\_ABORTED

1 PMIX\_ERR\_JOB\_KILLED\_BY\_CMD  
2 PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG  
3 PMIX\_ERR\_JOB\_TERM\_WO\_SYNC  
4 PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED  
5 PMIX\_ERR\_JOB\_NON\_ZERO\_TERM  
6 PMIX\_ERR\_JOB\_ABORTED\_BY\_SYS\_EVENT  
7 PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY  
8 PMIX\_DEBUGGER\_RELEASE  
9

### 10 **Fabric constants**

11 PMIX\_FABRIC\_UPDATE\_PENDING  
12 PMIX\_FABRIC\_UPDATED  
13 PMIX\_FABRIC\_UPDATE\_ENDPOINTS  
14 PMIX\_COORD\_VIEW\_UNDEF  
15 PMIX\_COORD\_LOGICAL\_VIEW  
16 PMIX\_COORD\_PHYSICAL\_VIEW  
17 PMIX\_LINK\_STATE\_UNKNOWN  
18 PMIX\_LINK\_DOWN  
19 PMIX\_LINK\_UP  
20 PMIX\_FABRIC\_REQUEST\_INFO  
21 PMIX\_FABRIC\_UPDATE\_INFO  
22

### 23 **Sets-Groups constants**

24 PMIX\_PROCESS\_SET\_DEFINE  
25 PMIX\_PROCESS\_SET\_DELETE  
26 PMIX\_GROUP\_INVITED  
27 PMIX\_GROUP\_LEFT  
28 PMIX\_GROUP\_MEMBER\_FAILED  
29 PMIX\_GROUP\_INVITE\_ACCEPTED  
30 PMIX\_GROUP\_INVITE\_DECLINED  
31 PMIX\_GROUP\_INVITE\_FAILED  
32 PMIX\_GROUP\_MEMBERSHIP\_UPDATE  
33 PMIX\_GROUP\_CONSTRUCT\_ABORT  
34 PMIX\_GROUP\_CONSTRUCT\_COMPLETE  
35 PMIX\_GROUP\_LEADER\_FAILED  
36 PMIX\_GROUP\_LEADER\_SELECTED  
37 PMIX\_GROUP\_CONTEXT\_ID\_ASSIGNED  
38

### 39 **Process-Mgmt constants**

40 PMIX\_ERR\_JOB\_ALLOC\_FAILED  
41 PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE

1 `PMIX_ERR_JOB_NO_EXE_SPECIFIED`  
2 `PMIX_ERR_JOB_FAILED_TO_MAP`  
3 `PMIX_ERR_JOB_FAILED_TO_LAUNCH`  
4 `PMIX_LOCALITY_UNKNOWN`  
5 `PMIX_LOCALITY_NONLOCAL`  
6 `PMIX_LOCALITY_SHARE_HWTHREAD`  
7 `PMIX_LOCALITY_SHARE_CORE`  
8 `PMIX_LOCALITY_SHARE_L1CACHE`  
9 `PMIX_LOCALITY_SHARE_L2CACHE`  
10 `PMIX_LOCALITY_SHARE_L3CACHE`  
11 `PMIX_LOCALITY_SHARE_PACKAGE`  
12 `PMIX_LOCALITY_SHARE_NUMA`  
13 `PMIX_LOCALITY_SHARE_NODE`  
14

## 15 Events

16 `PMIX_EVENT_SYS_BASE`  
17 `PMIX_EVENT_NODE_DOWN`  
18 `PMIX_EVENT_NODE_OFFLINE`  
19 `PMIX_EVENT_SYS_OTHER`  
20

## 21 C.8.2 Added Attributes

### 22 Sync-Access attributes

23 `PMIX_COLLECT_GENERATED_JOB_INFO` "pmix.collect.gen" (bool)

24 Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx  
25 servers. Some job-level information (e.g., distance between processes and fabric devices) is  
26 best determined on a distributed basis as it primarily pertains to local processes. Should  
27 remote processes need to access the information, it can either be obtained collectively using  
28 the `PMIx_Fence` operation with this directive, or can be retrieved one peer at a time using  
29 `PMIx_Get` without first having performed the job-wide collection.

30 `PMIX_ALL_CLONES_PARTICIPATE` "pmix.clone.part" (bool)

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

32 `PMIX_GET_POINTER_VALUES` "pmix.get.pntrs" (bool)

33 Request that any pointers in the returned value point directly to values in the key-value store.  
34 The user *must not* release any returned data pointers.

35 `PMIX_GET_STATIC_VALUES` "pmix.get.static" (bool)

36 Request that the data be returned in the provided storage location. The caller is responsible  
37 for destructing the `pmix_value_t` using the `PMIX_VALUE_DESTRUCT` macro when  
38 done.

39 `PMIX_GET_REFRESH_CACHE` "pmix.get.refresh" (bool)

1 When retrieving data for a remote process, refresh the existing local data cache for the  
2 process in case new values have been put and committed by the process since the last refresh.  
3 Local process information is assumed to be automatically updated upon posting by the  
4 process. A **NULL** key will cause all values associated with the process to be refreshed -  
5 otherwise, only the indicated key will be updated. A process rank of  
6 **PMIX\_RANK\_WILDCARD** can be used to update job-related information in dynamic  
7 environments. The user is responsible for subsequently updating refreshed values they may  
8 have cached in their own local memory.

9 **PMIX\_QUERY\_RESULTS** "pmix.qry.res" (pmix\_data\_array\_t)

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

16 **PMIX\_QUERY\_QUALIFIERS** "pmix.qry.quals" (pmix\_data\_array\_t)

17 Contains an array of qualifiers that were included in the query that produced the provided  
18 results. This attribute is solely for reporting purposes and cannot be used in **PMIx\_Get** or  
19 other query operations.

20 **PMIX\_QUERY\_SUPPORTED\_KEYS** "pmix.qry.keys" (char\*)

21 Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS.

22 **PMIX\_QUERY\_SUPPORTED\_QUALIFIERS** "pmix.qry.quals" (char\*)

23 Return comma-delimited list of qualifiers supported by a query on the provided key, instead  
24 of actually performing the query on the key. NO QUALIFIERS.

25 **PMIX\_QUERY\_NAMESPACE\_INFO** "pmix.qry.nsinfo" (pmix\_data\_array\_t\*)

26 Return an array of active namespace information - each element will itself contain an array  
27 including the namespace plus the command line of the application executing within it.  
28 OPTIONAL QUALIFIERS: **PMIX\_NAMESPACE** of specific namespace whose info is being  
29 requested.

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

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

34 **PMIX\_QUERY\_AVAIL\_SERVERS** "pmix.qry.asrvrs" (pmix\_data\_array\_t\*)

35 Return an array of **pmix\_info\_t**, each element itself containing a  
36 **PMIX\_SERVER\_INFO\_ARRAY** entry holding all available data for a server on this node to  
37 which the caller might be able to connect.

38 **PMIX\_SERVER\_INFO\_ARRAY** "pmix.srv.arr" (pmix\_data\_array\_t)

1 Array of `pmix_info_t` about a given server, starting with its `PMIX_NAMESPACE` and  
2 including at least one of the rendezvous-required pieces of information.

3 **PMIX\_CLIENT\_FUNCTIONS** "`pmix.client.fns`" (`bool`)

4 Request a list of functions supported by the PMIx client library.

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

6 Request attributes supported by the PMIx client library.

7 **PMIX\_SERVER\_FUNCTIONS** "`pmix.srvr.fns`" (`bool`)

8 Request a list of functions supported by the PMIx server library.

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

10 Request attributes supported by the PMIx server library.

11 **PMIX\_HOST\_FUNCTIONS** "`pmix.srvr.fns`" (`bool`)

12 Request a list of functions supported by the host environment.

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

14 Request attributes supported by the host environment.

15 **PMIX\_TOOL\_FUNCTIONS** "`pmix.tool.fns`" (`bool`)

16 Request a list of functions supported by the PMIx tool library.

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

18 Request attributes supported by the PMIx tool library functions.

### 19 Server attributes

20 **PMIX\_TOPOLOGY2** "`pmix.topo2`" (`pmix_topology_t`)

21 Provide a pointer to an implementation-specific description of the local node topology.

22 **PMIX\_SERVER\_SHARE\_TOPOLOGY** "`pmix.srvr.share`" (`bool`)

23 The PMIx server is to share its copy of the local node topology (whether given to it or  
24 self-discovered) with any clients.

25 **PMIX\_SERVER\_SESSION\_SUPPORT** "`pmix.srvr.sess`" (`bool`)

26 The host RM wants to declare itself as being the local session server for PMIx connection  
27 requests.

28 **PMIX\_SERVER\_START\_TIME** "`pmix.srvr.strtime`" (`char*`)

29 Time when the server started - i.e., when the server created its rendezvous file (given in  
30 ctime string format).

31 **PMIX\_SERVER\_SCHEDULER** "`pmix.srv.sched`" (`bool`)

32 Server is supporting system scheduler and desires access to appropriate WLM-supporting  
33 features. Indicates that the library is to be initialized for scheduler support.

34 **PMIX\_JOB\_INFO\_ARRAY** "`pmix.job.arr`" (`pmix_data_array_t`)

1 Provide an array of `pmix_info_t` containing job-realm information. The  
2 `PMIX_SESSION_ID` attribute of the `session` containing the `job` is required to be included in  
3 the array whenever the PMIx server library may host multiple sessions (e.g., when executing  
4 with a host RM daemon). As information is registered one job (aka namespace) at a time via  
5 the `PMIx_server_register_namespace` API, there is no requirement that the array  
6 contain either the `PMIX_NAMESPACE` or `PMIX_JOBID` attributes when used in that context  
7 (though either or both of them may be included). At least one of the job identifiers must be  
8 provided in all other contexts where the job being referenced is ambiguous.

9 **`PMIX_APP_INFO_ARRAY`** "`pmix.app.arr`" (`pmix_data_array_t`)

10 Provide an array of `pmix_info_t` containing application-realm information. The  
11 `PMIX_NAMESPACE` or `PMIX_JOBID` attributes of the `job` containing the application, plus its  
12 `PMIX_APPNUM` attribute, must to be included in the array when the array is *not* included as  
13 part of a call to `PMIx_server_register_namespace` - i.e., when the job containing the  
14 application is ambiguous. The job identification is otherwise optional.

15 **`PMIX_PROC_INFO_ARRAY`** "`pmix.pdata`" (`pmix_data_array_t`)

16 Provide an array of `pmix_info_t` containing process-realm information. The  
17 `PMIX_RANK` and `PMIX_NAMESPACE` attributes, or the `PMIX_PROCID` attribute, are required  
18 to be included in the array when the array is not included as part of a call to  
19 `PMIx_server_register_namespace` - i.e., when the job containing the process is  
20 ambiguous. All three may be included if desired. When the array is included in some  
21 broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID`  
22 attribute must be included (the others are optional).

23 **`PMIX_NODE_INFO_ARRAY`** "`pmix.node.arr`" (`pmix_data_array_t`)

24 Provide an array of `pmix_info_t` containing node-realm information. At a minimum,  
25 either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the  
26 array, though both may be included.

27 **`PMIX_MAX_VALUE`** "`pmix.descr.maxval`" (*varies*)

28 Used in `pmix_regattr_t` to describe the maximum valid value for the associated  
29 attribute.

30 **`PMIX_MIN_VALUE`** "`pmix.descr.minval`" (*varies*)

31 Used in `pmix_regattr_t` to describe the minimum valid value for the associated  
32 attribute.

33 **`PMIX_ENUM_VALUE`** "`pmix.descr.enum`" (`char*`)

34 Used in `pmix_regattr_t` to describe accepted values for the associated attribute.  
35 Numerical values shall be presented in a form convertible to the attribute's declared data  
36 type. Named values (i.e., values defined by constant names via a typical C-language enum  
37 declaration) must be provided as their numerical equivalent.

38 **`PMIX_HOMOGENEOUS_SYSTEM`** "`pmix.homo`" (`bool`)

39 The nodes comprising the session are homogeneous - i.e., they each contain the same  
40 number of identical packages, fabric interfaces, GPUs, and other devices.

1 **PMIX\_REQUIRED\_KEY** "pmix.req.key" (char\*)

2 Identifies a key that must be included in the requested information. If the specified key is not  
3 already available, then the PMIx servers are required to delay response to the dmodex  
4 request until either the key becomes available or the request times out.

## 5 Job-Mgmt 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\_QUEUE** "pmix.alloc.queue" (char\*)

10 Name of the WLM queue to which the allocation request is to be directed, or the queue being  
11 referenced in a query.

## 12 Publish attributes

13 **PMIX\_ACCESS\_PERMISSIONS** "pmix.aperms" (pmix\_data\_array\_t)

14 Define access permissions for the published data. The value shall contain an array of  
15 **pmix\_info\_t** structs containing the specified permissions.

16 **PMIX\_ACCESS\_USERIDS** "pmix.auids" (pmix\_data\_array\_t)

17 Array of effective UIDs that are allowed to access the published data.

18 **PMIX\_ACCESS\_GRPIDS** "pmix.agids" (pmix\_data\_array\_t)

19 Array of effective GIDs that are allowed to access the published data.

## 20 Reserved keys

21 **PMIX\_NUM\_ALLOCATED\_NODES** "pmix.num.anodes" (uint32\_t)

22 Number of nodes in the specified realm regardless of whether or not they currently host  
23 processes. Defaults to the *job* realm.

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

25 Number of nodes currently hosting processes in the specified realm. Defaults to the *job*  
26 realm.

27 **PMIX\_CMD\_LINE** "pmix.cmd.line" (char\*)

28 Command line used to execute the specified job (e.g., "mpirun -n 2 -map-by foo ./myapp : -n  
29 4 ./myapp2").

30 **PMIX\_APP\_ARGV** "pmix.app.argv" (char\*)

31 Consolidated argv passed to the spawn command for the given application (e.g., "./myapp  
32 arg1 arg2 arg3").

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

34 Rank of the specified process on the *package* where this process resides - refers to the  
35 numerical location (starting from zero) of the process on its package when counting only  
36 those processes from the same job that share the package, ordered by their overall rank  
37 within that job. Note that processes that are not bound to PUs within a single specific  
38 package cannot have a package rank.

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 the  
3     process has never been restarted. 5

4 **PMIX\_HOSTNAME\_ALIASES** "pmix.alias" (char\*)  
5     Comma-delimited list of names by which the target node is known.

6 **PMIX\_HOSTNAME\_KEEP\_FQDN** "pmix.fqdn" (bool)  
7     FQDNs are being retained by the PMIx library.

8 **PMIX\_CPuset\_BITMAP** "pmix.bitmap" (pmix\_cpuset\_t\*)  
9     Bitmap applied to the process upon launch.

10 **PMIX\_EXTERNAL\_PROGRESS** "pmix.evevt" (bool)  
11     The host shall progress the PMIx library via calls to **PMIx\_Progress**

12 **PMIX\_NODE\_MAP\_RAW** "pmix.nmap.raw" (char\*)  
13     Comma-delimited list of nodes containing procs within the specified realm. Defaults to the  
14     *job* realm.

15 **PMIX\_PROC\_MAP\_RAW** "pmix.pmap.raw" (char\*)  
16     Semi-colon delimited list of strings, each string containing a comma-delimited list of ranks  
17     on the corresponding node within the specified realm. Defaults to the *job* realm.

18 **Tool attributes**

19 **PMIX\_TOOL\_CONNECT\_OPTIONAL** "pmix.tool.conopt" (bool)  
20     The tool shall connect to a server if available, but otherwise continue to operate  
21     unconnected.

22 **PMIX\_TOOL\_ATTACHMENT\_FILE** "pmix.tool.attach" (char\*)  
23     Pathname of file containing connection information to be used for attaching to a specific  
24     server.

25 **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** "pmix.tool.lncrnd" (char\*)  
26     Pathname of file where the launcher is to store its connection information so that the  
27     spawning tool can connect to it.

28 **PMIX\_PRIMARY\_SERVER** "pmix.pri.srvr" (bool)  
29     The server to which the tool is connecting shall be designated the *primary* server once  
30     connection has been accomplished.

31 **PMIX\_NOHUP** "pmix.nohup" (bool)  
32     Any processes started on behalf of the calling tool (or the specified namespace, if such  
33     specification is included in the list of attributes) should continue after the tool disconnects  
34     from its server.

35 **PMIX\_LAUNCHER\_DAEMON** "pmix.lnch.dmn" (char\*)



1 Path to executable that is to be used as the backend daemon for the launcher. This replaces  
2 the launcher's own daemon with the specified executable. Note that the user is therefore  
3 responsible for ensuring compatibility of the specified executable and the host launcher.

4 **PMIX\_FORKEXEC\_AGENT** "pmix.frkex.agnt" (char\*)

5 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
6 application processes. The fork/exec agent shall connect back (as a PMIx tool) to the  
7 launcher's daemon to receive its spawn instructions, and is responsible for starting the actual  
8 application process it replaced. See Section 18.4.3 for details.

9 **PMIX\_EXEC\_AGENT** "pmix.exec.agnt" (char\*)

10 Path to executable that the launcher's backend daemons are to fork/exec in place of the actual  
11 application processes. The launcher's daemon shall pass the full command line of the  
12 application on the command line of the exec agent, which shall not connect back to the  
13 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
14 process in its own place. See Section 18.4.3 for details.

15 **PMIX\_IOF\_PUSH\_STDIN** "pmix.iof.stdin" (bool)

16 Requests that the PMIx library collect the **stdin** of the requester and forward it to the  
17 processes specified in the **PMIx\_IOF\_push** call. All collected data is sent to the same  
18 targets until **stdin** is closed, or a subsequent call to **PMIx\_IOF\_push** is made that  
19 includes the **PMIX\_IOF\_COMPLETE** attribute indicating that forwarding of **stdin** is to be  
20 terminated.

21 **PMIX\_IOF\_COPY** "pmix.iof.cpy" (bool)

22 Requests that the host environment deliver a copy of the specified output stream(s) to the  
23 tool, letting the stream(s) continue to also be delivered to the default location. This allows the  
24 tool to tap into the output stream(s) without redirecting it from its current final destination.

25 **PMIX\_IOF\_REDIRECT** "pmix.iof.redir" (bool)

26 Requests that the host environment intercept the specified output stream(s) and deliver it to  
27 the requesting tool instead of its current final destination. This might be used, for example,  
28 during a debugging procedure to avoid injection of debugger-related output into the  
29 application's results file. The original output stream(s) destination is restored upon  
30 termination of the tool.

31 **PMIX\_DEBUG\_TARGET** "pmix.dbg.tgt" (pmix\_proc\_t\*)

32 Identifier of process(es) to be debugged - a rank of **PMIX\_RANK\_WILDCARD** indicates that  
33 all processes in the specified namespace are to be included.

34 **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** "pmix.dbg.dpproc" (uint16\_t)

35 Number of debugger daemons to be spawned per application process. The launcher is to pass  
36 the identifier of the namespace to be debugged by including the **PMIX\_DEBUG\_TARGET**  
37 attribute in the daemon's job-level information. The debugger daemons spawned on a given  
38 node are responsible for self-determining their specific target process(es) - e.g., by  
39 referencing their own **PMIX\_LOCAL\_RANK** in the daemon debugger job versus the  
40 corresponding **PMIX\_LOCAL\_RANK** of the target processes on the node.

1 **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** "pmix.dbg.dpnd" (uint16\_t)  
2 Number of debugger daemons to be spawned on each node where the target job is executing.  
3 The launcher is to pass the identifier of the namespace to be debugged by including the  
4 **PMIX\_DEBUG\_TARGET** attribute in the daemon's job-level information. The debugger  
5 daemons spawned on a given node are responsible for self-determining their specific target  
6 process(es) - e.g., by referencing their own **PMIX\_LOCAL\_RANK** in the daemon debugger  
7 job versus the corresponding **PMIX\_LOCAL\_RANK** of the target processes on the node.

8 **PMIX\_WAIT\_FOR\_CONNECTION** "pmix.wait.conn" (bool)  
9 Wait until the specified process has connected to the requesting tool or server, or the  
10 operation times out (if the **PMIX\_TIMEOUT** directive is included in the request).

11 **PMIX\_LAUNCH\_DIRECTIVES** "pmix.lnch.dirs" (pmix\_data\_array\_t\*)  
12 Array of **pmix\_info\_t** containing directives for the launcher - a convenience attribute for  
13 retrieving all directives with a single call to **PMIx\_Get**.

14 **Fabric attributes**

15 **PMIX\_SERVER\_SCHEDULER** "pmix.srv.sched" (bool)  
16 Server is supporting system scheduler and desires access to appropriate WLM-supporting  
17 features. Indicates that the library is to be initialized for scheduler support.

18 **PMIX\_FABRIC\_COST\_MATRIX** "pmix.fab.cm" (pointer)  
19 Pointer to a two-dimensional square array of point-to-point relative communication costs  
20 expressed as **uint16\_t** values.

21 **PMIX\_FABRIC\_GROUPS** "pmix.fab.grps" (string)  
22 A string delineating the group membership of nodes in the overall system, where each fabric  
23 group consists of the group number followed by a colon and a comma-delimited list of nodes  
24 in that group, with the groups delimited by semi-colons (e.g.,  
25 0:node000,node002,node004,node006;1:node001,node003,  
26 node005,node007)

27 **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (string)  
28 Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

29 **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (string)  
30 An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

31 **PMIX\_FABRIC\_INDEX** "pmix.fab.idx" (size\_t)  
32 The index of the fabric as returned in **pmix\_fabric\_t**.

33 **PMIX\_FABRIC\_NUM\_DEVICES** "pmix.fab.nverts" (size\_t)  
34 Total number of fabric devices in the overall system - corresponds to the number of rows or  
35 columns in the cost matrix.

36 **PMIX\_FABRIC\_COORDINATES** "pmix.fab.coords" (pmix\_data\_array\_t)

1 Array of `pmix_geometry_t` fabric coordinates for devices on the specified node. The  
2 array will contain the coordinates of all devices on the node, including values for all  
3 supported coordinate views. The information for devices on the local node shall be provided  
4 if the node is not specified in the request.

5 **PMIX\_FABRIC\_DIMS** "`pmix.fab.dims`" (`uint32_t`)

6 Number of dimensions in the specified fabric plane/view. If no plane is specified in a  
7 request, then the dimensions of all planes in the overall system will be returned as a  
8 `pmix_data_array_t` containing an array of `uint32_t` values. Default is to provide  
9 dimensions in *logical* view.

10 **PMIX\_FABRIC\_ENDPT** "`pmix.fab.endpt`" (`pmix_data_array_t`)

11 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given  
12 process (e.g., in the case where multiple devices are associated with a package to which the  
13 process is bound), the returned values will be provided in a `pmix_data_array_t` of  
14 `pmix_endpoint_t` elements.

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 **PMIX\_SWITCH\_PEERS** "`pmix.speers`" (`pmix_data_array_t`)

29 Peer ranks that share the same switch as the process specified in the call to `PMIx_Get`.  
30 Returns a `pmix_data_array_t` array of `pmix_info_t` results, each element  
31 containing the `PMIX_SWITCH_PEERS` key with a three-element `pmix_data_array_t`  
32 array of `pmix_info_t` containing the `PMIX_DEVICE_ID` of the local fabric device, the  
33 `PMIX_FABRIC_SWITCH` identifying the switch to which it is connected, and a  
34 comma-delimited string of peer ranks sharing the switch to which that device is connected.

35 **PMIX\_FABRIC\_PLANE** "`pmix.fab.plane`" (`string`)

36 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
37 for information, specifies the plane whose information is to be returned. When used directly  
38 as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
39 planes in the overall system.

40 **PMIX\_FABRIC\_SWITCH** "`pmix.fab.switch`" (`string`)

1 ID string of a fabric switch. When used as a modifier in a request for information, specifies  
2 the switch whose information is to be returned. When used directly as a key in a request,  
3 returns a `pmix_data_array_t` of string identifiers for all fabric switches in the overall  
4 system.

5 **`PMIX_FABRIC_DEVICE`** "`pmix.fabdev`" (`pmix_data_array_t`)

6 An array of `pmix_info_t` describing a particular fabric device using one or more of the  
7 attributes defined below. The first element in the array shall be the `PMIX_DEVICE_ID` of  
8 the device.

9 **`PMIX_FABRIC_DEVICE_INDEX`** "`pmix.fabdev.idx`" (`uint32_t`)

10 Index of the device within an associated communication cost matrix.

11 **`PMIX_FABRIC_DEVICE_NAME`** "`pmix.fabdev.nm`" (`string`)

12 The operating system name associated with the device. This may be a logical fabric interface  
13 name (e.g. "eth0" or "eno1") or an absolute filename.

14 **`PMIX_FABRIC_DEVICE_VENDOR`** "`pmix.fabdev.vndr`" (`string`)

15 Indicates the name of the vendor that distributes the device.

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_VENDORID`** "`pmix.fabdev.vendid`" (`string`)

19 This is a vendor-provided identifier for the device or product.

20 **`PMIX_FABRIC_DEVICE_DRIVER`** "`pmix.fabdev.driver`" (`string`)

21 The name of the driver associated with the device.

22 **`PMIX_FABRIC_DEVICE_FIRMWARE`** "`pmix.fabdev.fmwr`" (`string`)

23 The device's firmware version.

24 **`PMIX_FABRIC_DEVICE_ADDRESS`** "`pmix.fabdev.addr`" (`string`)

25 The primary link-level address associated with the device, such as a MAC address. If  
26 multiple addresses are available, only one will be reported.

27 **`PMIX_FABRIC_DEVICE_COORDINATES`** "`pmix.fab.coord`" (`pmix_geometry_t`)

28 The `pmix_geometry_t` fabric coordinates for the device, including values for all  
29 supported coordinate views.

30 **`PMIX_FABRIC_DEVICE_MTU`** "`pmix.fabdev.mtu`" (`size_t`)

31 The maximum transfer unit of link level frames or packets, in bytes.

32 **`PMIX_FABRIC_DEVICE_SPEED`** "`pmix.fabdev.speed`" (`size_t`)

33 The active link data rate, given in bits per second.

34 **`PMIX_FABRIC_DEVICE_STATE`** "`pmix.fabdev.state`" (`pmix_link_state_t`)

35 The last available physical port state for the specified device. Possible values are  
36 `PMIX_LINK_STATE_UNKNOWN`, `PMIX_LINK_DOWN`, and `PMIX_LINK_UP`, to indicate  
37 if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

1 **PMIX\_FABRIC\_DEVICE\_TYPE** "pmix.fabdev.type" (string)  
 2 Specifies the type of fabric interface currently active on the device, such as Ethernet or  
 3 InfiniBand.

4 **PMIX\_FABRIC\_DEVICE\_PCI\_DEVID** "pmix.fabdev.pcidevid" (string)  
 5 A node-level unique identifier for a PCI device. Provided only if the device is located on a  
 6 PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of  
 7 the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in  
 8 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The  
 9 combination of node identifier (**PMIX\_HOSTNAME** or **PMIX\_NODEID**) and  
 10 **PMIX\_FABRIC\_DEVICE\_PCI\_DEVID** shall be unique within the overall system.

## Device attributes

11 **PMIX\_DEVICE\_DISTANCES** "pmix.dev.dist" (pmix\_data\_array\_t)  
 12 Return an array of **pmix\_device\_distance\_t** containing the minimum and maximum  
 13 distances of the given process location to all devices of the specified type on the local node.

14 **PMIX\_DEVICE\_TYPE** "pmix.dev.type" (pmix\_device\_type\_t)  
 15 Bitmask specifying the type(s) of device(s) whose information is being requested. Only used  
 16 as a directive/qualifier.

17 **PMIX\_DEVICE\_ID** "pmix.dev.id" (string)  
 18 System-wide UUID or node-local OS name of a particular device.

## Sets-Groups attributes

19 **PMIX\_QUERY\_NUM\_PSETS** "pmix.qry.psetnum" (size\_t)  
 20 Return the number of process sets defined in the specified range (defaults to  
 21 **PMIX\_RANGE\_SESSION**).

22 **PMIX\_QUERY\_PSET\_NAMES** "pmix.qry.psets" (pmix\_data\_array\_t\*)  
 23 Return a **pmix\_data\_array\_t** containing an array of strings of the process set names  
 24 defined in the specified range (defaults to **PMIX\_RANGE\_SESSION**).

25 **PMIX\_QUERY\_PSET\_MEMBERSHIP** "pmix.qry.pmems" (pmix\_data\_array\_t\*)  
 26 Return an array of **pmix\_proc\_t** containing the members of the specified process set.

27 **PMIX\_PSET\_NAME** "pmix.pset.nm" (char\*)  
 28 The name of the newly defined process set.

29 **PMIX\_PSET\_MEMBERS** "pmix.pset.mems" (pmix\_data\_array\_t\*)  
 30 An array of **pmix\_proc\_t** containing the members of the newly defined process set.

31 **PMIX\_PSET\_NAMES** "pmix.pset.nms" (pmix\_data\_array\_t\*)  
 32 Returns an array of **char\*** string names of the process sets in which the given process is a  
 33 member.

34 **PMIX\_QUERY\_NUM\_GROUPS** "pmix.qry.pgrpnum" (size\_t)  
 35 Return the number of process groups defined in the specified range (defaults to session).  
 36 OPTIONAL QUALIFIERS: **PMIX\_RANGE**.

1 **PMIX\_QUERY\_GROUP\_NAMES** "pmix.qry.pgrp" (pmix\_data\_array\_t\*)  
2     Return a **pmix\_data\_array\_t** containing an array of string names of the process groups  
3     defined in the specified range (defaults to session). OPTIONAL QUALIFIERS:  
4     **PMIX\_RANGE**.

5 **PMIX\_QUERY\_GROUP\_MEMBERSHIP**  
6 "pmix.qry.pgrpmems" (pmix\_data\_array\_t\*)  
7     Return a **pmix\_data\_array\_t** of **pmix\_proc\_t** containing the members of the  
8     specified process group. REQUIRED QUALIFIERS: **PMIX\_GROUP\_ID**.

9 **PMIX\_GROUP\_ID** "pmix.grp.id" (char\*)  
10     User-provided group identifier - as the group identifier may be used in PMIx operations, the  
11     user is required to ensure that the provided ID is unique within the scope of the host  
12     environment (e.g., by including some user-specific or application-specific prefix or suffix to  
13     the string).

14 **PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (bool)  
15     This process is the leader of the group.

16 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)  
17     Participation is optional - do not return an error if any of the specified processes terminate  
18     without having joined. The default is **false**.

19 **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)  
20     Notify remaining members when another member terminates without first leaving the group.  
21

22 **PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (bool)  
23     Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective  
24     operation.

25 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)  
26     Requests that the RM assign a new context identifier to the newly created group. The  
27     identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
28     specified in the request. Thus, the value serves as a means of identifying the group within  
29     that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

30 **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)  
31     Group operation only involves local processes. PMIx implementations are *required* to  
32     automatically scan an array of group members for local vs remote processes - if only local  
33     processes are detected, the implementation need not execute a global collective for the  
34     operation unless a context ID has been requested from the host environment. This can result  
35     in significant time savings. This attribute can be used to optimize the operation by indicating  
36     whether or not only local processes are represented, thus allowing the implementation to  
37     bypass the scan.

38 **PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (size\_t)  
39     Context identifier assigned to the group by the host RM.

1 **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)  
2 Data collected during group construction to ensure communication between group members  
3 is supported upon completion of the operation.  
4 **PMIX\_GROUP\_NAMES** "pmix.pgrp.nm" (pmix\_data\_array\_t\*)  
5 Returns an array of **char\*** string names of the process groups in which the given process is  
6 a member.

## Process Mgmt attributes

7 **PMIX\_OUTPUT\_TO\_DIRECTORY** "pmix.outdir" (char\*)  
8 Direct output into files of form "<directory>/<jobid>/rank.<rank>/  
9 **stdout[err]**" - can be assigned to the entire job (by including attribute in the *job\_info*  
10 array) or on a per-application basis in the *info* array for each **pmix\_app\_t**.  
11  
12 **PMIX\_TIMEOUT\_STACKTRACES** "pmix.tim.stack" (bool)  
13 Include process stacktraces in timeout report from a job.  
14 **PMIX\_TIMEOUT\_REPORT\_STATE** "pmix.tim.state" (bool)  
15 Report process states in timeout report from a job.  
16 **PMIX\_NOTIFY\_JOB\_EVENTS** "pmix.note.jev" (bool)  
17 Requests that the launcher generate the **PMIX\_EVENT\_JOB\_START**,  
18 **PMIX\_LAUNCH\_COMPLETE**, and **PMIX\_EVENT\_JOB\_END** events. Each event is to  
19 include at least the namespace of the corresponding job and a **PMIX\_EVENT\_TIMESTAMP**  
20 indicating the time the event occurred. Note that the requester must register for these  
21 individual events, or capture and process them by registering a default event handler instead  
22 of individual handlers and then process the events based on the returned status code. Another  
23 common method is to register one event handler for all job-related events, with a separate  
24 handler for non-job events - see **PMIx\_Register\_event\_handler** for details.  
25 **PMIX\_NOTIFY\_PROC\_TERMINATION** "pmix.noteproc" (bool)  
26 Requests that the launcher generate the **PMIX\_EVENT\_PROC\_TERMINATED** event  
27 whenever a process either normally or abnormally terminates.  
28 **PMIX\_NOTIFY\_PROC\_ABNORMAL\_TERMINATION** "pmix.noteabproc" (bool)  
29 Requests that the launcher generate the **PMIX\_EVENT\_PROC\_TERMINATED** event only  
30 when a process abnormally terminates.  
31 **PMIX\_LOG\_PROC\_TERMINATION** "pmix.logproc" (bool)  
32 Requests that the launcher log the **PMIX\_EVENT\_PROC\_TERMINATED** event whenever a  
33 process either normally or abnormally terminates.  
34 **PMIX\_LOG\_PROC\_ABNORMAL\_TERMINATION** "pmix.logabproc" (bool)  
35 Requests that the launcher log the **PMIX\_EVENT\_PROC\_TERMINATED** event only when a  
36 process abnormally terminates.  
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 13.4.3.

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 13.4.3. 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_FIRST_ENVAR` "pmix.envar.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 Event attributes

15 `PMIX_EVENT_TIMESTAMP` "pmix.evtstamp" (time\_t)

16 System time when the associated event occurred.

### 17 C.8.3 Added Environmental Variables

#### 18 Tool environmental variables

19 `PMIX_LAUNCHER_RNDZ_URI`

20 `PMIX_LAUNCHER_RNDZ_FILE`

21 `PMIX_KEEPALIVE_PIPE`

### 23 C.8.4 Added Macros

24 `PMIX_CHECK_RESERVED_KEY` `PMIX_INFO_WAS_PROCESSED` `PMIX_INFO_PROCESSED`

25 `PMIX_INFO_LIST_START` `PMIX_INFO_LIST_ADD` `PMIX_INFO_LIST_XFER`

26 `PMIX_INFO_LIST_CONVERT` `PMIX_INFO_LIST_RELEASE`

### 27 C.8.5 Deprecated APIs

28 `pmix_evhdlr_reg_cbfunc_t` Renamed to `pmix_hdlr_reg_cbfunc_t`

29 The `pmix_server_client_connected_fn_t` server module entry point has been  
30 *deprecated* in favor of `pmix_server_client_connected2_fn_t`

31 `PMIx_tool_connect_to_server` Replaced by `PMIx_tool_attach_to_server` to  
32 allow return of the process identifier of the server to which the tool has attached.



## 1 C.8.6 Deprecated constants

2 The following constants were deprecated in v4.0:

3	<b>PMIX_ERR_DEBUGGER_RELEASE</b>	Renamed to <b>PMIX_DEBUGGER_RELEASE</b>
4	<b>PMIX_ERR_JOB_TERMINATED</b>	Renamed to <b>PMIX_EVENT_JOB_END</b>
5	<b>PMIX_EXISTS</b>	Renamed to <b>PMIX_ERR_EXISTS</b>
6	<b>PMIX_ERR_PROC_ABORTED</b>	Consolidated with <b>PMIX_EVENT_PROC_TERMINATED</b>
7	<b>PMIX_ERR_PROC_ABORTING</b>	Consolidated with <b>PMIX_EVENT_PROC_TERMINATED</b>
8	<b>PMIX_ERR_LOST_CONNECTION_TO_SERVER</b>	Consolidated into
9	<b>PMIX_ERR_LOST_CONNECTION</b>	
10	<b>PMIX_ERR_LOST_PEER_CONNECTION</b>	Consolidated into
11	<b>PMIX_ERR_LOST_CONNECTION</b>	
12	<b>PMIX_ERR_LOST_CONNECTION_TO_CLIENT</b>	Consolidated into
13	<b>PMIX_ERR_LOST_CONNECTION</b>	
14	<b>PMIX_ERR_INVALID_TERMINATION</b>	Renamed to <b>PMIX_ERR_JOB_TERM_WO_SYNC</b>
15	<b>PMIX_PROC_TERMINATED</b>	Renamed to <b>PMIX_EVENT_PROC_TERMINATED</b>
16	<b>PMIX_ERR_NODE_DOWN</b>	Renamed to <b>PMIX_EVENT_NODE_DOWN</b>
17	<b>PMIX_ERR_NODE_OFFLINE</b>	Renamed to <b>PMIX_EVENT_NODE_OFFLINE</b>
18	<b>PMIX_ERR_SYS_OTHER</b>	Renamed to <b>PMIX_EVENT_SYS_OTHER</b>
19	<b>PMIX_CONNECT_REQUESTED</b>	Connection has been requested by a PMIx-based tool -
20		deprecated as not required.
21	<b>PMIX_PROC_HAS_CONNECTED</b>	A tool or client has connected to the PMIx server -
22		deprecated in favor of the new <b>pmix_server_client_connected2_fn_t</b> server
23		module API

## 24 C.8.7 Removed constants

25 The following constants were removed from the PMIx Standard in v4.0 as they are internal to a  
26 particular PMIx implementation.

27	<b>PMIX_ERR_HANDSHAKE_FAILED</b>	Connection handshake failed
28	<b>PMIX_ERR_READY_FOR_HANDSHAKE</b>	Ready for handshake
29	<b>PMIX_ERR_IN_ERRNO</b>	Error defined in <b>errno</b>
30	<b>PMIX_ERR_INVALID_VAL_LENGTH</b>	Invalid value length
31	<b>PMIX_ERR_INVALID_LENGTH</b>	Invalid argument length
32	<b>PMIX_ERR_INVALID_NUM_ARGS</b>	Invalid number of arguments
33	<b>PMIX_ERR_INVALID_ARGS</b>	Invalid arguments
34	<b>PMIX_ERR_INVALID_NUM_PARSED</b>	Invalid number parsed
35	<b>PMIX_ERR_INVALID_KEYVALP</b>	Invalid key/value pair
36	<b>PMIX_ERR_INVALID_SIZE</b>	Invalid size
37	<b>PMIX_ERR_PROC_REQUESTED_ABORT</b>	Process is already requested to abort
38	<b>PMIX_ERR_SERVER_FAILED_REQUEST</b>	Failed to connect to the server
39	<b>PMIX_ERR_PROC_ENTRY_NOT_FOUND</b>	Process not found

1	<b>PMIX_ERR_INVALID_ARG</b>	Invalid argument
2	<b>PMIX_ERR_INVALID_KEY</b>	Invalid key
3	<b>PMIX_ERR_INVALID_KEY_LENGTH</b>	Invalid key length
4	<b>PMIX_ERR_INVALID_VAL</b>	Invalid value
5	<b>PMIX_ERR_INVALID_NAMESPACE</b>	Invalid namespace
6	<b>PMIX_ERR_SERVER_NOT_AVAIL</b>	Server is not available
7	<b>PMIX_ERR_SILENT</b>	Silent error
8	<b>PMIX_ERR_PACK_MISMATCH</b>	Pack mismatch
9	<b>PMIX_ERR_DATA_VALUE_NOT_FOUND</b>	Data value not found
10	<b>PMIX_ERR_NOT_IMPLEMENTED</b>	Not implemented
11	<b>PMIX_GDS_ACTION_COMPLETE</b>	The GDS action has completed
12	<b>PMIX_NOTIFY_ALLOC_COMPLETE</b>	Notify that a requested allocation operation is complete
13		- the result of the request will be included in the <i>info</i> array

## 14 C.8.8 Deprecated attributes

15 The following attributes were deprecated in v4.0:

16	<b>PMIX_TOPOLOGY</b>	"pmix.topo" ( <code>hwloc_topology_t</code> )
17		Renamed to <b>PMIX_TOPOLOGY2</b> .
18	<b>PMIX_DEBUG_JOB</b>	"pmix.dbg.job" ( <code>char*</code> )
19		Renamed to <b>PMIX_DEBUG_TARGET</b>
20	<b>PMIX_RECONNECT_SERVER</b>	"pmix.tool.recon" ( <code>bool</code> )
21		Renamed to the <b>PMIX_tool_connect_to_server</b> API
22	<b>PMIX_ALLOC_NETWORK</b>	"pmix.alloc.net" ( <code>array</code> )
23		Renamed to <b>PMIX_ALLOC_FABRIC</b>
24	<b>PMIX_ALLOC_NETWORK_ID</b>	"pmix.alloc.netid" ( <code>char*</code> )
25		Renamed to <b>PMIX_ALLOC_FABRIC_ID</b>
26	<b>PMIX_ALLOC_NETWORK_QOS</b>	"pmix.alloc.netqos" ( <code>char*</code> )
27		Renamed to <b>PMIX_ALLOC_FABRIC_QOS</b>
28	<b>PMIX_ALLOC_NETWORK_TYPE</b>	"pmix.alloc.nettype" ( <code>char*</code> )
29		Renamed to <b>PMIX_ALLOC_FABRIC_TYPE</b>
30	<b>PMIX_ALLOC_NETWORK_PLANE</b>	"pmix.alloc.netplane" ( <code>char*</code> )
31		Renamed to <b>PMIX_ALLOC_FABRIC_PLANE</b>
32	<b>PMIX_ALLOC_NETWORK_ENDPTS</b>	"pmix.alloc.endpts" ( <code>size_t</code> )
33		Renamed to <b>PMIX_ALLOC_FABRIC_ENDPTS</b>
34	<b>PMIX_ALLOC_NETWORK_ENDPTS_NODE</b>	"pmix.alloc.endpts.nd" ( <code>size_t</code> )
35		Renamed to <b>PMIX_ALLOC_FABRIC_ENDPTS_NODE</b>
36	<b>PMIX_ALLOC_NETWORK_SEC_KEY</b>	"pmix.alloc.nsec" ( <code>pmix_byte_object_t</code> )
37		Renamed to <b>PMIX_ALLOC_FABRIC_SEC_KEY</b>
38	<b>PMIX_PROC_DATA</b>	"pmix.pdata" ( <code>pmix_data_array_t</code> )
39		Renamed to <b>PMIX_PROC_INFO_ARRAY</b>
40	<b>PMIX_LOCALITY</b>	"pmix.loc" ( <code>pmix_locality_t</code> )

1 Relative locality of the specified process to the requester, expressed as a bitmask as per the  
2 description in the [pmix\\_locality\\_t](#) section. This value is unique to the requesting  
3 process and thus cannot be communicated by the server as part of the job-level information.  
4 Its use has been replaced by the [PMIx\\_Get\\_relative\\_locality](#) function.

## 5 C.8.9 Removed attributes

6 The following attributes were removed from the PMIx Standard in v4.0 as they are internal to a  
7 particular PMIx implementation. Users are referred to the [PMIx\\_Load\\_topology](#) API for  
8 obtaining the local topology description.

9 **PMIX\_LOCAL\_TOPO** "pmix.ltopo" (char\*)

10 XML representation of local node topology.

11 **PMIX\_TOPOLOGY\_XML** "pmix.topo.xml" (char\*)

12 XML-based description of topology

13 **PMIX\_TOPOLOGY\_FILE** "pmix.topo.file" (char\*)

14 Full path to file containing XML topology description

15 **PMIX\_TOPOLOGY\_SIGNATURE** "pmix.toposig" (char\*)

16 Topology signature string.

17 **PMIX\_HWLOC\_SHMEM\_ADDR** "pmix.hwlocaddr" (size\_t)

18 Address of the HWLOC shared memory segment.

19 **PMIX\_HWLOC\_SHMEM\_SIZE** "pmix.hwlocsize" (size\_t)

20 Size of the HWLOC shared memory segment.

21 **PMIX\_HWLOC\_SHMEM\_FILE** "pmix.hwlocfile" (char\*)

22 Path to the HWLOC shared memory file.

23 **PMIX\_HWLOC\_XML\_V1** "pmix.hwlocxml1" (char\*)

24 XML representation of local topology using HWLOC's v1.x format.

25 **PMIX\_HWLOC\_XML\_V2** "pmix.hwlocxml2" (char\*)

26 XML representation of local topology using HWLOC's v2.x format.

27 **PMIX\_HWLOC\_SHARE\_TOPO** "pmix.hwlocsh" (bool)

28 Share the HWLOC topology via shared memory

29 **PMIX\_HWLOC\_HOLE\_KIND** "pmix.hwlocholek" (char\*)

30 Kind of VM "hole" HWLOC should use for shared memory

31 **PMIX\_DSTPATH** "pmix.dstpath" (char\*)

32 Path to shared memory data storage (dstore) files. Deprecated from Standard as being  
33 implementation specific.

34 **PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (char\*)

35 Comma-delimited list of algorithms to use for the collective operation. PMIx does not  
36 impose any requirements on a host environment's collective algorithms. Thus, the  
37 acceptable values for this attribute will be environment-dependent - users are encouraged to  
38 check their host environment for supported values.

39 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)

40 If **true**, indicates that the requested choice of algorithm is mandatory.

41 **PMIX\_PROC\_BLOB** "pmix.pblob" (pmix\_byte\_object\_t)

1 Packed blob of process data.

2 **PMIX\_MAP\_BLOB** "pmix.mblob" (pmix\_byte\_object\_t)

3 Packed blob of process location.

4 **PMIX\_MAPPER** "pmix.mapper" (char\*)

5 Mapping mechanism to use for placing spawned processes - when accessed using

6 **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping  
7 mechanism used for the provided namespace.

8 **PMIX\_NON\_PMI** "pmix.nonpmi" (bool)

9 Spawned processes will not call **PMIx\_Init**.

10 **PMIX\_PROC\_URI** "pmix.puri" (char\*)

11 URI containing contact information for the specified process.

12 **PMIX\_ARCH** "pmix.arch" (uint32\_t)

13 Architecture flag.

## 14 C.9 Version 4.1: TBD

15 The v4.1 update includes clarifications and corrections from the v4.0 document:

- 16 • Remove some stale language in [Chapter 9.1](#).
- 17 • Provisional Items:
  - 18 – Storage Chapter [19](#) on page [455](#)

### 19 C.9.1 Added Functions (Provisional)

- 20 • [PMIx\\_Data\\_load](#)
- 21 • [PMIx\\_Data\\_unload](#)
- 22 • [PMIx\\_Data\\_compress](#)
- 23 • [PMIx\\_Data\\_decompress](#)

### 24 C.9.2 Added Data Structures (Provisional)

- 25 • [pmix\\_storage\\_medium\\_t](#)
- 26 • [pmix\\_storage\\_accessibility\\_t](#)
- 27 • [pmix\\_storage\\_persistence\\_t](#)
- 28 • [pmix\\_storage\\_access\\_type\\_t](#)

### 29 C.9.3 Added Macros (Provisional)

- 30 • [PMIX\\_NAMESPACE\\_INVALID](#)
- 31 • [PMIX\\_RANK\\_IS\\_VALID](#)
- 32 • [PMIX\\_PROCID\\_INVALID](#)
- 33 • [PMIX\\_PROCID\\_XFER](#)

## 1 C.9.4 Added Constants (Provisional)

- 2 • `PMIX_PROC_NAMESPACE`

### 3 Storage constants

- 4 • `PMIX_STORAGE_MEDIUM_UNKNOWN`
- 5 • `PMIX_STORAGE_MEDIUM_TAPE`
- 6 • `PMIX_STORAGE_MEDIUM_HDD`
- 7 • `PMIX_STORAGE_MEDIUM_SSD`
- 8 • `PMIX_STORAGE_MEDIUM_NVME`
- 9 • `PMIX_STORAGE_MEDIUM_PMEM`
- 10 • `PMIX_STORAGE_MEDIUM_RAM`
- 11 • `PMIX_STORAGE_ACCESSIBILITY_NODE`
- 12 • `PMIX_STORAGE_ACCESSIBILITY_SESSION`
- 13 • `PMIX_STORAGE_ACCESSIBILITY_JOB`
- 14 • `PMIX_STORAGE_ACCESSIBILITY_RACK`
- 15 • `PMIX_STORAGE_ACCESSIBILITY_CLUSTER`
- 16 • `PMIX_STORAGE_ACCESSIBILITY_REMOTE`
- 17 • `PMIX_STORAGE_PERSISTENCE_TEMPORARY`
- 18 • `PMIX_STORAGE_PERSISTENCE_NODE`
- 19 • `PMIX_STORAGE_PERSISTENCE_SESSION`
- 20 • `PMIX_STORAGE_PERSISTENCE_JOB`
- 21 • `PMIX_STORAGE_PERSISTENCE_SCRATCH`
- 22 • `PMIX_STORAGE_PERSISTENCE_PROJECT`
- 23 • `PMIX_STORAGE_PERSISTENCE_ARCHIVE`
- 24 • `PMIX_STORAGE_ACCESS_RD`
- 25 • `PMIX_STORAGE_ACCESS_WR`
- 26 • `PMIX_STORAGE_ACCESS_RDWR`

## 27 C.9.5 Added Attributes (Provisional)

### 28 Storage attributes

- 29 `PMIX_STORAGE_ID` "`pmix.strg.id`" (`char*`)  
30 An identifier for the storage system (e.g., `lustre-fs1`, `daos-oss1`, `home-fs`)
- 31 `PMIX_STORAGE_PATH` "`pmix.strg.path`" (`char*`)  
32 Mount point path for the storage system (valid only for file-based storage systems)
- 33 `PMIX_STORAGE_TYPE` "`pmix.strg.type`" (`char*`)  
34 Type of storage system (i.e., `lustre`, `gpfs`, `daos`, `ext4`)
- 35 `PMIX_STORAGE_VERSION` "`pmix.strg.ver`" (`char*`)  
36 Version string for the storage system
- 37 `PMIX_STORAGE_MEDIUM` "`pmix.strg.medium`" (`pmix_storage_medium_t`)

Types of storage mediums utilized by the storage system (e.g., SSDs, HDDs, tape)

**PMIX\_STORAGE\_ACCESSIBILITY**

**"pmix.strg.access"** (**pmix\_storage\_accessibility\_t**)

Accessibility level of the storage system (e.g., within same node, within same session)

**PMIX\_STORAGE\_PERSISTENCE**

**"pmix.strg.persist"** (**pmix\_storage\_persistence\_t**)

Persistence level of the storage system (e.g., sratch storage or achive storage)

**PMIX\_QUERY\_STORAGE\_LIST** **"pmix.strg.list"** (**char\***)

Comma-delimited list of storage identifiers (i.e., **PMIX\_STORAGE\_ID** types) for available storage systems

**PMIX\_STORAGE\_CAPACITY\_LIMIT** **"pmix.strg.caplim"** (**double**)

Overall limit on capacity (in bytes) for the storage system

**PMIX\_STORAGE\_CAPACITY\_USED** **"pmix.strg.capuse"** (**double**)

Overall used capacity (in bytes) for the storage system

**PMIX\_STORAGE\_OBJECT\_LIMIT** **"pmix.strg.objlim"** (**uint64\_t**)

Overall limit on number of objects (e.g., inodes) for the storage system

**PMIX\_STORAGE\_OBJECTS\_USED** **"pmix.strg.objuse"** (**uint64\_t**)

Overall used number of objects (e.g., inodes) for the storage system

**PMIX\_STORAGE\_MINIMAL\_XFER\_SIZE** **"pmix.strg.minxfer"** (**double**)

Minimal transfer size (in bytes) for the storage system - this is the storage system's atomic unit of transfer (e.g., block size)

**PMIX\_STORAGE\_SUGGESTED\_XFER\_SIZE** **"pmix.strg.sxfer"** (**double**)

Suggested transfer size (in bytes) for the storage system

**PMIX\_STORAGE\_BW\_MAX** **"pmix.strg.bwmax"** (**double**)

Maximum bandwidth (in bytes/sec) for storage system - provided as the theoretical maximum or the maximum observed bandwidth value

**PMIX\_STORAGE\_BW\_CUR** **"pmix.strg.bwcur"** (**double**)

Observed bandwidth (in bytes/sec) for storage system - provided as a recently observed bandwidth value, with the exact measurement interval depending on the storage system and/or PMIx library implementation

**PMIX\_STORAGE\_IOPS\_MAX** **"pmix.strg.iopsmax"** (**double**)

Maximum IOPS (in I/O operations per second) for storage system - provided as the theoretical maximum or the maximum observed IOPS value

**PMIX\_STORAGE\_IOPS\_CUR** **"pmix.strg.iopscur"** (**double**)

Observed IOPS (in I/O operations per second) for storage system - provided as a recently observed IOPS value, with the exact measurement interval depending on the storage system and/or PMIx library implementation

1  
2  
3  
4  
5

**PMIX\_STORAGE\_ACCESS\_TYPE**

"**pmix.strg.atype**" (**pmix\_storage\_access\_type\_t**)

Qualifier describing the type of storage access to return information for (e.g., for qualifying

**PMIX\_STORAGE\_BW\_CUR**, **PMIX\_STORAGE\_IOPS\_CUR**, or

**PMIX\_STORAGE\_SUGGESTED\_XFER\_SIZE** attributes)

Unofficial Draft

## APPENDIX D

# Acknowledgements

---

1 This document represents the work of many people who have contributed to the PMIx community.  
2 Without the hard work and dedication of these people this document would not have been possible.  
3 The sections below list some of the active participants and organizations in the various PMIx  
4 standard iterations.

## 5 D.1 Version 4.0

6 The following list includes some of the active participants in the PMIx v4 standardization process.

- 7 • Ralph H. Castain and Danielle Sikich
- 8 • Joshua Hursey and David Solt
- 9 • Dirk Schubert
- 10 • John DelSignore
- 11 • Aurelien Bouteiller
- 12 • Michael A Raymond
- 13 • Howard Pritchard and Nathan Hjelm
- 14 • Brice Goglin
- 15 • Kathryn Mohror and Stephen Herbein
- 16 • Thomas Naughton and Swaroop Pophale
- 17 • William E. Allcock and Paul Rich
- 18 • Michael Karo
- 19 • Artem Polyakov

20 The following institutions supported this effort through time and travel support for the people listed  
21 above.

- 22 • Intel Corporation
- 23 • IBM, Inc.
- 24 • Allinea (ARM)



- 1           • Perforce
- 2           • University of Tennessee, Knoxville
- 3           • The Exascale Computing Project, an initiative of the US Department of Energy
- 4           • National Science Foundation
- 5           • HPE Co.
- 6           • Los Alamos National Laboratory
- 7           • INRIA
- 8           • Lawrence Livermore National Laboratory
- 9           • Oak Ridge National Laboratory
- 10          • Argonne National Laboratory
- 11          • Altair
- 12          • NVIDIA

## 13 **D.2 Version 3.0**

14           The following list includes some of the active participants in the PMIx v3 standardization process.

- 15          • Ralph H. Castain, Andrew Friedley, Brandon Yates
- 16          • Joshua Hursey and David Solt
- 17          • Aurelien Bouteiller and George Bosilca
- 18          • Dirk Schubert
- 19          • Kevin Harms
- 20          • Artem Polyakov

21           The following institutions supported this effort through time and travel support for the people listed  
22           above.

- 23          • Intel Corporation
- 24          • IBM, Inc.
- 25          • University of Tennessee, Knoxville
- 26          • The Exascale Computing Project, an initiative of the US Department of Energy
- 27          • National Science Foundation
- 28          • Argonne National Laboratory

- 1 • Allinea (ARM)
- 2 • NVIDIA

### 3 **D.3 Version 2.0**

4 The following list includes some of the active participants in the PMIx v2 standardization process.

- 5 • Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm  
6 and Terry Wilmarth
- 7 • Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
- 8 • Aurelien Bouteiller and George Bosilca
- 9 • Artem Polyakov, Igor Ivanov and Boris Karasev
- 10 • Gilles Gouillardet
- 11 • Michael A Raymond and Jim Stoffel
- 12 • Dirk Schubert
- 13 • Moe Jette
- 14 • Takahiro Kawashima and Shinji Sumimoto
- 15 • Howard Pritchard
- 16 • David Beer
- 17 • Brice Goglin
- 18 • Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
- 19 • Adam Moody and Martin Schulz
- 20 • Ryan Grant and Stephen Olivier
- 21 • Michael Karo

22 The following institutions supported this effort through time and travel support for the people listed  
23 above.

- 24 • Intel Corporation
- 25 • IBM, Inc.
- 26 • University of Tennessee, Knoxville
- 27 • The Exascale Computing Project, an initiative of the US Department of Energy
- 28 • National Science Foundation
- 29 • Mellanox, Inc.

- 1 • Research Organization for Information Science and Technology
- 2 • HPE Co.
- 3 • Allinea (ARM)
- 4 • SchedMD, Inc.
- 5 • Fujitsu Limited
- 6 • Los Alamos National Laboratory
- 7 • Adaptive Solutions, Inc.
- 8 • INRIA
- 9 • Oak Ridge National Laboratory
- 10 • Lawrence Livermore National Laboratory
- 11 • Sandia National Laboratory
- 12 • Altair

## 13 **D.4 Version 1.0**

14 The following list includes some of the active participants in the PMIx v1 standardization process.

- 15 • Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- 16 • Joshua Hursey and David Solt
- 17 • Aurelien Bouteiller and George Bosilca
- 18 • Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- 19 • Gilles Gouaillardet
- 20 • Gary Brown
- 21 • Moe Jette

22 The following institutions supported this effort through time and travel support for the people listed  
23 above.

- 24 • Intel Corporation
- 25 • IBM, Inc.
- 26 • University of Tennessee, Knoxville
- 27 • Mellanox, Inc.
- 28 • Research Organization for Information Science and Technology

- 1           • Adaptive Solutions, Inc.
- 2           • SchedMD, Inc.

Unofficial Draft

# 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.
- [3] Khaled Hamidouche, Jian Lin, Mingzhe Li, Jie Zhang, and D.K.Panda. Supporting hybrid MPI+PGAS programming models through unified communication runtime: An MVAPICH2-X approach. In *4th MVAPICH User's Group*, MUG, 2016.
- [4] Message Passing Interface Forum. *MPI: A Message-passing Interface Standard, Version 3.1*. High-Performance Computing Center Stuttgart, University of Stuttgart, June 2015.
- [5] Geoffroy R. Vallée. MOC - MPI Open MP Coordination library. <https://github.com/OMPI-X/MOC>, 2018. [Online; accessed 20-Dec-2019].
- [6] Geoffroy R. Vallée and David Bernhold. Improving support of MPI+OpenMP applications. In *Proceedings of the 25th European MPI Users' Group Meeting*, EuroMPI, 2018.

# Index

---

General terms and other items not induced in the other indices.

application, [8](#), [89](#), [116](#), [307](#), [309](#), [313](#), [564](#), [567](#)

attribute, [9](#)

client, [5](#), [8](#), [60](#)

clients, [8](#)

clone, [8](#)

clones, [8](#), [69](#), [72](#), [183](#), [186](#), [187](#), [189](#), [212](#), [215](#), [575](#)

data realm, [80](#), [272](#), [273](#)

data realms, [80](#)

device, [9](#)

devices, [9](#)

Direct Modex, [260](#), [323](#)

fabric, [9](#)

fabric device, [9](#)

fabric devices, [9](#)

fabric plane, [9](#), [168](#), [173](#), [204](#), [207](#), [208](#), [277](#), [326](#)

fabric planes, [9](#)

fabrics, [9](#)

host environment, [8](#)

instant on, [9](#), [94](#), [259](#)

job, [7](#), [8](#), [82–86](#), [89](#), [116](#), [297–301](#), [303](#), [307–309](#), [311](#), [313](#), [324](#), [326](#), [327](#), [514](#), [532](#), [564](#), [565](#), [567](#), [578–580](#)

key, [9](#)

namespace, [7](#)

node, [8](#), [89](#), [116](#), [168](#), [173](#), [204](#), [207](#), [208](#), [307](#), [326](#)

package, [8](#), [87](#), [305](#), [579](#)

peer, [8](#), [88](#), [303](#)

peers, [8](#)

process, [8](#), [89](#), [116](#), [168](#), [173](#), [204](#), [207](#), [208](#), [307](#), [326](#)

processing unit, [8](#)

rank, [8](#), [313](#)

realm, [80](#)

realms, [80](#)

resource manager, [8](#)

RM, [8](#)

scheduler, [8](#), [275](#)

server, [5](#)

session, [7](#), [82](#), [89](#), [90](#), [116](#), [297](#), [298](#), [307](#), [308](#), [564](#), [565](#), [567](#), [578](#)

singleton, [5](#)

thread, [8](#)

threads, [8](#)

tool, [5](#)

workflow, [8](#)

workflows, [8](#), [395](#)

Unofficial Draft

# Index of APIs

---

PMIx\_Abort, [26](#), [162](#), [163](#), [348](#), [350](#), [469](#), [484](#), [562](#)  
    PMIxClient.abort (Python), [483](#)

PMIx\_Alloc\_directive\_string, [57](#), [504](#), [563](#)  
    PMIxClient.alloc\_directive\_string (Python), [504](#)

PMIx\_Allocation\_request, [118](#), [201](#), [201](#), [207](#), [492](#), [565](#), [567](#), [570](#)  
    PMIxClient.allocation\_request (Python), [491](#)

PMIx\_Allocation\_request\_nb, [204](#), [207](#), [209](#), [563](#)

PMIx\_Commit, [69](#), [71](#), [72](#), [95](#), [97](#), [97](#), [98](#), [323](#), [324](#), [351](#), [355](#), [485](#), [530](#), [531](#), [562](#)  
    PMIxClient.commit (Python), [485](#)

PMIx\_Compute\_distances, [195](#), [196](#), [501](#), [571](#)  
    PMIxClient.compute\_distances (Python), [501](#)

PMIx\_Compute\_distances\_nb, [196](#), [571](#)

PMIx\_Connect, [182](#), [184](#), [186](#), [187](#), [189](#), [233–235](#), [438](#), [439](#), [489](#), [562](#), [564](#)  
    PMIxClient.connect (Python), [488](#)

PMIx\_Connect\_nb, [184](#), [184](#), [562](#)

pmix\_connection\_cbfunc\_t, [376](#), [376](#)

pmix\_credential\_cbfunc\_t, [283](#), [393](#), [394](#)

PMIx\_Data\_compress, [159](#), [161](#), [592](#)

PMIx\_Data\_copy, [156](#), [563](#)

PMIx\_Data\_copy\_payload, [157](#), [563](#)

PMIx\_Data\_decompress, [160](#), [592](#)

PMIx\_Data\_load, [158](#), [592](#)

PMIx\_Data\_pack, [152](#), [153](#), [295](#), [563](#)

PMIx\_Data\_print, [156](#), [563](#)

PMIx\_Data\_range\_string, [57](#), [503](#), [563](#)  
    PMIxClient.data\_range\_string (Python), [503](#)

PMIx\_Data\_type\_string, [57](#), [504](#), [563](#)  
    PMIxClient.data\_type\_string (Python), [504](#)

PMIx\_Data\_unload, [159](#), [592](#)

PMIx\_Data\_unpack, [154](#), [159](#), [563](#)

PMIx\_Deregister\_event\_handler, [144](#), [497](#), [549](#), [563](#), [570](#)  
    PMIxClient.deregister\_event\_handler (Python), [497](#)

pmix\_device\_dist\_cbfunc\_t, [197](#), [197](#), [572](#)

PMIx\_Device\_type\_string, [59](#), [507](#), [571](#)  
    PMIxClient.device\_type\_string (Python), [507](#)

PMIx\_Disconnect, [186](#), [187–189](#), [235](#), [438](#), [439](#), [489](#), [562](#), [564](#)  
    PMIxClient.disconnect (Python), [489](#)

PMIx\_Disconnect\_nb, [188](#), [189](#), [235](#), [562](#)



pmix\_dmodex\_response\_fn\_t, [323](#), [324](#)  
PMIx\_Error\_string, [56](#), [502](#), [562](#)  
    PMIxClient.error\_string (Python), [501](#)  
pmix\_event\_notification\_cbfunc\_fn\_t, [143](#), [149](#), [149](#), [554](#), [557](#)  
PMIx\_Fabric\_deregister, [279](#), [280](#), [499](#), [571](#)  
    PMIxClient.fabric\_deregister (Python), [499](#)  
PMIx\_Fabric\_deregister\_nb, [280](#), [571](#)  
PMIx\_Fabric\_register, [268](#), [276](#), [278](#), [498](#), [571](#)  
    PMIxClient.fabric\_register (Python), [498](#)  
PMIx\_Fabric\_register\_nb, [277](#), [571](#)  
PMIx\_Fabric\_update, [277](#), [278](#), [279](#), [499](#), [571](#)  
    PMIxClient.fabric\_update (Python), [498](#)  
PMIx\_Fabric\_update\_nb, [279](#), [571](#)  
PMIx\_Fence, [4](#), [68](#), [69](#), [70](#), [72](#), [94](#), [184](#), [187](#), [232](#), [241](#), [245](#), [260](#), [293](#), [323](#), [350](#), [352](#), [470](#), [486](#), [530](#),  
    [531](#), [562](#), [575](#)  
    PMIxClient.fence (Python), [485](#)  
PMIx\_Fence\_nb, [54](#), [70](#), [350](#), [352](#), [470](#), [562](#), [570](#)  
PMIx\_Finalize, [26](#), [64](#), [66](#), [66](#), [181](#), [347](#), [348](#), [438](#), [439](#), [469](#), [483](#), [562](#)  
    PMIxClient.finalize (Python), [483](#)  
PMIx\_generate\_ppn, [296](#), [509](#), [562](#), [568](#)  
    PMIxServer.generate\_ppn (Python), [509](#)  
PMIx\_generate\_regex, [294](#), [296](#), [308](#), [509](#), [562](#), [568](#)  
    PMIxServer.generate\_regex (Python), [508](#)  
PMIx\_Get, [3](#), [9](#), [29](#), [64](#), [69](#), [72](#), [73](#), [74](#), [75](#), [77–83](#), [85–87](#), [89](#), [90](#), [94](#), [98](#), [100](#), [103](#), [106–108](#), [112](#),  
    [116–118](#), [165–167](#), [171](#), [172](#), [175–177](#), [191](#), [193](#), [194](#), [207](#), [216](#), [222](#), [229](#), [231](#), [233](#), [237](#),  
    [241](#), [260](#), [262](#), [272](#), [273](#), [275](#), [300](#), [304](#), [308](#), [338](#), [364](#), [366](#), [414](#), [423–425](#), [439](#), [457](#), [486](#),  
    [530](#), [531](#), [533](#), [536](#), [538](#), [542](#), [543](#), [548](#), [560](#), [562](#), [564](#), [569](#), [575](#), [576](#), [582](#), [583](#), [592](#)  
    PMIxClient.get (Python), [486](#)  
PMIx\_Get\_attribute\_name, [58](#), [506](#), [571](#)  
    PMIxClient.get\_attribute\_name (Python), [506](#)  
PMIx\_Get\_attribute\_string, [58](#), [506](#), [571](#)  
    PMIxClient.get\_attribute\_string (Python), [505](#)  
PMIx\_Get\_cpuset, [194](#), [500](#), [571](#)  
    PMIxClient.get\_cpuset (Python), [500](#)  
PMIx\_Get\_credential, [282](#), [284](#), [394](#), [493](#), [565](#), [570](#)  
    PMIxClient.get\_credential (Python), [493](#)  
PMIx\_Get\_credential\_nb, [283](#)  
PMIx\_Get\_nb, [54](#), [75](#), [562](#)  
PMIx\_Get\_relative\_locality, [191](#), [193](#), [304](#), [339](#), [500](#), [533](#), [571](#), [591](#)  
    PMIxClient.get\_relative\_locality (Python), [500](#)  
PMIx\_Get\_version, [11](#), [61](#), [483](#), [562](#)  
    PMIxClient.get\_version (Python), [483](#)  
PMIx\_Group\_construct, [232](#), [233](#), [238](#), [240](#), [241](#), [244](#), [494](#), [560](#), [571](#)

PMIxCliient.group\_construct (Python), 494  
 PMIx\_Group\_construct\_nb, [241](#), 244, 571  
 PMIx\_Group\_destruct, [235](#), [244](#), 245, 247, 257, 496, 571  
     PMIxCliient.group\_destruct (Python), 496  
 PMIx\_Group\_destruct\_nb, [245](#), 247, 571  
 PMIx\_Group\_invite, [234](#), [247](#), 249, 250, 252, 495, 571  
     PMIxCliient.group\_invite (Python), 494  
 PMIx\_Group\_invite\_nb, [250](#), 571  
 PMIx\_Group\_join, [234](#), 249, [252](#), 252–254, 256, 495, 571  
     PMIxCliient.group\_join (Python), 495  
 PMIx\_Group\_join\_nb, 252, [254](#), 256, 571  
 PMIx\_Group\_leave, [235](#), [256](#), 257, 258, 496, 571  
     PMIxCliient.group\_leave (Python), 496  
 PMIx\_Group\_leave\_nb, [257](#), 571  
 pmix\_hdlr\_reg\_cbfunc\_t, [55](#), 138, 450, 451, 588  
 pmix\_info\_cbfunc\_t, [53](#), [55](#), 55, 106, 196, 205, 213, 215, 220, 221, 242, 250, 255, 336, 377, 385,  
     [387](#), 388, 390, 391, 404, 407  
 PMIx\_Info\_directives\_string, [57](#), 504, 563  
     PMIxCliient.info\_directives\_string (Python), 503  
 PMIx\_Init, [8](#), [60](#), [61](#), [64](#), 86, 103, 108, 346, 418, 422, 432, 433, 440, 482, 530, 532, 537, 543, 552,  
     [563](#), 569, 592  
     PMIxCliient.init (Python), 482  
 PMIx\_Initialized, [60](#), 482, 562  
     PMIxCliient.initialized (Python), 482  
 pmix\_iof\_cbfunc\_t, [401](#), 449, 467  
     iofcbfunc (Python), 467  
 PMIx\_IOF\_channel\_string, [58](#), 505, 565  
     PMIxCliient.iof\_channel\_string (Python), 505  
 PMIx\_IOF\_deregister, [451](#), 522, 565, 570  
     PMIxTool.iof\_deregister (Python), 522  
 PMIx\_IOF\_pull, [365](#), [381](#), 416, 417, 421, 424, 425, 427, [449](#), 451, 522, 538, 542, 543, 548, 565,  
     [570](#)  
     PMIxTool.iof\_pull (Python), 521  
 PMIx\_IOF\_push, [365](#), [381](#), 416, 421, 424, 427–429, 431, [452](#), 454, 523, 565, 570, 581  
     PMIxTool.iof\_push (Python), 522  
 PMIx\_Job\_control, [201](#), [210](#), 211, 214–216, 389, 434, 492, 565, 570  
     PMIxCliient.job\_ctrl (Python), 492  
 PMIx\_Job\_control\_nb, [100](#), 209, [212](#), 306, 563  
 PMIx\_Job\_state\_string, [58](#), 505, 571  
     PMIxCliient.job\_state\_string (Python), 505  
 PMIx\_Link\_state\_string, [58](#), 506, 571  
     PMIxCliient.link\_state\_string (Python), 506  
 PMIx\_Load\_topology, [190](#), 499, 571, 591

PMIXClient.load\_topology (Python), 499  
PMIX\_Log, 178, [223](#), 225, 229, 418, 437, 491, 565, 588  
PMIXClient.log (Python), 491  
PMIX\_Log\_nb, [226](#), 229, 563  
PMIX\_Lookup, 119, [124](#), 125, 127, 486, 487, 562  
PMIXClient.lookup (Python), 487  
pmix\_lookup\_cbfunc\_t, [130](#), 130, 358  
PMIX\_Lookup\_nb, [125](#), 130, 131, 562  
pmix\_modex\_cbfunc\_t, 53, 350, [353](#), 353, 354  
pmix\_notification\_fn\_t, 138, [143](#), 143, 468  
evhandler (Python), 467  
PMIX\_Notify\_event, [145](#), 375, 498, 536, 541, 549, 553, 554, 563, 570  
PMIXClient.notify\_event (Python), 497  
pmix\_op\_cbfunc\_t, [54](#), 54, 121, 133, 145, 146, 149, 185, 188, 226, 246, 257, 278–280, 297,  
317–321, 328, 334, 335, 337, 345–347, 349, 356, 360, 367, 369, 371, 373, 374, 383, 399,  
402, 452  
PMIX\_Parse\_cpuset\_string, [193](#), 339, 501, 571  
PMIXClient.parse\_cpuset\_string (Python), 500  
PMIX\_Persistence\_string, [57](#), 503, 563  
PMIXClient.persistence\_string (Python), 503  
PMIX\_Proc\_state\_string, [56](#), 502, 563  
PMIXClient.proc\_state\_string (Python), 502  
PMIX\_Process\_monitor, 201, [217](#), 221, 493, 565, 570  
PMIXClient.monitor (Python), 492  
PMIX\_Process\_monitor\_nb, [219](#), 222, 563  
PMIX\_Progress, 63, [66](#), 292, 294, 444, 507, 571, 580  
PMIXClient.progress (Python), 507  
PMIX\_Publish, [119](#), 121–124, 357, 487, 562  
PMIXClient.publish (Python), 486  
PMIX\_Publish\_nb, [121](#), 124, 562  
PMIX\_Put, 29, 68–72, 80, [95](#), 95–98, 116, 182, 241, 249, 323, 324, 351, 355, 485, 530, 531, 562  
PMIXClient.put (Python), 484  
PMIX\_Query\_info, 9, [101](#), 101, 106, 110, 112, 115–117, 231, 233, 272, 273, 408, 412, 432, 433,  
457, 491, 536, 542, 545, 576  
PMIXClient.query (Python), 490  
PMIX\_Query\_info\_nb, 100, [106](#), 106, 110, 112, 115, 118, 181, 308, 329, 548, 563, 564  
PMIX\_Register\_attributes, [329](#), 515, 570, 571  
PMIXServer.register\_attributes (Python), 514  
PMIX\_Register\_event\_handler, 100, [137](#), 177, 418, 438, 497, 536, 541, 545, 549, 553, 554, 557,  
563, 570, 587  
PMIXClient.register\_event\_handler (Python), 496  
pmix\_release\_cbfunc\_t, [53](#), 53  
PMIX\_Resolve\_nodes, [116](#), 490, 562

PMIXClient.resolve\_nodes (Python), 490  
PMIX\_Resolve\_peers, 88, **115**, 303, 490, 532, 562  
PMIXClient.resolve\_peers (Python), 489  
PMIX\_Scope\_string, **56**, 502, 563  
PMIXClient.scope\_string (Python), 502  
pmix\_server\_abort\_fn\_t, **348**, 470  
clientaborted (Python), 469  
pmix\_server\_alloc\_fn\_t, **384**, 477  
allocate (Python), 477  
pmix\_server\_client\_connected2\_fn\_t, 54, 281, 320, 321, 344, **345**, 345–347, 469, 571, 588, 589  
clientconnected2 (Python), 468  
pmix\_server\_client\_finalized\_fn\_t, **347**, 348, 469  
clientfinalized (Python), 469  
PMIX\_server\_collect\_inventory, **336**, 338, 516, 565  
PMIXServer.collect\_inventory (Python), 516  
pmix\_server\_connect\_fn\_t, 182, **367**, 368, 370, 473  
connect (Python), 473  
PMIX\_server\_define\_process\_set, 231, **341**, 517, 571  
PMIXServer.define\_process\_set (Python), 517  
PMIX\_server\_delete\_process\_set, 231, **342**, 518, 571  
PMIXServer.delete\_process\_set (Python), 517  
PMIX\_server\_deliver\_inventory, **337**, 517, 565  
PMIXServer.deliver\_inventory (Python), 516  
PMIX\_server\_deregister\_client, **321**, 513, 562  
PMIXServer.deregister\_client (Python), 513  
pmix\_server\_deregister\_events\_fn\_t, **372**, 475  
deregister\_events (Python), 475  
PMIX\_server\_deregister\_nspace, **317**, 321, 511, 562  
PMIXServer.deregister\_nspace (Python), 511  
PMIX\_server\_deregister\_resources, **318**, 512, 518, 572  
PMIXServer.deregister\_resources (Python), 512, 518  
pmix\_server\_disconnect\_fn\_t, **368**, 370, 474  
disconnect (Python), 474  
pmix\_server\_dmodex\_req\_fn\_t, 89, 98, 99, **353**, 353, 471, 565, 567  
dmodex (Python), 471  
PMIX\_server\_dmodex\_request, **322**, 323, 324, 514, 562  
PMIXServer.dmodex\_request (Python), 513  
pmix\_server\_fabric\_fn\_t, 268, 275, **406**, 481, 571  
fabric (Python), 481  
pmix\_server\_fenceb\_fn\_t, **350**, 352, 353, 470, 567  
fence (Python), 470  
PMIX\_server\_finalize, **293**, 508, 562  
PMIXServer.finalize (Python), 508

PMIx\_server\_generate\_cpuset\_string, 194, [339](#), 510, 572  
     PMIxServer.generate\_cpuset\_string (Python), 510  
 PMIx\_server\_generate\_locality\_string, 190, 191, [338](#), 510, 571  
     PMIxServer.generate\_locality\_string (Python), 509  
 pmix\_server\_get\_cred\_fn\_t, [393](#), 397, 479  
     get\_credential (Python), 478  
 pmix\_server\_grp\_fn\_t, [403](#), 481, 571  
     group (Python), 480  
 PMIx\_server\_init, 60, [289](#), 293, 329, 343, 409, 410, 414, 508, 549, 562, 571  
     PMIxServer.init (Python), 507  
 PMIx\_server\_IOF\_deliver, [335](#), 426, 516, 565  
     PMIxServer.iof\_deliver (Python), 515  
 pmix\_server\_iof\_fn\_t, [398](#), 480  
     iof\_pull (Python), 479  
 pmix\_server\_job\_control\_fn\_t, [387](#), 478  
     job\_control (Python), 477  
 pmix\_server\_listener\_fn\_t, [375](#)  
 pmix\_server\_log\_fn\_t, [382](#), 477  
     log (Python), 476  
 pmix\_server\_lookup\_fn\_t, [357](#), 472  
     lookup (Python), 472  
 pmix\_server\_module\_t, 290, 292, 329, 330, [343](#), 343, 344, 508  
 pmix\_server\_monitor\_fn\_t, [390](#), 478  
     monitor (Python), 478  
 pmix\_server\_notify\_event\_fn\_t, 144, 148, [374](#), 375, 475  
     notify\_event (Python), 475  
 pmix\_server\_publish\_fn\_t, [355](#), 471  
     publish (Python), 471  
 pmix\_server\_query\_fn\_t, [377](#), 476  
     query (Python), 475  
 PMIx\_server\_register\_client, 281, [319](#), 321, 346, 348, 512, 562  
     PMIxServer.register\_client (Python), 512  
 pmix\_server\_register\_events\_fn\_t, [370](#), 474  
     register\_events (Python), 474  
 PMIx\_server\_register\_nspace, 11, 54, 295, [296](#), 297, 298, 307, 308, 311, 318, 335, 339, 511, 562, 564, 578  
     PMIxServer.register\_nspace (Python), 510  
 PMIx\_server\_register\_resources, 299, 302, 303, [317](#), 512, 518, 572  
     PMIxServer.register\_resources (Python), 511, 518  
 PMIx\_server\_setup\_application, [324](#), 328, 334, 338, 514, 563, 567  
     PMIxServer.setup\_application (Python), 514  
 PMIx\_server\_setup\_fork, [322](#), 513, 562  
     PMIxServer.setup\_fork (Python), 513

PMIx\_server\_setup\_local\_support, [333](#), [515](#), [563](#)  
    PMIxServer.setup\_local\_support (Python), [515](#)  
pmix\_server\_spawn\_fn\_t, [181](#), [362](#), [419](#), [473](#)  
    spawn (Python), [473](#)  
pmix\_server\_stdin\_fn\_t, [402](#), [480](#)  
    iof\_push (Python), [480](#)  
pmix\_server\_tool\_connection\_fn\_t, [281](#), [379](#), [409](#), [476](#)  
    tool\_connected (Python), [476](#)  
pmix\_server\_unpublish\_fn\_t, [360](#), [472](#)  
    unpublish (Python), [472](#)  
pmix\_server\_validate\_cred\_fn\_t, [395](#), [479](#)  
    validate\_credential (Python), [479](#)  
pmix\_setup\_application\_cbfunc\_t, [325](#), [328](#)  
PMIx\_Spawn, [84](#), [86](#), [87](#), [163](#), [164](#), [169](#), [174](#), [175](#), [178](#), [204](#), [207](#), [304](#), [306](#), [322](#), [362](#), [363](#), [365](#),  
    [366](#), [386](#), [414](#), [416](#), [419](#), [420](#), [423–425](#), [431–436](#), [439](#), [440](#), [473](#), [488](#), [534](#), [536](#), [538](#), [541](#),  
    [544](#), [545](#), [547](#), [550](#), [562](#), [568](#), [579](#)  
    PMIxClient.spawn (Python), [488](#)  
pmix\_spawn\_cbfunc\_t, [169](#), [181](#), [181](#), [362](#)  
PMIx\_Spawn\_nb, [169](#), [178](#), [181](#), [562](#)  
PMIx\_Store\_internal, [95](#), [96](#), [484](#), [562](#)  
    PMIxClient.store\_internal (Python), [484](#)  
PMIx\_tool\_attach\_to\_server, [411](#), [414](#), [423](#), [444](#), [446](#), [520](#), [541](#), [572](#), [588](#)  
    PMIxTool.attach\_to\_server (Python), [520](#)  
PMIx\_tool\_connect\_to\_server, [565](#), [590](#)  
pmix\_tool\_connection\_cbfunc\_t, [380](#), [381](#), [382](#)  
PMIx\_tool\_disconnect, [445](#), [520](#), [572](#)  
    PMIxTool.disconnect (Python), [519](#)  
PMIx\_tool\_finalize, [444](#), [519](#), [563](#)  
    PMIxTool.finalize (Python), [519](#)  
PMIx\_tool\_get\_servers, [447](#), [521](#), [572](#)  
    PMIxTool.get\_servers (Python), [520](#)  
PMIx\_tool\_init, [60](#), [408](#), [411](#), [413–415](#), [422–424](#), [426](#), [441](#), [444](#), [519](#), [534](#), [541](#), [545](#), [563](#)  
    PMIxTool.init (Python), [519](#)  
PMIx\_tool\_set\_server, [410](#), [423](#), [424](#), [447](#), [448](#), [521](#), [572](#)  
    PMIxTool.set\_server (Python), [521](#)  
PMIx\_Unpublish, [132](#), [133](#), [134](#), [488](#), [562](#)  
    PMIxClient.unpublish (Python), [487](#)  
PMIx\_Unpublish\_nb, [133](#), [562](#)  
PMIx\_Validate\_credential, [285](#), [494](#), [565](#), [570](#)  
    PMIxClient.validate\_credential (Python), [493](#)  
PMIx\_Validate\_credential\_nb, [286](#)  
pmix\_validation\_cbfunc\_t, [287](#), [396](#), [397](#)  
pmix\_value\_cbfunc\_t, [54](#), [54](#)

pmix\_evhdlr\_reg\_cbfunc\_t  
(Deprecated), [588](#)  
pmix\_server\_client\_connected\_fn\_t  
(Deprecated), [344](#), [588](#)  
PMIx\_tool\_connect\_to\_server  
(Deprecated), [588](#)

Unofficial Draft

# Index of Support Macros

---

PMIX\_APP\_CONSTRUCT, [179](#)  
PMIX\_APP\_CREATE, [180](#)  
PMIX\_APP\_DESTRUCT, [179](#)  
PMIX\_APP\_FREE, [180](#)  
PMIX\_APP\_INFO\_CREATE, [180](#), [565](#), [567](#)  
PMIX\_APP\_RELEASE, [180](#)  
PMIX\_ARGV\_APPEND, [47](#)  
PMIX\_ARGV\_APPEND\_UNIQUE, [48](#)  
PMIX\_ARGV\_COPY, [50](#)  
PMIX\_ARGV\_COUNT, [50](#)  
PMIX\_ARGV\_FREE, [48](#)  
PMIX\_ARGV\_JOIN, [49](#)  
PMIX\_ARGV\_PREPEND, [47](#)  
PMIX\_ARGV\_SPLIT, [49](#)  
PMIX\_BYTE\_OBJECT\_CONSTRUCT, [44](#)  
PMIX\_BYTE\_OBJECT\_CREATE, [44](#)  
PMIX\_BYTE\_OBJECT\_DESTRUCT, [44](#)  
PMIX\_BYTE\_OBJECT\_FREE, [45](#)  
PMIX\_BYTE\_OBJECT\_LOAD, [45](#)  
PMIX\_CHECK\_KEY, [18](#)  
PMIX\_CHECK\_NAMESPACE, [19](#)  
PMIX\_CHECK\_PROCID, [23](#)  
PMIX\_CHECK\_RANK, [21](#)  
PMIX\_CHECK\_RESERVED\_KEY, [18](#), [588](#)  
PMIX\_COORD\_CONSTRUCT, [265](#)  
PMIX\_COORD\_CREATE, [265](#)  
PMIX\_COORD\_DESTRUCT, [265](#)  
PMIX\_COORD\_FREE, [265](#)  
PMIX\_CPUSET\_CONSTRUCT, [340](#)  
PMIX\_CPUSET\_CREATE, [340](#)  
PMIX\_CPUSET\_DESTRUCT, [340](#)  
PMIX\_CPUSET\_FREE, [341](#)  
PMIX\_DATA\_ARRAY\_CONSTRUCT, [46](#)  
PMIX\_DATA\_ARRAY\_CREATE, [46](#)  
PMIX\_DATA\_ARRAY\_DESTRUCT, [46](#)  
PMIX\_DATA\_ARRAY\_FREE, [46](#)  
PMIX\_DATA\_BUFFER\_CONSTRUCT, [151](#), [153](#), [155](#)  
PMIX\_DATA\_BUFFER\_CREATE, [151](#), [153](#), [155](#)



PMIX\_DATA\_BUFFER\_DESTRUCT, [151](#)  
PMIX\_DATA\_BUFFER\_LOAD, [152](#)  
PMIX\_DATA\_BUFFER\_RELEASE, [151](#)  
PMIX\_DATA\_BUFFER\_UNLOAD, [152](#), [295](#)  
PMIX\_DEVICE\_DIST\_CONSTRUCT, [199](#)  
PMIX\_DEVICE\_DIST\_CREATE, [199](#)  
PMIX\_DEVICE\_DIST\_DESTRUCT, [199](#)  
PMIX\_DEVICE\_DIST\_FREE, [200](#)  
PMIX\_ENDPOINT\_CONSTRUCT, [263](#)  
PMIX\_ENDPOINT\_CREATE, [263](#)  
PMIX\_ENDPOINT\_DESTRUCT, [263](#)  
PMIX\_ENDPOINT\_FREE, [264](#)  
PMIX\_ENVAR\_CONSTRUCT, [42](#)  
PMIX\_ENVAR\_CREATE, [43](#)  
PMIX\_ENVAR\_DESTRUCT, [14](#), [42](#)  
PMIX\_ENVAR\_FREE, [43](#)  
PMIX\_ENVAR\_LOAD, [43](#)  
PMIX\_FABRIC\_CONSTRUCT, [272](#)  
PMIX\_GEOMETRY\_CONSTRUCT, [266](#)  
PMIX\_GEOMETRY\_CREATE, [267](#)  
PMIX\_GEOMETRY\_DESTRUCT, [267](#)  
PMIX\_GEOMETRY\_FREE, [267](#)  
PMIx\_Heartbeat, [221](#), [563](#)  
PMIX\_INFO\_CONSTRUCT, [34](#)  
PMIX\_INFO\_CREATE, [34](#), [39](#), [41](#)  
PMIX\_INFO\_DESTRUCT, [34](#)  
PMIX\_INFO\_FREE, [35](#)  
PMIX\_INFO\_IS\_END, [41](#), [565](#), [567](#)  
PMIX\_INFO\_IS\_OPTIONAL, [41](#)  
PMIX\_INFO\_IS\_REQUIRED, [39](#), [40](#), [40](#)  
PMIX\_INFO\_LIST\_ADD, [37](#), [588](#)  
PMIX\_INFO\_LIST\_CONVERT, [38](#), [588](#)  
PMIX\_INFO\_LIST\_RELEASE, [38](#), [588](#)  
PMIX\_INFO\_LIST\_START, [37](#), [37](#), [38](#), [588](#)  
PMIX\_INFO\_LIST\_XFER, [38](#), [588](#)  
PMIX\_INFO\_LOAD, [35](#)  
PMIX\_INFO\_OPTIONAL, [40](#)  
PMIX\_INFO\_PROCESSED, [41](#), [588](#)  
PMIX\_INFO\_REQUIRED, [39](#), [40](#)  
PMIX\_INFO\_TRUE, [36](#)  
PMIX\_INFO\_WAS\_PROCESSED, [41](#), [588](#)  
PMIX\_INFO\_XFER, [36](#), [308](#)  
PMIX\_LOAD\_KEY, [18](#)

PMIX\_LOAD\_NAMESPACE, [20](#)  
PMIX\_LOAD\_PROCID, [23](#), [24](#)  
PMIX\_MULTICLUSTER\_NAMESPACE\_CONSTRUCT, [25](#)  
PMIX\_MULTICLUSTER\_NAMESPACE\_PARSE, [25](#)  
PMIX\_NAMESPACE\_INVALID, [20](#), [592](#)  
PMIX\_PDATA\_CONSTRUCT, [128](#)  
PMIX\_PDATA\_CREATE, [128](#)  
PMIX\_PDATA\_DESTRUCT, [128](#)  
PMIX\_PDATA\_FREE, [129](#)  
PMIX\_PDATA\_LOAD, [129](#)  
PMIX\_PDATA\_RELEASE, [128](#)  
PMIX\_PDATA\_XFER, [130](#)  
PMIX\_PROC\_CONSTRUCT, [22](#)  
PMIX\_PROC\_CREATE, [22](#)  
PMIX\_PROC\_DESTRUCT, [22](#)  
PMIX\_PROC\_FREE, [23](#), [116](#)  
PMIX\_PROC\_INFO\_CONSTRUCT, [27](#)  
PMIX\_PROC\_INFO\_CREATE, [28](#)  
PMIX\_PROC\_INFO\_DESTRUCT, [27](#)  
PMIX\_PROC\_INFO\_FREE, [28](#)  
PMIX\_PROC\_INFO\_RELEASE, [28](#)  
PMIX\_PROC\_LOAD, [23](#)  
PMIX\_PROC\_RELEASE, [22](#)  
PMIX\_PROCID\_INVALID, [24](#), [592](#)  
PMIX\_PROCID\_XFER, [24](#), [592](#)  
PMIX\_QUERY\_CONSTRUCT, [113](#)  
PMIX\_QUERY\_CREATE, [114](#)  
PMIX\_QUERY\_DESTRUCT, [114](#)  
PMIX\_QUERY\_FREE, [114](#)  
PMIX\_QUERY\_QUALIFIERS\_CREATE, [115](#), [565](#), [567](#)  
PMIX\_QUERY\_RELEASE, [114](#)  
PMIX\_RANK\_IS\_VALID, [21](#), [592](#)  
PMIX\_REGATTR\_CONSTRUCT, [331](#)  
PMIX\_REGATTR\_CREATE, [332](#)  
PMIX\_REGATTR\_DESTRUCT, [332](#)  
PMIX\_REGATTR\_FREE, [332](#)  
PMIX\_REGATTR\_LOAD, [333](#)  
PMIX\_REGATTR\_XFER, [333](#)  
PMIX\_SETENV, [50](#)  
PMIX\_SYSTEM\_EVENT, [141](#)  
PMIX\_TOPOLOGY\_CONSTRUCT, [192](#)  
PMIX\_TOPOLOGY\_CREATE, [192](#)  
PMIX\_TOPOLOGY\_DESTRUCT, [192](#)

PMIX\_TOPOLOGY\_FREE, [192](#)  
PMIX\_VALUE\_CONSTRUCT, [30](#)  
PMIX\_VALUE\_CREATE, [31](#)  
PMIX\_VALUE\_DESTRUCT, [30](#), [74](#), [78](#), [575](#)  
PMIX\_VALUE\_FREE, [31](#)  
PMIX\_VALUE\_GET\_NUMBER, [33](#)  
PMIX\_VALUE\_LOAD, [31](#)  
PMIX\_VALUE\_RELEASE, [31](#)  
PMIX\_VALUE\_UNLOAD, [32](#)  
PMIX\_VALUE\_XFER, [33](#)

Unofficial Draft

# Index of Data Structures

---

pmix\_alloc\_directive\_t, [52](#), [57](#), [202](#), [205](#), [209](#), [209](#), [385](#), [463](#), [504](#)  
pmix\_app\_t, [47](#), [48](#), [51](#), [164](#), [166](#), [169–171](#), [176](#), [178](#), [178–180](#), [362](#), [363](#), [365](#), [416](#), [418–420](#), [422](#),  
[423](#), [432](#), [435](#), [440](#), [464](#), [537–539](#), [543](#), [544](#), [547](#), [565](#), [567](#), [587](#)  
pmix\_bind\_envelope\_t, [194](#), [194](#), [465](#), [572](#)  
pmix\_byte\_object\_t, [44](#), [44](#), [45](#), [52](#), [158](#), [159](#), [282](#), [283](#), [285](#), [287](#), [335](#), [395](#), [396](#), [402](#), [452](#), [462](#)  
pmix\_coord\_t, [52](#), [264](#), [264–266](#), [465](#), [572](#)  
pmix\_coord\_view\_t, [267](#), [465](#), [572](#)  
pmix\_cpuset\_t, [52](#), [195](#), [196](#), [338](#), [339](#), [340](#), [340](#), [341](#), [464](#), [572](#)  
pmix\_data\_array\_t, [29](#), [38](#), [45](#), [45](#), [46](#), [52](#), [88](#), [104](#), [105](#), [109](#), [111](#), [112](#), [118](#), [203](#), [206](#), [208](#), [233](#),  
[236](#), [270](#), [272–276](#), [298](#), [299](#), [301–303](#), [311](#), [313](#), [314](#), [326](#), [378](#), [387](#), [407](#), [433](#), [441](#), [463](#),  
[539](#), [544](#), [547](#), [560](#), [565](#), [567](#), [583–586](#)  
pmix\_data\_buffer\_t, [150](#), [150–154](#), [157–159](#)  
pmix\_data\_range\_t, [52](#), [57](#), [123](#), [123](#), [146](#), [374](#), [462](#), [503](#)  
pmix\_data\_type\_t, [32](#), [33](#), [35](#), [37](#), [46](#), [51](#), [51](#), [52](#), [57](#), [129](#), [153](#), [155–157](#), [333](#), [461](#), [504](#)  
pmix\_device\_distance\_t, [52](#), [195](#), [197](#), [198](#), [198–200](#), [305](#), [465](#), [572](#), [585](#)  
pmix\_device\_type\_t, [53](#), [59](#), [197](#), [197](#), [198](#), [275](#), [465](#), [507](#), [572](#)  
pmix\_endpoint\_t, [53](#), [262](#), [262–264](#), [275](#), [464](#), [583](#)  
pmix\_envar\_t, [14](#), [42](#), [42](#), [43](#), [52](#), [463](#), [550](#)  
pmix\_fabric\_operation\_t, [268](#), [268](#), [406](#)  
pmix\_fabric\_t, [262](#), [268](#), [269](#), [269](#), [272](#), [273](#), [276–280](#), [407](#), [464](#), [572](#), [582](#)  
pmix\_geometry\_t, [52](#), [261](#), [266](#), [266](#), [267](#), [274](#), [465](#), [572](#), [583](#), [584](#)  
pmix\_group\_operation\_t, [404](#), [406](#), [406](#), [572](#)  
pmix\_group\_opt\_t, [252](#), [255](#), [256](#), [256](#), [495](#), [572](#)  
pmix\_info\_directives\_t, [39](#), [39](#), [52](#), [57](#), [463](#), [503](#)  
pmix\_info\_t, [4](#), [5](#), [9](#), [18](#), [34](#), [34–41](#), [52](#), [55](#), [61](#), [64](#), [66](#), [101](#), [105](#), [112](#), [113](#), [115](#), [118](#), [120](#), [122–125](#),  
[143](#), [146](#), [149](#), [180](#), [195](#), [196](#), [202](#), [203](#), [205](#), [206](#), [208–210](#), [212](#), [215](#), [216](#), [218](#), [221](#), [225](#),  
[228](#), [230](#), [238](#), [240](#), [242](#), [244](#), [246–248](#), [250](#), [252](#), [255–257](#), [269](#), [270](#), [273–275](#), [282](#), [283](#),  
[285](#), [287](#), [290](#), [292](#), [297–299](#), [301](#), [302](#), [307](#), [308](#), [311](#), [313](#), [314](#), [325](#), [326](#), [331](#), [333](#),  
[335–337](#), [342](#), [345](#), [365](#), [374](#), [380](#), [381](#), [384](#), [386](#), [387](#), [390](#), [391](#), [398](#), [399](#), [401](#), [407](#), [416](#),  
[418–420](#), [422](#), [425](#), [432](#), [440](#), [442](#), [446](#), [448](#), [449](#), [451](#), [452](#), [463](#), [466](#), [537–539](#), [542–544](#),  
[547](#), [555](#), [563](#), [565](#), [567](#), [576–579](#), [582–584](#)  
pmix\_iof\_channel\_t, [52](#), [58](#), [335](#), [399](#), [401](#), [429](#), [429](#), [449](#), [463](#), [505](#)  
pmix\_job\_state\_t, [28](#), [28](#), [52](#), [58](#), [464](#), [505](#), [572](#)  
pmix\_key\_t, [9](#), [17](#), [17](#), [73](#), [95](#), [333](#), [461](#)  
pmix\_link\_state\_t, [52](#), [58](#), [262](#), [268](#), [268](#), [271](#), [274](#), [464](#), [506](#), [572](#), [584](#)  
pmix\_locality\_t, [53](#), [191](#), [193](#), [193](#), [464](#), [572](#), [590](#), [591](#)  
pmix\_nspace\_t, [19](#), [19](#), [20](#), [23–25](#), [52](#), [181](#), [461](#), [462](#)  
pmix\_pdata\_t, [124](#), [125](#), [127](#), [127–131](#), [463](#)

pmix\_persistence\_t, 52, 57, **123**, 123, 462, 503  
pmix\_proc\_info\_t, **26**, 26–28, 52, 103, 104, 108, 109, 111, 378, 433, 441, 462, 539, 544, 547  
pmix\_proc\_state\_t, **25**, 25, 52, 56, 462, 502  
pmix\_proc\_t, 20, **21**, 21–24, 52, 64, 68, 70, 71, 75, 88, 112, 129, 139, 140, 142, 143, 146, 147, 153,  
154, 162, 163, 233, 236–238, 242, 247, 250, 253, 303, 320–323, 333, 335, 341, 345–347,  
349, 350, 354, 356, 358, 360, 362, 367, 369, 374, 377, 382, 383, 385, 388, 391, 393, 396,  
399, 401, 402, 404–406, 442, 444–448, 462, 560, 585, 586  
pmix\_query\_t, 52, **101**, 101, 103, 108, 112–115, 117, 377, 379, 464, 565, 567, 576  
pmix\_rank\_t, **20**, 20, 21, 23, 24, 52, 462  
pmix\_regattr\_t, 52, 118, 329, **330**, 330–333, 464, 570, 572, 578  
pmix\_scope\_t, 52, 56, **96**, 96, 462, 502  
pmix\_status\_t, **15**, 15, 32, 33, 37, 38, 47, 48, 50, 52, 55, 56, 138, 141, 143, 146, 149, 197, 324, 328,  
371, 373, 374, 382, 395, 398, 461, 475, 501, 555  
pmix\_storage\_access\_type\_t, **457**, 457, 592  
pmix\_storage\_accessibility\_t, **456**, 456, 592  
pmix\_storage\_medium\_t, **455**, 455, 456, 592  
pmix\_storage\_persistence\_t, **456**, 456, 592  
pmix\_topology\_t, 53, 190, **191**, 191, 192, 195, 196, 572  
pmix\_value\_t, 9, **29**, 29–33, 52, 54, 74, 75, 78, 95, 463, 575

# Index of Constants

---

PMIX\_ALLOC\_DIRECTIVE, [52](#)  
PMIX\_ALLOC\_EXTEND, [209](#)  
PMIX\_ALLOC\_EXTERNAL, [209](#)  
PMIX\_ALLOC\_NEW, [209](#)  
PMIX\_ALLOC\_REAQUIRE, [209](#)  
PMIX\_ALLOC\_RELEASE, [209](#)  
PMIX\_APP, [52](#)  
PMIX\_APP\_WILDCARD, [14](#)  
PMIX\_BOOL, [51](#)  
PMIX\_BUFFER, [52](#)  
PMIX\_BYTE, [51](#)  
PMIX\_BYTE\_OBJECT, [52](#)  
PMIX\_COMMAND, [52](#)  
PMIX\_COMPRESSED\_BYTE\_OBJECT, [52](#)  
PMIX\_COMPRESSED\_STRING, [52](#)  
PMIX\_COORD, [52](#)  
PMIX\_COORD\_LOGICAL\_VIEW, [268](#)  
PMIX\_COORD\_PHYSICAL\_VIEW, [268](#)  
PMIX\_COORD\_VIEW\_UNDEF, [268](#)  
PMIX\_CPUBIND\_PROCESS, [194](#)  
PMIX\_CPUBIND\_THREAD, [194](#)  
PMIX\_DATA\_ARRAY, [52](#)  
PMIX\_DATA\_RANGE, [52](#)  
PMIX\_DATA\_TYPE, [52](#)  
PMIX\_DATA\_TYPE\_MAX, [53](#)  
PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY, [439](#)  
PMIX\_DEBUGGER\_RELEASE, [439](#)  
PMIX\_DEVICE\_DIST, [52](#)  
PMIX\_DEVTYPE, [53](#)  
PMIX\_DEVTYPE\_BLOCK, [198](#)  
PMIX\_DEVTYPE\_COPROC, [198](#)  
PMIX\_DEVTYPE\_DMA, [198](#)  
PMIX\_DEVTYPE\_GPU, [198](#)  
PMIX\_DEVTYPE\_NETWORK, [198](#)  
PMIX\_DEVTYPE\_OPENFABRICS, [198](#)  
PMIX\_DEVTYPE\_UNKNOWN, [198](#)  
PMIX\_DOUBLE, [51](#)  
PMIX\_ENDPOINT, [53](#)

PMIX\_ENVAR, [52](#)  
PMIX\_ERR\_BAD\_PARAM, [16](#)  
PMIX\_ERR\_COMM\_FAILURE, [16](#)  
PMIX\_ERR\_CONFLICTING\_CLEANUP\_DIRECTIVES, [215](#)  
PMIX\_ERR\_DUPLICATE\_KEY, [122](#)  
PMIX\_ERR\_EMPTY, [16](#)  
PMIX\_ERR\_EVENT\_REGISTRATION, [140](#)  
PMIX\_ERR\_EXISTS, [15](#)  
PMIX\_ERR\_EXISTS\_OUTSIDE\_SCOPE, [15](#)  
PMIX\_ERR\_INIT, [16](#)  
PMIX\_ERR\_INVALID\_CRED, [15](#)  
PMIX\_ERR\_INVALID\_OPERATION, [16](#)  
PMIX\_ERR\_IOF\_COMPLETE, [430](#)  
PMIX\_ERR\_IOF\_FAILURE, [430](#)  
PMIX\_ERR\_JOB\_ABORTED, [438](#)  
PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG, [438](#)  
PMIX\_ERR\_JOB\_ABORTED\_BY\_SYS\_EVENT, [439](#)  
PMIX\_ERR\_JOB\_ALLOC\_FAILED, [174](#)  
PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE, [174](#)  
PMIX\_ERR\_JOB\_CANCELED, [438](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_LAUNCH, [174](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_MAP, [174](#)  
PMIX\_ERR\_JOB\_KILLED\_BY\_CMD, [438](#)  
PMIX\_ERR\_JOB\_NO\_EXE\_SPECIFIED, [174](#)  
PMIX\_ERR\_JOB\_NON\_ZERO\_TERM, [439](#)  
PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED, [439](#)  
PMIX\_ERR\_JOB\_TERM\_WO\_SYNC, [439](#)  
PMIX\_ERR\_LOST\_CONNECTION, [16](#)  
PMIX\_ERR\_NO\_PERMISSIONS, [16](#)  
PMIX\_ERR\_NOMEM, [16](#)  
PMIX\_ERR\_NOT\_FOUND, [16](#)  
PMIX\_ERR\_NOT\_SUPPORTED, [16](#)  
PMIX\_ERR\_OUT\_OF\_RESOURCE, [16](#)  
PMIX\_ERR\_PACK\_FAILURE, [16](#)  
PMIX\_ERR\_PARAM\_VALUE\_NOT\_SUPPORTED, [16](#)  
PMIX\_ERR\_PARTIAL\_SUCCESS, [17](#)  
PMIX\_ERR\_PROC\_CHECKPOINT, [215](#)  
PMIX\_ERR\_PROC\_MIGRATE, [215](#)  
PMIX\_ERR\_PROC\_RESTART, [215](#)  
PMIX\_ERR\_PROC\_TERM\_WO\_SYNC, [438](#)  
PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION, [330](#)  
PMIX\_ERR\_RESOURCE\_BUSY, [16](#)  
PMIX\_ERR\_TIMEOUT, [16](#)

PMIX\_ERR\_TYPE\_MISMATCH, [15](#)  
PMIX\_ERR\_UNKNOWN\_DATA\_TYPE, [15](#)  
PMIX\_ERR\_UNPACK\_FAILURE, [16](#)  
PMIX\_ERR\_UNPACK\_INADEQUATE\_SPACE, [15](#)  
PMIX\_ERR\_UNPACK\_READ\_PAST\_END\_OF\_BUFFER, [16](#)  
PMIX\_ERR\_UNREACH, [16](#)  
PMIX\_ERR\_WOULD\_BLOCK, [15](#)  
PMIX\_ERROR, [15](#)  
PMIX\_EVENT\_ACTION\_COMPLETE, [149](#)  
PMIX\_EVENT\_ACTION\_DEFERRED, [149](#)  
PMIX\_EVENT\_JOB\_END, [438](#)  
PMIX\_EVENT\_JOB\_START, [438](#)  
PMIX\_EVENT\_NO\_ACTION\_TAKEN, [149](#)  
PMIX\_EVENT\_NODE\_DOWN, [141](#)  
PMIX\_EVENT\_NODE\_OFFLINE, [141](#)  
PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN, [149](#)  
PMIX\_EVENT\_PROC\_TERMINATED, [438](#)  
PMIX\_EVENT\_SESSION\_END, [438](#)  
PMIX\_EVENT\_SESSION\_START, [438](#)  
PMIX\_EVENT\_SYS\_BASE, [141](#)  
PMIX\_EVENT\_SYS\_OTHER, [141](#)  
PMIX\_EXTERNAL\_ERR\_BASE, [17](#)  
PMIX\_FABRIC\_REQUEST\_INFO, [268](#)  
PMIX\_FABRIC\_UPDATE\_ENDPOINTS, [262](#)  
PMIX\_FABRIC\_UPDATE\_INFO, [268](#)  
PMIX\_FABRIC\_UPDATE\_PENDING, [262](#)  
PMIX\_FABRIC\_UPDATED, [262](#)  
PMIX\_FLOAT, [51](#)  
PMIX\_FWD\_ALL\_CHANNELS, [429](#)  
PMIX\_FWD\_NO\_CHANNELS, [429](#)  
PMIX\_FWD\_STDDIAG\_CHANNEL, [429](#)  
PMIX\_FWD\_STDERR\_CHANNEL, [429](#)  
PMIX\_FWD\_STDIN\_CHANNEL, [429](#)  
PMIX\_FWD\_STDOUT\_CHANNEL, [429](#)  
PMIX\_GEOMETRY, [52](#)  
PMIX\_GLOBAL, [96](#)  
PMIX\_GROUP\_ACCEPT, [256](#)  
PMIX\_GROUP\_CONSTRUCT, [406](#)  
PMIX\_GROUP\_CONSTRUCT\_ABORT, [236](#)  
PMIX\_GROUP\_CONSTRUCT\_COMPLETE, [236](#)  
PMIX\_GROUP\_CONTEXT\_ID\_ASSIGNED, [236](#)  
PMIX\_GROUP\_DECLINE, [256](#)  
PMIX\_GROUP\_DESTRUCT, [406](#)



PMIX\_GROUP\_INVITE\_ACCEPTED, [236](#)  
PMIX\_GROUP\_INVITE\_DECLINED, [236](#)  
PMIX\_GROUP\_INVITE\_FAILED, [236](#)  
PMIX\_GROUP\_INVITED, [235](#)  
PMIX\_GROUP\_LEADER\_FAILED, [236](#)  
PMIX\_GROUP\_LEADER\_SELECTED, [236](#)  
PMIX\_GROUP\_LEFT, [235](#)  
PMIX\_GROUP\_MEMBER\_FAILED, [235](#)  
PMIX\_GROUP\_MEMBERSHIP\_UPDATE, [236](#)  
PMIX\_INFO, [52](#)  
PMIX\_INFO\_ARRAY\_END, [39](#)  
PMIX\_INFO\_DIR\_RESERVED, [39](#)  
PMIX\_INFO\_DIRECTIVES, [52](#)  
PMIX\_INFO\_REQD, [39](#)  
PMIX\_INFO\_REQD\_PROCESSED, [39](#)  
PMIX\_INT, [51](#)  
PMIX\_INT16, [51](#)  
PMIX\_INT32, [51](#)  
PMIX\_INT64, [51](#)  
PMIX\_INT8, [51](#)  
PMIX\_INTERNAL, [96](#)  
PMIX\_IOF\_CHANNEL, [52](#)  
PMIX\_JCTRL\_CHECKPOINT, [215](#)  
PMIX\_JCTRL\_CHECKPOINT\_COMPLETE, [215](#)  
PMIX\_JCTRL\_PREEMPT\_ALERT, [215](#)  
PMIX\_JOB\_STATE, [52](#)  
PMIX\_JOB\_STATE\_AWAITING\_ALLOC, [29](#)  
PMIX\_JOB\_STATE\_CONNECTED, [29](#)  
PMIX\_JOB\_STATE\_LAUNCH\_UNDERWAY, [29](#)  
PMIX\_JOB\_STATE\_RUNNING, [29](#)  
PMIX\_JOB\_STATE\_SUSPENDED, [29](#)  
PMIX\_JOB\_STATE\_TERMINATED, [29](#)  
PMIX\_JOB\_STATE\_TERMINATED\_WITH\_ERROR, [29](#)  
PMIX\_JOB\_STATE\_UNDEF, [29](#)  
PMIX\_JOB\_STATE\_UNTERMINATED, [29](#)  
PMIX\_KVAL, [52](#)  
PMIX\_LAUNCH\_COMPLETE, [438](#)  
PMIX\_LAUNCHER\_READY, [425](#)  
PMIX\_LINK\_DOWN, [268](#)  
PMIX\_LINK\_STATE, [52](#)  
PMIX\_LINK\_STATE\_UNKNOWN, [268](#)  
PMIX\_LINK\_UP, [268](#)  
PMIX\_LOCAL, [96](#)

PMIX\_LOCALITY\_NONLOCAL, [193](#)  
PMIX\_LOCALITY\_SHARE\_CORE, [193](#)  
PMIX\_LOCALITY\_SHARE\_HWTHREAD, [193](#)  
PMIX\_LOCALITY\_SHARE\_L1CACHE, [193](#)  
PMIX\_LOCALITY\_SHARE\_L2CACHE, [193](#)  
PMIX\_LOCALITY\_SHARE\_L3CACHE, [193](#)  
PMIX\_LOCALITY\_SHARE\_NODE, [193](#)  
PMIX\_LOCALITY\_SHARE\_NUMA, [193](#)  
PMIX\_LOCALITY\_SHARE\_PACKAGE, [193](#)  
PMIX\_LOCALITY\_UNKNOWN, [193](#)  
PMIX\_LOCTYPE, [53](#)  
PMIX\_MAX\_KEYLEN, [14](#)  
PMIX\_MAX\_NSLEN, [14](#)  
PMIX\_MODEL\_DECLARED, [64](#)  
PMIX\_MODEL\_RESOURCES, [64](#)  
PMIX\_MONITOR\_FILE\_ALERT, [222](#)  
PMIX\_MONITOR\_HEARTBEAT\_ALERT, [222](#)  
PMIX\_OPENMP\_PARALLEL\_ENTERED, [64](#)  
PMIX\_OPENMP\_PARALLEL\_EXITED, [64](#)  
PMIX\_OPERATION\_IN\_PROGRESS, [16](#)  
PMIX\_OPERATION\_SUCCEEDED, [16](#)  
PMIX\_PDATA, [52](#)  
PMIX\_PERSIST, [52](#)  
PMIX\_PERSIST\_APP, [123](#)  
PMIX\_PERSIST\_FIRST\_READ, [123](#)  
PMIX\_PERSIST\_INDEF, [123](#)  
PMIX\_PERSIST\_INVALID, [123](#)  
PMIX\_PERSIST\_PROC, [123](#)  
PMIX\_PERSIST\_SESSION, [123](#)  
PMIX\_PID, [51](#)  
PMIX\_POINTER, [52](#)  
PMIX\_PROC, [52](#)  
PMIX\_PROC\_CPUSET, [52](#)  
PMIX\_PROC\_INFO, [52](#)  
PMIX\_PROC\_NAMESPACE, [52](#)  
PMIX\_PROC\_RANK, [52](#)  
PMIX\_PROC\_STATE, [52](#)  
PMIX\_PROC\_STATE\_ABORTED, [26](#)  
PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG, [26](#)  
PMIX\_PROC\_STATE\_CALLED\_ABORT, [26](#)  
PMIX\_PROC\_STATE\_CANNOT\_RESTART, [26](#)  
PMIX\_PROC\_STATE\_COMM\_FAILED, [26](#)  
PMIX\_PROC\_STATE\_CONNECTED, [26](#)

PMIX\_PROC\_STATE\_ERROR, [26](#)  
PMIX\_PROC\_STATE\_FAILED\_TO\_LAUNCH, [26](#)  
PMIX\_PROC\_STATE\_FAILED\_TO\_START, [26](#)  
PMIX\_PROC\_STATE\_HEARTBEAT\_FAILED, [26](#)  
PMIX\_PROC\_STATE\_KILLED\_BY\_CMD, [26](#)  
PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY, [26](#)  
PMIX\_PROC\_STATE\_MIGRATING, [26](#)  
PMIX\_PROC\_STATE\_PREPPED, [26](#)  
PMIX\_PROC\_STATE\_RESTART, [26](#)  
PMIX\_PROC\_STATE\_RUNNING, [26](#)  
PMIX\_PROC\_STATE\_SENSOR\_BOUND\_EXCEEDED, [26](#)  
PMIX\_PROC\_STATE\_TERM\_NON\_ZERO, [26](#)  
PMIX\_PROC\_STATE\_TERM\_WO\_SYNC, [26](#)  
PMIX\_PROC\_STATE\_TERMINATE, [26](#)  
PMIX\_PROC\_STATE\_TERMINATED, [26](#)  
PMIX\_PROC\_STATE\_UNDEF, [26](#)  
PMIX\_PROC\_STATE\_UNTERMINATED, [26](#)  
PMIX\_PROCESS\_SET\_DEFINE, [232](#)  
PMIX\_PROCESS\_SET\_DELETE, [232](#)  
PMIX\_QUERY, [52](#)  
PMIX\_RANGE\_CUSTOM, [123](#)  
PMIX\_RANGE\_GLOBAL, [123](#)  
PMIX\_RANGE\_INVALID, [123](#)  
PMIX\_RANGE\_LOCAL, [123](#)  
PMIX\_RANGE\_NAMESPACE, [123](#)  
PMIX\_RANGE\_PROC\_LOCAL, [123](#)  
PMIX\_RANGE\_RM, [123](#)  
PMIX\_RANGE\_SESSION, [123](#)  
PMIX\_RANGE\_UNDEF, [123](#)  
PMIX\_RANK\_INVALID, [21](#)  
PMIX\_RANK\_LOCAL\_NODE, [20](#)  
PMIX\_RANK\_LOCAL\_PEERS, [21](#)  
PMIX\_RANK\_UNDEF, [20](#)  
PMIX\_RANK\_VALID, [21](#)  
PMIX\_RANK\_WILDCARD, [20](#)  
PMIX\_REGATTR, [52](#)  
PMIX\_REGEX, [52](#)  
PMIX\_REMOTE, [96](#)  
PMIX\_SCOPE, [52](#)  
PMIX\_SCOPE\_UNDEF, [96](#)  
PMIX\_SIZE, [51](#)  
PMIX\_STATUS, [52](#)  
PMIX\_STORAGE\_ACCESS\_RD, [457](#)

PMIX\_STORAGE\_ACCESS\_RDWR, [457](#)  
PMIX\_STORAGE\_ACCESS\_WR, [457](#)  
PMIX\_STORAGE\_ACCESSIBILITY\_CLUSTER, [456](#)  
PMIX\_STORAGE\_ACCESSIBILITY\_JOB, [456](#)  
PMIX\_STORAGE\_ACCESSIBILITY\_NODE, [456](#)  
PMIX\_STORAGE\_ACCESSIBILITY\_RACK, [456](#)  
PMIX\_STORAGE\_ACCESSIBILITY\_REMOTE, [456](#)  
PMIX\_STORAGE\_ACCESSIBILITY\_SESSION, [456](#)  
PMIX\_STORAGE\_MEDIUM\_HDD, [455](#)  
PMIX\_STORAGE\_MEDIUM\_NVME, [455](#)  
PMIX\_STORAGE\_MEDIUM\_PMEM, [455](#)  
PMIX\_STORAGE\_MEDIUM\_RAM, [455](#)  
PMIX\_STORAGE\_MEDIUM\_SSD, [455](#)  
PMIX\_STORAGE\_MEDIUM\_TAPE, [455](#)  
PMIX\_STORAGE\_MEDIUM\_UNKNOWN, [455](#)  
PMIX\_STORAGE\_PERSISTENCE\_ARCHIVE, [457](#)  
PMIX\_STORAGE\_PERSISTENCE\_JOB, [456](#)  
PMIX\_STORAGE\_PERSISTENCE\_NODE, [456](#)  
PMIX\_STORAGE\_PERSISTENCE\_PROJECT, [456](#)  
PMIX\_STORAGE\_PERSISTENCE\_SCRATCH, [456](#)  
PMIX\_STORAGE\_PERSISTENCE\_SESSION, [456](#)  
PMIX\_STORAGE\_PERSISTENCE\_TEMPORARY, [456](#)  
PMIX\_STRING, [51](#)  
PMIX\_SUCCESS, [15](#)  
PMIX\_TIME, [52](#)  
PMIX\_TIMEVAL, [52](#)  
PMIX\_TOPO, [53](#)  
PMIX\_UINT, [51](#)  
PMIX\_UINT16, [51](#)  
PMIX\_UINT32, [51](#)  
PMIX\_UINT64, [51](#)  
PMIX\_UINT8, [51](#)  
PMIX\_UNDEF, [51](#)  
PMIX\_VALUE, [52](#)

*PMIX\_CONNECT\_REQUESTED*

**Deprecated, [589](#)**

*PMIX\_ERR\_DATA\_VALUE\_NOT\_FOUND*

**Deprecated, [568](#)**

**Removed, [590](#)**

*PMIX\_ERR\_DEBUGGER\_RELEASE*

**Deprecated, [589](#)**

*PMIX\_ERR\_HANDSHAKE\_FAILED*

**Deprecated, [568](#)**

**Removed, [589](#)**  
*PMIX\_ERR\_IN\_ERRNO*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_ARG*  
**Deprecated, [568](#)**  
**Removed, [590](#)**  
*PMIX\_ERR\_INVALID\_ARGS*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_KEY*  
**Deprecated, [568](#)**  
**Removed, [590](#)**  
*PMIX\_ERR\_INVALID\_KEY\_LENGTH*  
**Deprecated, [568](#)**  
**Removed, [590](#)**  
*PMIX\_ERR\_INVALID\_KEYVALP*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_LENGTH*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_NAMESPACE*  
**Deprecated, [568](#)**  
**Removed, [590](#)**  
*PMIX\_ERR\_INVALID\_NUM\_ARGS*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_NUM\_PARSED*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_SIZE*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_INVALID\_TERMINATION*  
**Deprecated, [589](#)**  
*PMIX\_ERR\_INVALID\_VAL*  
**Deprecated, [568](#)**  
**Removed, [590](#)**  
*PMIX\_ERR\_INVALID\_VAL\_LENGTH*  
**Deprecated, [568](#)**  
**Removed, [589](#)**  
*PMIX\_ERR\_JOB\_TERMINATED*

**Deprecated, 589**  
*PMIX\_ERR\_LOST\_CONNECTION\_TO\_CLIENT*  
**Deprecated, 589**  
*PMIX\_ERR\_LOST\_CONNECTION\_TO\_SERVER*  
**Deprecated, 589**  
*PMIX\_ERR\_LOST\_PEER\_CONNECTION*  
**Deprecated, 589**  
*PMIX\_ERR\_NODE\_DOWN*  
**Deprecated, 589**  
*PMIX\_ERR\_NODE\_OFFLINE*  
**Deprecated, 589**  
*PMIX\_ERR\_NOT\_IMPLEMENTED*  
**Deprecated, 568**  
**Removed, 590**  
*PMIX\_ERR\_PACK\_MISMATCH*  
**Deprecated, 568**  
**Removed, 590**  
*PMIX\_ERR\_PROC\_ABORTED*  
**Deprecated, 589**  
*PMIX\_ERR\_PROC\_ABORTING*  
**Deprecated, 589**  
*PMIX\_ERR\_PROC\_ENTRY\_NOT\_FOUND*  
**Deprecated, 568**  
**Removed, 589**  
*PMIX\_ERR\_PROC\_REQUESTED\_ABORT*  
**Deprecated, 568**  
**Removed, 589**  
*PMIX\_ERR\_READY\_FOR\_HANDSHAKE*  
**Deprecated, 568**  
**Removed, 589**  
*PMIX\_ERR\_SERVER\_FAILED\_REQUEST*  
**Deprecated, 568**  
**Removed, 589**  
*PMIX\_ERR\_SERVER\_NOT\_AVAIL*  
**Deprecated, 568**  
**Removed, 590**  
*PMIX\_ERR\_SILENT*  
**Deprecated, 568**  
**Removed, 590**  
*PMIX\_ERR\_SYS\_OTHER*  
**Deprecated, 589**  
*PMIX\_EXISTS*  
**Deprecated, 589**

*PMIX\_GDS\_ACTION\_COMPLETE*

**Deprecated, [568](#)**

**Removed, [590](#)**

*PMIX\_INFO\_ARRAY*

**Deprecated, [563](#)**

*PMIX\_MODEX*

**Deprecated, [563](#)**

*PMIX\_NOTIFY\_ALLOC\_COMPLETE*

**Deprecated, [568](#)**

**Removed, [590](#)**

*PMIX\_PROC\_HAS\_CONNECTED*

**Deprecated, [589](#)**

*PMIX\_PROC\_TERMINATED*

**Deprecated, [589](#)**

# Index of Environmental Variables

---

PMIX\_KEEPALIVE\_PIPE, [414](#), [423](#), [424](#), [588](#)

PMIX\_LAUNCHER\_RNDZ\_FILE, [410](#), [414](#), [588](#)

PMIX\_LAUNCHER\_RNDZ\_URI, [414](#), [423](#), [424](#), [544](#), [588](#)

Unofficial Draft



# Index of Attributes

---

PMIX\_ACCESS\_GRPIDS, [123](#), [579](#)  
PMIX\_ACCESS\_PERMISSIONS, [120](#), [122](#), [122](#), [579](#)  
PMIX\_ACCESS\_USERIDS, [123](#), [579](#)  
PMIX\_ADD\_ENVAR, [167](#), [172](#), [178](#), [550](#)  
PMIX\_ADD\_HOST, [165](#), [170](#), [175](#), [364](#)  
PMIX\_ADD\_HOSTFILE, [165](#), [170](#), [175](#), [364](#)  
PMIX\_ALL\_CLONES\_PARTICIPATE, [69](#), [72](#), [72](#), [183](#), [186](#), [187](#), [189](#), [575](#)  
PMIX\_ALLOC\_BANDWIDTH, [168](#), [173](#), [203](#), [206](#), [208](#), [208](#), [326](#), [387](#)  
PMIX\_ALLOC\_CPU\_LIST, [167](#), [173](#), [203](#), [206](#), [208](#), [386](#)  
PMIX\_ALLOC\_FABRIC, [203](#), [206](#), [208](#), [325](#), [386](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_ENDPTS, [168](#), [173](#), [203](#), [204](#), [206](#), [207](#), [208](#), [208](#), [325](#), [326](#), [386](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE, [168](#), [173](#), [204](#), [207](#), [208](#), [326](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_ID, [203](#), [206](#), [208](#), [208](#), [325](#), [386](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_PLANE, [168](#), [173](#), [203](#), [206](#), [207](#), [208](#), [208](#), [326](#), [387](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_QOS, [168](#), [173](#), [203](#), [206](#), [207](#), [208](#), [208](#), [326](#), [387](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_SEC\_KEY, [203](#), [204](#), [206](#), [207](#), [208](#), [208](#), [326](#), [387](#), [590](#)  
PMIX\_ALLOC\_FABRIC\_TYPE, [168](#), [173](#), [203](#), [206](#), [207](#), [208](#), [208](#), [325](#), [326](#), [386](#), [387](#), [590](#)  
PMIX\_ALLOC\_ID, [204](#), [207](#), [386](#), [567](#), [568](#), [579](#)  
PMIX\_ALLOC\_MEM\_SIZE, [167](#), [173](#), [203](#), [206](#), [208](#), [386](#)  
PMIX\_ALLOC\_NODE\_LIST, [167](#), [173](#), [203](#), [206](#), [207](#), [386](#)  
PMIX\_ALLOC\_NUM\_CPU\_LIST, [167](#), [173](#), [203](#), [206](#), [208](#), [386](#)  
PMIX\_ALLOC\_NUM\_CPUS, [167](#), [173](#), [202](#), [206](#), [207](#), [386](#)  
PMIX\_ALLOC\_NUM\_NODES, [167](#), [173](#), [202](#), [205](#), [207](#), [386](#)  
PMIX\_ALLOC\_QUEUE, [104](#), [109](#), [111](#), [167](#), [173](#), [207](#), [378](#), [579](#)  
PMIX\_ALLOC\_REQ\_ID, [202](#), [205](#), [207](#), [567](#)  
PMIX\_ALLOC\_TIME, [167](#), [173](#), [202](#), [206](#), [208](#), [386](#)  
PMIX\_ALLOCATED\_NODELIST, [82](#), [299](#)  
PMIX\_ANL\_MAP, [83](#), [84](#), [300](#)  
PMIX\_APP\_ARGV, [85](#), [301](#), [579](#)  
PMIX\_APP\_INFO, [74](#), [77](#), [80](#), [85](#), [88](#), [102](#), [107](#), [301](#), [302](#), [532](#)  
PMIX\_APP\_INFO\_ARRAY, [298](#), [301](#), [307](#), [307](#), [313](#), [578](#)  
PMIX\_APP\_MAP\_REGEX, [86](#), [301](#)  
PMIX\_APP\_MAP\_TYPE, [86](#), [301](#)  
PMIX\_APP\_RANK, [86](#), [304](#)  
PMIX\_APP\_SIZE, [85](#), [301](#), [312](#)  
PMIX\_APPEND\_ENVAR, [167](#), [172](#), [178](#), [550](#)  
PMIX\_APPLDR, [85](#), [301](#), [313](#)  
PMIX\_APPNUM, [74](#), [77](#), [81](#), [85](#), [86](#), [88](#), [102](#), [107](#), [298](#), [301](#)–[303](#), [307](#), [313](#), [532](#), [578](#)

PMIX\_ATTR\_UNDEF, [5](#)  
PMIX\_AVAIL\_PHYS\_MEMORY, [88](#), 112, 303  
PMIX\_BINDTO, [165](#), [171](#), [175](#), 300, 364  
PMIX\_CLEANUP\_EMPTY, [211](#), [214](#), [217](#)  
PMIX\_CLEANUP\_IGNORE, [211](#), [214](#), [217](#)  
PMIX\_CLEANUP\_LEAVE\_TOPDIR, [211](#), [214](#), [217](#)  
PMIX\_CLEANUP\_RECURSIVE, [211](#), [214](#), [217](#)  
PMIX\_CLIENT\_ATTRIBUTES, [103](#), [108](#), [113](#), [117](#), [432](#), [570](#), [577](#)  
PMIX\_CLIENT\_AVG\_MEMORY, [105](#), [110](#), [112](#)  
PMIX\_CLIENT\_FUNCTIONS, [103](#), [104](#), [108](#), [111](#), [113](#), [117](#), [576](#), [577](#)  
PMIX\_CLUSTER\_ID, [81](#), [299](#)  
PMIX\_CMD\_LINE, [84](#), [579](#)  
PMIX\_COLLECT\_DATA, [69](#), [71](#), [72](#), [94](#), [351](#)  
PMIX\_COLLECT\_GENERATED\_JOB\_INFO, [69](#), [71](#), [72](#), [72](#), [260](#), [351](#), [575](#)  
PMIX\_COLLECTIVE\_ALGO, [564](#)  
PMIX\_CONNECT\_MAX\_RETRIES, [414](#), [443](#)  
PMIX\_CONNECT\_RETRY\_DELAY, [415](#), [443](#)  
PMIX\_CONNECT\_SYSTEM\_FIRST, [412](#), [414](#), [442](#), [446](#)  
PMIX\_CONNECT\_TO\_SYSTEM, [412](#), [414](#), [442](#), [446](#)  
PMIX\_COSPAWN\_APP, [168](#), [173](#), [440](#), [538](#), [542](#)  
PMIX\_CPU\_LIST, [166](#), [172](#), [176](#), [366](#)  
PMIX\_CPUS\_PER\_PROC, [166](#), [172](#), [176](#), [366](#)  
PMIX\_CPUSSET, [87](#), [194](#), [305](#), [339](#)  
PMIX\_CPUSSET\_BITMAP, [87](#), [305](#), [580](#)  
PMIX\_CRED\_TYPE, [284](#), [394](#)  
PMIX\_CREDENTIAL, [87](#), [380](#)  
PMIX\_CRYPT\_KEY, [284](#), [301](#)  
PMIX\_DAEMON\_MEMORY, [105](#), [110](#), [112](#)  
PMIX\_DATA\_SCOPE, [74](#), [77](#), [78](#)  
PMIX\_DEBUG\_DAEMONS\_PER\_NODE, [365](#), [434](#), [435](#), [440](#), [441](#), [537](#), [539](#), [543](#), [544](#), [547](#), [582](#)  
PMIX\_DEBUG\_DAEMONS\_PER\_PROC, [365](#), [434](#), [435](#), [440](#), [441](#), [537](#), [539](#), [543](#), [544](#), [547](#), [581](#)  
PMIX\_DEBUG\_STOP\_IN\_INIT, [418](#), [422](#), [424](#), [432](#), [435](#), [440](#), [537](#), [543](#)  
PMIX\_DEBUG\_STOP\_ON\_EXEC, [418](#), [424](#), [431](#), [433](#), [439](#), [537](#), [543](#)  
PMIX\_DEBUG\_TARGET, [365](#), [433–435](#), [440](#), [440](#), [441](#), [537–539](#), [543](#), [544](#), [547](#), [581](#), [582](#), [590](#)  
PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY, [419](#), [432](#), [439](#), [440](#), [539](#)  
PMIX\_DEBUGGER\_DAEMONS, [365](#), [434](#), [435](#), [440](#), [534](#), [538](#), [544](#), [547](#)  
PMIX\_DEVICE\_DISTANCES, [200](#), [275](#), [305](#), [585](#)  
PMIX\_DEVICE\_ID, [200](#), [260](#), [261](#), [271](#), [274](#), [275](#), [319](#), [583–585](#)  
PMIX\_DEVICE\_TYPE, [200](#), [585](#)  
PMIX\_DISPLAY\_MAP, [165](#), [171](#), [175](#), [364](#)  
PMIX\_EMBED\_BARRIER, [66](#), [66](#)  
PMIX\_ENUM\_VALUE, [331](#), [331](#), [570](#), [578](#)  
PMIX\_EVENT\_ACTION\_TIMEOUT, [142](#), [147](#)

PMIX\_EVENT\_AFFECTED\_PROC, 140, [142](#), 147, 438, 538  
PMIX\_EVENT\_AFFECTED\_PROCS, 140, [142](#), 147, 438  
PMIX\_EVENT\_BASE, [63](#), 292, 444  
PMIX\_EVENT\_CUSTOM\_RANGE, 139, [142](#), 147  
PMIX\_EVENT\_DO\_NOT\_CACHE, [142](#), 147  
PMIX\_EVENT\_HDLR\_AFTER, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_APPEND, 139, [142](#), 558  
PMIX\_EVENT\_HDLR\_BEFORE, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_FIRST, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_LAST, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_NAME, 139, [141](#), 558  
PMIX\_EVENT\_HDLR\_PREPEND, 139, [141](#)  
PMIX\_EVENT\_NON\_DEFAULT, [142](#), 147  
PMIX\_EVENT\_PROXY, [142](#), 147  
PMIX\_EVENT\_RETURN\_OBJECT, 139, [142](#)  
PMIX\_EVENT\_SILENT\_TERMINATION, 168, 174, [178](#)  
PMIX\_EVENT\_TERMINATE\_JOB, [142](#), 147  
PMIX\_EVENT\_TERMINATE\_NODE, [142](#), 147  
PMIX\_EVENT\_TERMINATE\_PROC, [142](#), 147  
PMIX\_EVENT\_TERMINATE\_SESSION, [142](#), 147  
PMIX\_EVENT\_TEXT\_MESSAGE, [142](#), 147  
PMIX\_EVENT\_TIMESTAMP, [142](#), 177, 178, 418, 419, 437, 438, 538, 544, 548, 587, 588  
PMIX\_EXEC\_AGENT, 422, [425](#), 436, 581  
PMIX\_EXIT\_CODE, [86](#), 177, 178, 418, 419, 437, 538, 544, 548, 588  
PMIX\_EXTERNAL\_PROGRESS, 62, 292, [294](#), 444, 580  
PMIX\_FABRIC\_COORDINATES, [273](#), 582  
PMIX\_FABRIC\_COST\_MATRIX, 270, [272](#), 582  
PMIX\_FABRIC\_DEVICE, 260, 270, 273, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_ADDRESS, 261, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_BUS\_TYPE, 261, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_COORDINATES, 261, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_DRIVER, 261, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_FIRMWARE, 261, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_INDEX, 261, [274](#), 407, 584  
PMIX\_FABRIC\_DEVICE\_MTU, 262, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_NAME, 260, 261, 270, [274](#), 319, 584  
PMIX\_FABRIC\_DEVICE\_PCI\_DEVID, 261, 271, [274](#), 275, 585  
PMIX\_FABRIC\_DEVICE\_SPEED, 262, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_STATE, 262, 271, [274](#), 584  
PMIX\_FABRIC\_DEVICE\_TYPE, 262, 271, [274](#), 585  
PMIX\_FABRIC\_DEVICE\_VENDOR, 261, 271, [274](#), 584

PMIX\_FABRIC\_DEVICE\_VENDORID, 261, [274](#), 584  
PMIX\_FABRIC\_DEVICES, 260, [273](#)  
PMIX\_FABRIC\_DIMS, 270, [273](#), 583  
PMIX\_FABRIC\_ENDPT, [275](#), 583  
PMIX\_FABRIC\_GROUPS, 270, [272](#), 582  
PMIX\_FABRIC\_IDENTIFIER, 269, [273](#), 276, 407, 582  
PMIX\_FABRIC\_INDEX, 268, [273](#), 273, 582  
PMIX\_FABRIC\_NUM\_DEVICES, 269, [273](#), 582  
PMIX\_FABRIC\_PLANE, 270, [272](#), 273, 276, 277, 407, 583  
PMIX\_FABRIC\_SHAPE, 270, [273](#), 583  
PMIX\_FABRIC\_SHAPE\_STRING, 270, [273](#), 583  
PMIX\_FABRIC\_SWITCH, [272](#), 275, 583  
PMIX\_FABRIC\_VENDOR, 269, [273](#), 276, 407, 582  
PMIX\_FIRST\_ENVAR, 167, 172, [178](#), 588  
PMIX\_FORKEXEC\_AGENT, 422, 424, [425](#), 436, 581  
PMIX\_FWD\_STDDIAG, 417, 421, [425](#), 567  
PMIX\_FWD\_STDERR, 365, 381, 416, 421, [425](#), 426, 538, 542, 548  
PMIX\_FWD\_STDIN, 364, 381, 416, 420, [424](#), 426  
PMIX\_FWD\_STDOUT, 365, 381, 416, 421, [424](#), 424, 426, 538, 542, 548  
PMIX\_GET\_POINTER\_VALUES, 74, 75, 77, [78](#), 575  
PMIX\_GET\_REFRESH\_CACHE, 74, 77, [78](#), 98, 575  
PMIX\_GET\_STATIC\_VALUES, 73–75, [78](#), 78, 575  
PMIX\_GLOBAL\_RANK, [86](#), 304, 533  
PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID, [237](#), 239, 243, 248, 251, 405, 406, 586  
PMIX\_GROUP\_CONTEXT\_ID, [237](#), 405, 586  
PMIX\_GROUP\_ENDPT\_DATA, [237](#), 405, 587  
PMIX\_GROUP\_FT\_COLLECTIVE, [237](#), 239, 243, 248, 251, 586  
PMIX\_GROUP\_ID, [236](#), 236, 405, 560, 586  
PMIX\_GROUP\_LEADER, [237](#), 239, 240, 243, 249, 254, 586  
PMIX\_GROUP\_LOCAL\_ONLY, [237](#), 239, 243, 405, 586  
PMIX\_GROUP\_MEMBERSHIP, [237](#), 240, 405  
PMIX\_GROUP\_NAMES, [237](#), 587  
PMIX\_GROUP\_NOTIFY\_TERMINATION, [237](#), 239, 240, 243, 245, 248, 251, 586  
PMIX\_GROUP\_OPTIONAL, [237](#), 239, 240, 243, 248, 251, 405, 586  
PMIX\_GRPID, 120, 121, 124, 127, 132, 134, 202, 205, 210, 213, 218, 220, 224, 227, 282, 284,  
286, 287, 356, 357, 359, 361, 363, 372, 378, 380, [382](#), 383, 386, 389, 392, 394, 397, 398,  
400, 403  
PMIX\_HOMOGENEOUS\_SYSTEM, 292, [294](#), 578  
PMIX\_HOST, 164, 170, [175](#), 363  
PMIX\_HOST\_ATTRIBUTES, 103, 108, [113](#), 118, 433, 570, 577  
PMIX\_HOST\_FUNCTIONS, 103, 104, 108, 111, [113](#), 118, 576, 577  
PMIX\_HOSTFILE, 165, 170, [175](#), 363  
PMIX\_HOSTNAME, 74, 77, 81, [88](#), 88, 102, 104, 105, 107, 109–112, 260, 261, 271, 275, 298,

302–305, 307, 319, 378, 433, 441, 532, 533, 539, 544, 578, 585  
PMIX\_HOSTNAME\_ALIASES, [88](#), 302, 580  
PMIX\_HOSTNAME\_KEEP\_FQDN, [82](#), 300, 580  
PMIX\_IMMEDIATE, [74](#), [76](#), [78](#), 89, 98  
PMIX\_INDEX\_ARGV, [166](#), [171](#), [176](#), 365  
PMIX\_IOF\_BUFFERING\_SIZE, 400, 417, 421, [430](#), 450, 453  
PMIX\_IOF\_BUFFERING\_TIME, 400, 417, 421, [430](#), 450, 453  
PMIX\_IOF\_CACHE\_SIZE, 400, 417, 421, [430](#), 450, 453  
PMIX\_IOF\_COMPLETE, [401](#), [429](#), [430](#), 431, 454, 467, 581  
PMIX\_IOF\_COPY, [427](#), [431](#), 581  
PMIX\_IOF\_DROP\_NEWEST, 400, 417, 421, [430](#), 450, 453  
PMIX\_IOF\_DROP\_OLDEST, 400, 417, 421, [430](#), 450, 453  
PMIX\_IOF\_PUSH\_STDIN, [428](#), [430](#), 453, 581  
PMIX\_IOF\_REDIRECT, [427](#), [431](#), 581  
PMIX\_IOF\_TAG\_OUTPUT, [417](#), [421](#), [427](#), [430](#), 450  
PMIX\_IOF\_TIMESTAMP\_OUTPUT, [417](#), [422](#), [428](#), [430](#), 450  
PMIX\_IOF\_XML\_OUTPUT, [417](#), [422](#), [428](#), [430](#), 451  
PMIX\_JOB\_CONTINUOUS, [166](#), [172](#), [177](#), 366  
PMIX\_JOB\_CTRL\_CANCEL, [211](#), [214](#), [216](#), 389  
PMIX\_JOB\_CTRL\_CHECKPOINT, [211](#), [214](#), [216](#), 389  
PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT, [211](#), [214](#), [216](#), 390  
PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD, [212](#), [215](#), [216](#), 390  
PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL, [211](#), [214](#), [216](#), 390  
PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT, [212](#), [214](#), [216](#), 390  
PMIX\_JOB\_CTRL\_ID, [210](#), [211](#), [213](#), [214](#), [216](#), 216, 389  
PMIX\_JOB\_CTRL\_KILL, [211](#), [213](#), [216](#), 389  
PMIX\_JOB\_CTRL\_PAUSE, [210](#), [213](#), [216](#), 389  
PMIX\_JOB\_CTRL\_PREEMPTIBLE, [212](#), [215](#), [216](#), 390  
PMIX\_JOB\_CTRL\_PROVISION, [212](#), [215](#), [216](#), 390  
PMIX\_JOB\_CTRL\_PROVISION\_IMAGE, [212](#), [215](#), [216](#), 390  
PMIX\_JOB\_CTRL\_RESTART, [211](#), [214](#), [216](#), 389  
PMIX\_JOB\_CTRL\_RESUME, [210](#), [213](#), [216](#), 389  
PMIX\_JOB\_CTRL\_SIGNAL, [211](#), [214](#), [216](#), 389  
PMIX\_JOB\_CTRL\_TERMINATE, [211](#), [214](#), [216](#), 389  
PMIX\_JOB\_INFO, [74](#), [77](#), [80](#), 83, 102, 107  
PMIX\_JOB\_INFO\_ARRAY, [297](#), [299](#), [307](#), 307, 308, 311, 565, 577  
PMIX\_JOB\_NUM\_APPS, [84](#), 300, 311  
PMIX\_JOB\_RECOVERABLE, [166](#), [172](#), [177](#), 366  
PMIX\_JOB\_SIZE, [84](#), 299, 311, 312, 532, 564, 567  
PMIX\_JOB\_TERM\_STATUS, [177](#), [178](#), [418](#), [419](#), [437](#), [438](#), [439](#), 538, 544, 548, 588  
PMIX\_JOBID, [83](#), 297–299, 307, 311, 438, 578  
PMIX\_LAUNCH\_DIRECTIVES, [424](#), [425](#), 542, 582  
PMIX\_LAUNCHER, [409](#), [414](#), 415

PMIX\_LAUNCHER\_DAEMON, [422](#), [425](#), [580](#)  
PMIX\_LAUNCHER\_RENDEZVOUS\_FILE, [410](#), [415](#), [580](#)  
PMIX\_LOCAL\_CPUSSETS, [88](#), [303](#), [315](#)  
PMIX\_LOCAL\_PEERS, [88](#), [88](#), [303](#), [314](#), [532](#)  
PMIX\_LOCAL\_PROCS, [88](#), [303](#)  
PMIX\_LOCAL\_RANK, [87](#), [304](#), [434–436](#), [441](#), [533](#), [537](#), [538](#), [543](#), [547](#), [581](#), [582](#)  
PMIX\_LOCAL\_SIZE, [88](#), [302](#), [532](#)  
PMIX\_LOCALITY\_STRING, [191](#), [193](#), [304](#), [339](#), [533](#)  
PMIX\_LOCALLDR, [88](#), [302](#)  
PMIX\_LOG\_COMPLETION, [178](#), [418](#), [437](#), [588](#)  
PMIX\_LOG\_EMAIL, [225](#), [228](#), [230](#), [384](#)  
PMIX\_LOG\_EMAIL\_ADDR, [225](#), [228](#), [230](#), [384](#)  
PMIX\_LOG\_EMAIL\_MSG, [225](#), [228](#), [230](#), [384](#)  
PMIX\_LOG\_EMAIL\_SENDER\_ADDR, [225](#), [228](#), [230](#)  
PMIX\_LOG\_EMAIL\_SERVER, [225](#), [228](#), [230](#)  
PMIX\_LOG\_EMAIL\_SRVR\_PORT, [225](#), [228](#), [230](#)  
PMIX\_LOG\_EMAIL\_SUBJECT, [225](#), [228](#), [230](#), [384](#)  
PMIX\_LOG\_GENERATE\_TIMESTAMP, [224](#), [227](#), [229](#)  
PMIX\_LOG\_GLOBAL\_DATASTORE, [225](#), [228](#), [230](#)  
PMIX\_LOG\_GLOBAL\_SYSLOG, [224](#), [227](#), [229](#)  
PMIX\_LOG\_JOB\_EVENTS, [177](#), [418](#), [437](#), [587](#)  
PMIX\_LOG\_JOB\_RECORD, [225](#), [228](#), [230](#)  
PMIX\_LOG\_LOCAL\_SYSLOG, [224](#), [227](#), [229](#)  
PMIX\_LOG\_MSG, [229](#), [384](#)  
PMIX\_LOG\_ONCE, [224](#), [227](#), [229](#)  
PMIX\_LOG\_PROC\_ABNORMAL\_TERMINATION, [177](#), [587](#)  
PMIX\_LOG\_PROC\_TERMINATION, [177](#), [587](#)  
PMIX\_LOG\_SOURCE, [224](#), [227](#), [229](#)  
PMIX\_LOG\_STDERR, [224](#), [227](#), [229](#), [383](#)  
PMIX\_LOG\_STDOUT, [224](#), [227](#), [229](#), [383](#)  
PMIX\_LOG\_SYSLOG, [224](#), [227](#), [229](#), [383](#)  
PMIX\_LOG\_SYSLOG\_PRI, [224](#), [227](#), [229](#)  
PMIX\_LOG\_TAG\_OUTPUT, [225](#), [228](#), [229](#)  
PMIX\_LOG\_TIMESTAMP, [224](#), [227](#), [229](#)  
PMIX\_LOG\_TIMESTAMP\_OUTPUT, [225](#), [228](#), [229](#)  
PMIX\_LOG\_XML\_OUTPUT, [225](#), [228](#), [229](#)  
PMIX\_MAPBY, [165](#), [171](#), [175](#), [175](#), [300](#), [364](#), [538](#), [543](#), [547](#)  
PMIX\_MAX\_PROCS, [82](#), [82–85](#), [298–301](#), [303](#), [330](#), [532](#), [567](#)  
PMIX\_MAX\_RESTARTS, [166](#), [172](#), [177](#), [366](#)  
PMIX\_MAX\_VALUE, [331](#), [331](#), [570](#), [578](#)  
PMIX\_MERGE\_STDERR\_STDOUT, [166](#), [171](#), [176](#), [365](#)  
PMIX\_MIN\_VALUE, [331](#), [331](#), [570](#), [578](#)  
PMIX\_MODEL\_AFFINITY\_POLICY, [63](#), [65](#), [553](#)

PMIX\_MODEL\_CPU\_TYPE, 63, [65](#), 552  
 PMIX\_MODEL\_LIBRARY\_NAME, 63, [65](#), 302, 327, 552  
 PMIX\_MODEL\_LIBRARY\_VERSION, 63, [65](#), 302, 327, 552  
 PMIX\_MODEL\_NUM\_CPUS, 63, [65](#), 552  
 PMIX\_MODEL\_NUM\_THREADS, 63, [65](#), 552  
 PMIX\_MODEL\_PHASE\_NAME, [65](#), 142, 552, 553  
 PMIX\_MODEL\_PHASE\_TYPE, [65](#), 142, 552  
 PMIX\_MONITOR\_APP\_CONTROL, 218, 221, [222](#), 392  
 PMIX\_MONITOR\_CANCEL, 218, 221, [222](#), 392  
 PMIX\_MONITOR\_FILE, 218, 219, 221, [222](#), 392  
 PMIX\_MONITOR\_FILE\_ACCESS, 218, 221, [223](#), 392  
 PMIX\_MONITOR\_FILE\_CHECK\_TIME, 219, 221, [223](#), 393  
 PMIX\_MONITOR\_FILE\_DROPS, 219, 221, [223](#), 393  
 PMIX\_MONITOR\_FILE\_MODIFY, 219, 221, [223](#), 392  
 PMIX\_MONITOR\_FILE\_SIZE, 218, 221, [222](#), 392  
 PMIX\_MONITOR\_HEARTBEAT, 218, 221, [222](#), 392  
 PMIX\_MONITOR\_HEARTBEAT\_DROPS, 218, 221, [222](#), 392  
 PMIX\_MONITOR\_HEARTBEAT\_TIME, 218, 221, [222](#), 392  
 PMIX\_MONITOR\_ID, 218, 220, [222](#), 392  
 PMIX\_NO\_OVERSUBSCRIBE, 166, 172, [176](#), 366  
 PMIX\_NO\_PROCS\_ON\_HEAD, 166, 172, [176](#), 366  
 PMIX\_NODE\_INFO, 74, 77, [81](#), 88, 102, 107, 303  
 PMIX\_NODE\_INFO\_ARRAY, 298, 302, [307](#), 308, 312, 314, 318, 319, 578  
 PMIX\_NODE\_LIST, [82](#), 84, 86, 532  
 PMIX\_NODE\_MAP, [82](#), 84, 85, 300, 311–313, 326, 327, 567  
 PMIX\_NODE\_MAP\_RAW, [83](#), 580  
 PMIX\_NODE\_RANK, [87](#), 304, 435  
 PMIX\_NODE\_SIZE, [88](#), 302  
 PMIX\_NODEID, 74, 77, 81, [88](#), 88, 102, 105, 107, 110, 112, 261, 271, 275, 298, 302–304, 307, 319, 532, 578, 585  
 PMIX\_NOHUP, 417, 422, [425](#), 580  
 PMIX\_NOTIFY\_COMPLETION, [177](#), 418, 437, 538, 544, 548  
 PMIX\_NOTIFY\_JOB\_EVENTS, [177](#), 417, 437, 587  
 PMIX\_NOTIFY\_PROC\_ABNORMAL\_TERMINATION, [177](#), 587  
 PMIX\_NOTIFY\_PROC\_TERMINATION, [177](#), 587  
 PMIX\_NPROC\_OFFSET, [83](#), 300  
 PMIX\_NSDIR, [84](#), 87, 303, 305  
 PMIX\_NSPACE, [83](#), 103–105, 108–111, 113, 297–299, 307, 312, 378, 379, 433, 438, 441, 539, 544, 547, 576–578  
 PMIX\_NUM\_ALLOCATED\_NODES, [82](#), 579  
 PMIX\_NUM\_NODES, 80, [82](#), 84, 85, 311, 312, 532, 579  
 PMIX\_NUM\_SLOTS, [82](#), 83, 85  
 PMIX\_OPTIONAL, 73, 76, [78](#), 98



PMIX\_OUTPUT\_TO\_DIRECTORY, [176](#), [587](#)  
PMIX\_OUTPUT\_TO\_FILE, [166](#), [171](#), [176](#), [365](#)  
PMIX\_PACKAGE\_RANK, [87](#), [305](#), [579](#)  
PMIX\_PARENT\_ID, [86](#), [363](#), [423](#)  
PMIX\_PERSISTENCE, [120](#), [122](#), [122](#), [356](#), [462](#)  
PMIX\_PERSONALITY, [165](#), [170](#), [175](#), [364](#)  
PMIX\_PPR, [165](#), [171](#), [175](#), [364](#)  
PMIX\_PREFIX, [164](#), [170](#), [175](#), [363](#)  
PMIX\_PRELOAD\_BIN, [165](#), [170](#), [175](#), [364](#)  
PMIX\_PRELOAD\_FILES, [165](#), [170](#), [175](#), [364](#)  
PMIX\_PREPEND\_ENVAR, [167](#), [172](#), [178](#), [550](#)  
PMIX\_PRIMARY\_SERVER, [415](#), [447](#), [580](#)  
PMIX\_PROC\_INFO, [81](#), [103](#), [107](#)  
PMIX\_PROC\_INFO\_ARRAY, [298](#), [303](#), [307](#), [313](#), [578](#), [590](#)  
PMIX\_PROC\_MAP, [83](#), [83](#), [84](#), [86](#), [300](#), [311](#), [312](#), [326](#), [327](#), [532](#), [567](#)  
PMIX\_PROC\_MAP\_RAW, [83](#), [580](#)  
PMIX\_PROC\_PID, [87](#), [105](#), [110](#)  
PMIX\_PROC\_STATE\_STATUS, [105](#), [110](#), [439](#)  
PMIX\_PROC\_TERM\_STATUS, [438](#), [439](#)  
PMIX\_PROCDIR, [87](#), [305](#)  
PMIX\_PROCID, [86](#), [103](#), [104](#), [107–109](#), [111](#), [177](#), [178](#), [298](#), [307](#), [379](#), [418](#), [419](#), [437](#), [538](#), [544](#),  
[548](#), [578](#), [588](#)  
PMIX\_PROGRAMMING\_MODEL, [63](#), [65](#), [302](#), [327](#), [552](#)  
PMIX\_PSET\_MEMBERS, [232](#), [233](#), [585](#)  
PMIX\_PSET\_NAME, [232](#), [233](#), [585](#)  
PMIX\_PSET\_NAMES, [231](#), [233](#), [301](#), [560](#), [585](#)  
PMIX\_QUERY\_ALLOC\_STATUS, [105](#), [109](#), [113](#), [379](#)  
PMIX\_QUERY\_ATTRIBUTE\_SUPPORT, [103](#), [104](#), [108](#), [111](#), [117](#), [432](#), [576](#)  
PMIX\_QUERY\_AUTHORIZATIONS, [105](#), [110](#), [111](#)  
PMIX\_QUERY\_AVAIL\_SERVERS, [112](#), [412](#), [576](#)  
PMIX\_QUERY\_DEBUG\_SUPPORT, [104](#), [109](#), [111](#), [378](#), [537](#)  
PMIX\_QUERY\_GROUP\_MEMBERSHIP, [236](#), [560](#), [586](#)  
PMIX\_QUERY\_GROUP\_NAMES, [236](#), [560](#), [586](#)  
PMIX\_QUERY\_JOB\_STATUS, [104](#), [109](#), [111](#), [378](#)  
PMIX\_QUERY\_LOCAL\_ONLY, [112](#), [379](#)  
PMIX\_QUERY\_LOCAL\_PROC\_TABLE, [104](#), [109](#), [111](#), [378](#), [433](#), [441](#), [539](#), [544](#), [545](#)  
PMIX\_QUERY\_MEMORY\_USAGE, [104](#), [109](#), [111](#), [379](#)  
PMIX\_QUERY\_NAMESPACE\_INFO, [110](#), [576](#)  
PMIX\_QUERY\_NAMESPACES, [104](#), [109](#), [110](#), [378](#), [433](#), [548](#)  
PMIX\_QUERY\_NUM\_GROUPS, [236](#), [560](#), [585](#)  
PMIX\_QUERY\_NUM\_PSETS, [111](#), [233](#), [585](#)  
PMIX\_QUERY\_PROC\_TABLE, [104](#), [109](#), [111](#), [378](#), [433](#), [441](#), [544](#), [545](#), [547](#)  
PMIX\_QUERY\_PSET\_MEMBERSHIP, [112](#), [233](#), [585](#)



PMIX\_QUERY\_PSET\_NAMES, [111](#), [233](#), [585](#)  
PMIX\_QUERY\_QUALIFIERS, [105](#), [112](#), [112](#), [576](#)  
PMIX\_QUERY\_QUEUE\_LIST, [104](#), [109](#), [111](#), [378](#)  
PMIX\_QUERY\_QUEUE\_STATUS, [104](#), [109](#), [111](#), [378](#)  
PMIX\_QUERY\_REFRESH\_CACHE, [102](#), [106](#), [107](#), [112](#), [117](#)  
PMIX\_QUERY\_REPORT\_AVG, [105](#), [109](#), [113](#), [379](#)  
PMIX\_QUERY\_REPORT\_MINMAX, [105](#), [109](#), [113](#), [379](#)  
PMIX\_QUERY\_RESULTS, [105](#), [112](#), [576](#)  
PMIX\_QUERY\_SPAWN\_SUPPORT, [104](#), [109](#), [111](#), [378](#), [537](#)  
PMIX\_QUERY\_STORAGE\_LIST, [457](#), [594](#)  
PMIX\_QUERY\_SUPPORTED\_KEYS, [110](#), [576](#)  
PMIX\_QUERY\_SUPPORTED\_QUALIFIERS, [110](#), [576](#)  
PMIX\_RANGE, [120](#), [122](#), [122](#), [125](#), [127](#), [133](#), [134](#), [139](#), [219](#), [236](#), [356](#), [359](#), [361](#), [375](#), [405](#), [406](#),  
[462](#), [560](#), [585](#), [586](#)  
PMIX\_RANK, [86](#), [103](#), [104](#), [108](#), [109](#), [111](#), [168](#), [174](#), [298](#), [303](#), [307](#), [379](#), [436](#), [440](#), [533](#), [538](#), [542](#),  
[578](#)  
PMIX\_RANKBY, [165](#), [171](#), [175](#), [300](#), [364](#)  
PMIX\_REGISTER\_CLEANUP, [211](#), [214](#), [216](#)  
PMIX\_REGISTER\_CLEANUP\_DIR, [211](#), [214](#), [217](#)  
PMIX\_REGISTER\_NODATA, [297](#), [307](#)  
PMIX\_REINCARNATION, [87](#), [304](#), [580](#)  
PMIX\_REPORT\_BINDINGS, [166](#), [172](#), [176](#), [366](#)  
PMIX\_REQUESTOR\_IS\_CLIENT, [363](#), [367](#)  
PMIX\_REQUESTOR\_IS\_TOOL, [363](#), [367](#), [548](#)  
PMIX\_REQUIRED\_KEY, [354](#), [355](#), [579](#)  
PMIX\_RM\_NAME, [82](#), [299](#)  
PMIX\_RM\_VERSION, [82](#), [299](#)  
PMIX\_SEND\_HEARTBEAT, [219](#), [221](#), [222](#)  
PMIX\_SERVER\_ATTRIBUTES, [103](#), [108](#), [113](#), [117](#), [570](#), [577](#)  
PMIX\_SERVER\_ENABLE\_MONITORING, [292](#), [294](#)  
PMIX\_SERVER\_FUNCTIONS, [103](#), [104](#), [108](#), [111](#), [113](#), [117](#), [576](#), [577](#)  
PMIX\_SERVER\_GATEWAY, [290](#), [294](#)  
PMIX\_SERVER\_HOSTNAME, [299](#), [414](#)  
PMIX\_SERVER\_INFO\_ARRAY, [112](#), [113](#), [576](#)  
PMIX\_SERVER\_NAMESPACE, [290](#), [294](#), [299](#), [412](#), [442](#), [446](#)  
PMIX\_SERVER\_PIDINFO, [412](#), [414](#), [442](#), [446](#)  
PMIX\_SERVER\_RANK, [290](#), [294](#), [299](#)  
PMIX\_SERVER\_REMOTE\_CONNECTIONS, [291](#), [293](#)  
PMIX\_SERVER\_SCHEDULER, [272](#), [275](#), [290](#), [294](#), [577](#), [582](#)  
PMIX\_SERVER\_SESSION\_SUPPORT, [290](#), [294](#), [577](#)  
PMIX\_SERVER\_SHARE\_TOPOLOGY, [292](#), [293](#), [577](#)  
PMIX\_SERVER\_START\_TIME, [294](#), [577](#)  
PMIX\_SERVER\_SYSTEM\_SUPPORT, [290](#), [294](#), [409](#)

PMIX\_SERVER\_TMPDIR, 290, 292, [294](#), 409–411  
PMIX\_SERVER\_TOOL\_SUPPORT, 281, 290, 292, [293](#)  
PMIX\_SERVER\_URI, 105, 110, 411, 412, [414](#), 442, 446  
PMIX\_SESSION\_ID, [81](#), 81, 83, 90, 297, 298, 307, 311, 438, 578  
PMIX\_SESSION\_INFO, 74, 77, [80](#), 82, 90, 102, 107, 298, 300, 327  
PMIX\_SESSION\_INFO\_ARRAY, 297, 298, [307](#), 308, 311, 565  
PMIX\_SET\_ENVAR, 167, 172, [178](#), 550  
PMIX\_SET\_SESSION\_CWD, 164, 170, [176](#), 363  
PMIX\_SETUP\_APP\_ALL, 325, [328](#)  
PMIX\_SETUP\_APP\_ENVARS, 325, [328](#), 538, 543  
PMIX\_SETUP\_APP\_NONENVARS, 325, [328](#)  
PMIX\_SINGLE\_LISTENER, 62, 291, [293](#)  
PMIX\_SOCKET\_MODE, 62, 291, [293](#), 443  
PMIX\_SPAWN\_TOOL, 168, 174, [177](#), 424, 542  
PMIX\_SPAWNED, [87](#), 304, 363  
PMIX\_STDIN\_TGT, 165, 171, [176](#), 364  
PMIX\_STORAGE\_ACCESS\_TYPE, [458](#), 595  
PMIX\_STORAGE\_ACCESSIBILITY, [457](#), 594  
PMIX\_STORAGE\_BW\_CUR, [458](#), 458, 594, 595  
PMIX\_STORAGE\_BW\_MAX, [458](#), 594  
PMIX\_STORAGE\_CAPACITY\_LIMIT, [457](#), 594  
PMIX\_STORAGE\_CAPACITY\_USED, [457](#), 594  
PMIX\_STORAGE\_ID, [457](#), 457, 593, 594  
PMIX\_STORAGE\_IOPS\_CUR, [458](#), 458, 594, 595  
PMIX\_STORAGE\_IOPS\_MAX, [458](#), 594  
PMIX\_STORAGE\_MEDIUM, [457](#), 593  
PMIX\_STORAGE\_MINIMAL\_XFER\_SIZE, [457](#), 594  
PMIX\_STORAGE\_OBJECT\_LIMIT, [457](#), 594  
PMIX\_STORAGE\_OBJECTS\_USED, [457](#), 594  
PMIX\_STORAGE\_PATH, [457](#), 593  
PMIX\_STORAGE\_PERSISTENCE, [457](#), 594  
PMIX\_STORAGE\_SUGGESTED\_XFER\_SIZE, [458](#), 458, 594, 595  
PMIX\_STORAGE\_TYPE, [457](#), 593  
PMIX\_STORAGE\_VERSION, [457](#), 593  
PMIX\_SWITCH\_PEERS, [275](#), 275, 583  
PMIX\_SYSTEM\_TMPDIR, 290, [294](#), 409, 411  
PMIX\_TAG\_OUTPUT, 165, 171, [176](#), 365  
PMIX\_TCP\_DISABLE\_IPV4, 62, [65](#), 291, 443  
PMIX\_TCP\_DISABLE\_IPV6, 62, [65](#), 291, 443  
PMIX\_TCP\_IF\_EXCLUDE, 62, [65](#), 291, 443  
PMIX\_TCP\_IF\_INCLUDE, 62, [65](#), 291, 443  
PMIX\_TCP\_IPV4\_PORT, 62, [65](#), 291, 443  
PMIX\_TCP\_IPV6\_PORT, 62, [65](#), 291, 443

PMIX\_TCP\_REPORT\_URI, [62](#), [64](#), [291](#), [443](#)  
PMIX\_TCP\_URI, [65](#), [411](#), [412](#), [442](#), [446](#)  
PMIX\_TDIR\_RMCLEAN, [82](#), [301](#)  
PMIX\_THREADING\_MODEL, [63](#), [65](#), [552](#)  
PMIX\_TIME\_REMAINING, [100](#), [105](#), [109](#), [111](#), [379](#)  
PMIX\_TIMEOUT, [4](#), [69](#), [72](#), [75](#), [78](#), [79](#), [89](#), [98](#), [99](#), [120](#), [122](#), [125](#), [127](#), [133](#), [134](#), [183](#), [186](#), [187](#),  
[189](#), [235](#), [239](#), [241](#), [244](#), [245](#), [247](#), [248](#), [252](#), [253](#), [255](#), [283](#), [284](#), [286](#), [288](#), [351](#), [355](#), [357](#),  
[359](#), [361](#), [366](#), [368](#), [370](#), [394](#), [397](#), [415](#), [449](#), [544](#), [582](#)  
PMIX\_TIMEOUT\_REPORT\_STATE, [177](#), [587](#)  
PMIX\_TIMEOUT\_STACKTRACES, [177](#), [587](#)  
PMIX\_TIMESTAMP\_OUTPUT, [166](#), [171](#), [176](#), [365](#)  
PMIX\_TMPDIR, [82](#), [84](#), [303](#)  
PMIX\_TOOL\_ATTACHMENT\_FILE, [411](#), [412](#), [415](#), [442](#), [446](#), [580](#)  
PMIX\_TOOL\_ATTRIBUTES, [103](#), [108](#), [113](#), [118](#), [570](#), [577](#)  
PMIX\_TOOL\_CONNECT\_OPTIONAL, [415](#), [580](#)  
PMIX\_TOOL\_DO\_NOT\_CONNECT, [411](#), [415](#), [442](#), [444](#)  
PMIX\_TOOL\_FUNCTIONS, [103](#), [104](#), [108](#), [111](#), [113](#), [118](#), [576](#), [577](#)  
PMIX\_TOOL\_NAMESPACE, [380](#), [410](#), [413](#), [442](#), [444](#)  
PMIX\_TOOL\_RANK, [380](#), [410](#), [414](#), [442](#), [444](#)  
PMIX\_TOPOLOGY2, [292](#), [293](#), [577](#), [590](#)  
PMIX\_UNIV\_SIZE, [9](#), [81](#), [298](#), [311](#), [564](#), [565](#), [567](#)  
PMIX\_UNSET\_ENVAR, [167](#), [172](#), [178](#), [550](#)  
PMIX\_USERID, [120](#), [121](#), [124](#), [127](#), [132](#), [134](#), [202](#), [205](#), [210](#), [213](#), [218](#), [220](#), [224](#), [227](#), [282](#), [284](#),  
[285](#), [287](#), [356–359](#), [361](#), [363](#), [371](#), [378](#), [380](#), [382](#), [383](#), [385](#), [389](#), [392](#), [394](#), [396](#), [398](#), [399](#),  
[403](#)  
PMIX\_USOCK\_DISABLE, [62](#), [291](#), [293](#)  
PMIX\_VERSION\_INFO, [381](#), [382](#)  
PMIX\_WAIT, [77](#), [79](#), [125](#), [127](#), [359](#)  
PMIX\_WAIT\_FOR\_CONNECTION, [415](#), [449](#), [544](#), [582](#)  
PMIX\_WDIR, [164](#), [170](#), [175](#), [301](#), [363](#)

*PMIX\_ALLOC\_NETWORK*

**Deprecated, [590](#)**

*PMIX\_ALLOC\_NETWORK\_ENDPTS*

**Deprecated, [590](#)**

*PMIX\_ALLOC\_NETWORK\_ENDPTS\_NODE*

**Deprecated, [590](#)**

*PMIX\_ALLOC\_NETWORK\_ID*

**Deprecated, [590](#)**

*PMIX\_ALLOC\_NETWORK\_PLANE*

**Deprecated, [590](#)**

*PMIX\_ALLOC\_NETWORK\_QOS*

**Deprecated, [590](#)**

*PMIX\_ALLOC\_NETWORK\_SEC\_KEY*

**Deprecated, [590](#)**  
*PMIX\_ALLOC\_NETWORK\_TYPE*  
**Deprecated, [590](#)**  
*PMIX\_ARCH*  
**Deprecated, [569](#)**  
**Removed, [592](#)**  
*PMIX\_COLLECTIVE\_ALGO*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_COLLECTIVE\_ALGO\_REQD*  
**Deprecated, [566](#)**  
**Removed, [591](#)**  
*PMIX\_DEBUG\_JOB*  
**Deprecated, [590](#)**  
*PMIX\_DSTPATH*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_ERROR\_GROUP\_ABORT*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_COMM*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_GENERAL*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_LOCAL*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_MIGRATE*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_NODE*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_RESOURCE*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_GROUP\_SPAWN*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_ERROR\_HANDLER\_ID*  
**Deprecated, [564](#)**

**Removed, [566](#)**  
*PMIX\_ERROR\_NAME*  
**Deprecated, [564](#)**  
**Removed, [566](#)**  
*PMIX\_HWLOC\_HOLE\_KIND*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_HWLOC\_SHARE\_TOPO*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_HWLOC\_SHMEM\_ADDR*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_HWLOC\_SHMEM\_FILE*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_HWLOC\_SHMEM\_SIZE*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_HWLOC\_XML\_V1*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_HWLOC\_XML\_V2*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_LOCAL\_TOPO*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_LOCALITY*  
**Deprecated, [590](#)**  
*PMIX\_MAP\_BLOB*  
**Deprecated, [569](#)**  
**Removed, [592](#)**  
*PMIX\_MAPPER*  
**Deprecated, [569](#)**  
**Removed, [592](#)**  
*PMIX\_NON\_PMI*  
**Deprecated, [569](#)**  
**Removed, [592](#)**  
*PMIX\_PROC\_BLOB*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_PROC\_DATA*

**Deprecated, [590](#)**  
*PMIX\_PROC\_URI*  
**Deprecated, [569](#)**  
**Removed, [592](#)**  
*PMIX\_RECONNECT\_SERVER*  
**Deprecated, [590](#)**  
*PMIX\_TOPOLOGY*  
**Deprecated, [590](#)**  
*PMIX\_TOPOLOGY\_FILE*  
**Deprecated, [569](#)**  
**Removed, [591](#)**  
*PMIX\_TOPOLOGY\_SIGNATURE*  
**Deprecated, [570](#)**  
**Removed, [591](#)**  
*PMIX\_TOPOLOGY\_XML*  
**Deprecated, [570](#)**  
**Removed, [591](#)**

Unofficial Draft