

Process Management Interface for Exascale (PMIx) Standard

Version 4.1

October 2021

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 4.1.

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

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

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

This page intentionally left blank

Contents

1. Introduction	1
1.1. Background	1
1.2. PMIx Architecture Overview	1
1.3. Portability of Functionality	3
1.3.1. Attributes in PMIx	3
2. PMIx Terms and Conventions	6
2.1. Notational Conventions	8
2.2. Semantics	9
2.3. Naming Conventions	10
2.4. Procedure Conventions	10
3. Data Structures and Types	12
3.1. Constants	13
3.1.1. PMIx Return Status Constants	14
3.1.1.1. User-Defined Error and Event Constants	15
3.2. Data Types	16
3.2.1. Key Structure	16
3.2.1.1. Key support macros	16
3.2.2. Namespace Structure	18
3.2.2.1. Namespace support macros	18
3.2.3. Rank Structure	19
3.2.3.1. Rank support macros	20
3.2.4. Process Structure	20
3.2.4.1. Process structure support macros	21
3.2.5. Process State Structure	24
3.2.6. Process Information Structure	25
3.2.6.1. Process information structure support macros	26
3.2.7. Job State Structure	27

3.2.8.	Value Structure	28
3.2.8.1.	Value structure support macros	29
3.2.9.	Info Structure	33
3.2.9.1.	Info structure support macros	33
3.2.9.2.	Info structure list macros	35
3.2.10.	Info Type Directives	38
3.2.10.1.	Info Directive support macros	39
3.2.11.	Environmental Variable Structure	41
3.2.11.1.	Environmental variable support macros	41
3.2.12.	Byte Object Type	43
3.2.12.1.	Byte object support macros	43
3.2.13.	Data Array Structure	44
3.2.13.1.	Data array support macros	44
3.2.14.	Argument Array Macros	46
3.2.15.	Set Environment Variable	49
3.3.	Generalized Data Types Used for Packing/Unpacking	50
3.4.	General Callback Functions	52
3.4.1.	Release Callback Function	52
3.4.2.	Op Callback Function	53
3.4.3.	Value Callback Function	53
3.4.4.	Info Callback Function	54
3.4.5.	Handler registration callback function	54
3.5.	PMIx Datatype Value String Representations	55
4.	Client Initialization and Finalization	59
4.1.	PMIx_Initialized	59
4.2.	PMIx_Get_version	60
4.3.	PMIx_Init	60
4.3.1.	Initialization events	63
4.3.2.	Initialization attributes	63
4.3.2.1.	Connection attributes	63
4.3.2.2.	Programming model attributes	64
4.4.	PMIx_Finalize	65
4.4.1.	Finalize attributes	65

4.5. PMIx_Progress	65
5. Synchronization and Data Access Operations	67
5.1. PMIx_Fence	67
5.2. PMIx_Fence_nb	69
5.2.1. Fence-related attributes	71
5.3. PMIx_Get	72
5.3.1. PMIx_Get_nb	74
5.3.2. Retrieval attributes	77
5.4. Query	78
5.4.1. PMIxResolve_peers	79
5.4.2. PMIxResolve_nodes	79
5.4.3. PMIxQuery_info	80
5.4.4. PMIxQuery_info_nb	85
5.4.5. Query-specific constants	89
5.4.6. Query attributes	90
5.4.7. Query Structure	92
5.4.7.1. Query structure support macros	93
5.5. Using Get vs Query	95
5.6. Accessing attribute support information	95
6. Reserved Keys	98
6.1. Data realms	98
6.1.1. Session realm attributes	99
6.1.2. Job realm attributes	101
6.1.3. Application realm attributes	103
6.1.4. Process realm attributes	104
6.1.5. Node realm keys	105
6.2. Retrieval rules for reserved keys	107
6.2.1. Accessing information: examples	107
6.2.1.1. Session-level information	108
6.2.1.2. Job-level information	109
6.2.1.3. Application-level information	109
6.2.1.4. Process-level information	110

6.2.1.5. Node-level information	110
7. Process-Related Non-Reserved Keys	112
7.1. Posting Key/Value Pairs	113
7.1.1. PMIx_Put	113
7.1.1.1. Scope of Put Data	114
7.1.2. PMIx_Store_internal	114
7.1.3. PMIx_Commit	115
7.2. Retrieval rules for non-reserved keys	116
8. Publish/Lookup Operations	118
8.1. PMIx_Publish	118
8.2. PMIx_Publish_nb	120
8.3. Publish-specific constants	121
8.4. Publish-specific attributes	121
8.5. Publish-Lookup Datatypes	122
8.5.1. Range of Published Data	122
8.5.2. Data Persistence Structure	122
8.6. PMIx_Lookup	123
8.7. PMIx_Lookup_nb	125
8.7.1. Lookup Returned Data Structure	127
8.7.1.1. Lookup data structure support macros	127
8.7.2. Lookup Callback Function	130
8.8. Retrieval rules for published data	130
8.9. PMIx_Unpublish	131
8.10. PMIx_Unpublish_nb	133
9. Event Notification	135
9.1. Notification and Management	135
9.1.1. Events versus status constants	137
9.1.2. PMIx_Register_event_handler	137
9.1.3. Event registration constants	140
9.1.4. System events	140
9.1.5. Event handler registration and notification attributes	141
9.1.5.1. Fault tolerance event attributes	142

9.1.5.2. Hybrid programming event attributes	142
9.1.6. Notification Function	142
9.1.7. PMIx_Deregister_event_handler	144
9.1.8. PMIx_Notify_event	145
9.1.9. Notification Handler Completion Callback Function	149
9.1.9.1. Completion Callback Function Status Codes	149
10. Data Packing and Unpacking	150
10.1. Data Buffer Type	150
10.2. Support Macros	151
10.3. General Routines	152
10.3.1. PMIx_Data_pack	152
10.3.2. PMIx_Data_unpack	154
10.3.3. PMIx_Data_copy	156
10.3.4. PMIx_Data_print	156
10.3.5. PMIx_Data_copy_payload	157
10.3.6. PMIx_Data_load	158
10.3.7. PMIx_Data_unload	159
10.3.8. PMIx_Data_compress	159
10.3.9. PMIx_Data_decompress	160
11. Process Management	162
11.1. Abort	162
11.1.1. PMIx_Abort	162
11.2. Process Creation	163
11.2.1. PMIx_Spawn	163
11.2.2. PMIx_Spawn_nb	169
11.2.3. Spawn-specific constants	174
11.2.4. Spawn attributes	174
11.2.5. Application Structure	178
11.2.5.1. App structure support macros	178
11.2.5.2. Spawn Callback Function	180
11.3. Connecting and Disconnecting Processes	181
11.3.1. PMIx_Connect	182

11.3.2. PMIx_Connect_nb	184
11.3.3. PMIx_Disconnect	185
11.3.4. PMIx_Disconnect_nb	187
11.4. Process Locality	189
11.4.1. PMIx_Load_topology	189
11.4.2. PMIx_Get_relative_locality	190
11.4.2.1. Topology description	190
11.4.2.2. Topology support macros	191
11.4.2.3. Relative locality of two processes	192
11.4.2.4. Locality keys	192
11.4.3. PMIx_Parse_cpuset_string	192
11.4.4. PMIx_Get_cpuset	193
11.4.4.1. Binding envelope	193
11.4.5. PMIx_Compute_distances	194
11.4.6. PMIx_Compute_distances_nb	195
11.4.7. Device Distance Callback Function	196
11.4.8. Device type	196
11.4.9. Device Distance Structure	197
11.4.10. Device distance support macros	198
11.4.11. Device distance attributes	199
12. Job Management and Reporting	200
12.1. Allocation Requests	200
12.1.1. PMIx_Allocation_request	200
12.1.2. PMIx_Allocation_request_nb	203
12.1.3. Job Allocation attributes	206
12.1.4. Job Allocation Directives	207
12.2. Job Control	208
12.2.1. PMIx_Job_control	208
12.2.2. PMIx_Job_control_nb	211
12.2.3. Job control constants	214
12.2.4. Job control events	214
12.2.5. Job control attributes	215

12.3. Process and Job Monitoring	216
12.3.1. PMIx_Process_monitor	216
12.3.2. PMIx_Process_monitor_nb	218
12.3.3. PMIx_Heartbeat	220
12.3.4. Monitoring events	221
12.3.5. Monitoring attributes	221
12.4. Logging	222
12.4.1. PMIx_Log	222
12.4.2. PMIx_Log_nb	225
12.4.3. Log attributes	228
13. Process Sets and Groups	230
13.1. Process Sets	230
13.1.1. Process Set Constants	231
13.1.2. Process Set Attributes	232
13.2. Process Groups	232
13.2.1. Relation to the host environment	232
13.2.2. Construction procedure	233
13.2.3. Destruct procedure	234
13.2.4. Process Group Events	234
13.2.5. Process Group Attributes	235
13.2.6. PMIx_Group_construct	237
13.2.7. PMIx_Group_construct_nb	240
13.2.8. PMIx_Group_destruct	243
13.2.9. PMIx_Group_destruct_nb	244
13.2.10. PMIx_Group_invite	246
13.2.11. PMIx_Group_invite_nb	249
13.2.12. PMIx_Group_join	252
13.2.13. PMIx_Group_join_nb	254
13.2.13.1. Group accept/decline directives	255
13.2.14. PMIx_Group_leave	255
13.2.15. PMIx_Group_leave_nb	257

14. Fabric Support Definitions	259
14.1. Fabric Support Events	262
14.2. Fabric Support Datatypes	262
14.2.1. Fabric Endpoint Structure	262
14.2.2. Fabric endpoint support macros	263
14.2.3. Fabric Coordinate Structure	264
14.2.4. Fabric coordinate support macros	264
14.2.5. Fabric Geometry Structure	266
14.2.6. Fabric geometry support macros	266
14.2.7. Fabric Coordinate Views	267
14.2.8. Fabric Link State	268
14.2.9. Fabric Operation Constants	268
14.2.10. Fabric registration structure	269
14.2.10.1. Initialize the fabric structure	272
14.3. Fabric Support Attributes	272
14.4. Fabric Support Functions	275
14.4.1. PMIx_Fabric_register	276
14.4.2. PMIx_Fabric_register_nb	277
14.4.3. PMIx_Fabric_update	278
14.4.4. PMIx_Fabric_update_nb	279
14.4.5. PMIx_Fabric_deregister	279
14.4.6. PMIx_Fabric_deregister_nb	280
15. Security	281
15.1. Obtaining Credentials	281
15.1.1. PMIx_Get_credential	282
15.1.2. PMIx_Get_credential_nb	283
15.1.3. Credential Attributes	284
15.2. Validating Credentials	285
15.2.1. PMIx_Validate_credential	285
15.2.2. PMIx_Validate_credential_nb	286

16. Server-Specific Interfaces	289
16.1. Server Initialization and Finalization	289
16.1.1. PMIx_server_init	289
16.1.2. PMIx_server_finalize	293
16.1.3. Server Initialization Attributes	293
16.2. Server Support Functions	294
16.2.1. PMIx_generate_regex	294
16.2.2. PMIx_generate_ppn	296
16.2.3. PMIx_server_register_nspace	296
16.2.3.1. Namespace registration attributes	307
16.2.3.2. Assembling the registration information	308
16.2.4. PMIx_server_deregister_nspace	317
16.2.5. PMIx_server_register_resources	317
16.2.6. PMIx_server_deregister_resources	318
16.2.7. PMIx_server_register_client	319
16.2.8. PMIx_server_deregister_client	321
16.2.9. PMIx_server_setup_fork	321
16.2.10. PMIx_server_dmodex_request	322
16.2.10.1. Server Direct Modex Response Callback Function	323
16.2.11. PMIx_server_setup_application	324
16.2.11.1. Server Setup Application Callback Function	327
16.2.11.2. Server Setup Application Attributes	328
16.2.12. PMIx_Register_attributes	328
16.2.12.1. Attribute registration constants	330
16.2.12.2. Attribute registration structure	330
16.2.12.3. Attribute registration structure descriptive attributes	331
16.2.12.4. Attribute registration structure support macros	331
16.2.13. PMIx_server_setup_local_support	333
16.2.14. PMIx_server_IOF_deliver	335
16.2.15. PMIx_server_collect_inventory	336
16.2.16. PMIx_server_deliver_inventory	337
16.2.17. PMIx_server_generate_locality_string	338

16.2.18. PMIx_server_generate_cpuset_string	339
16.2.18.1. Cpuset Structure	340
16.2.18.2. Cpuset support macros	340
16.2.19. PMIx_server_define_process_set	341
16.2.20. PMIx_server_delete_process_set	342
16.3. Server Function Pointers	342
16.3.1. pmix_server_module_t Module	343
16.3.2. pmix_server_client_connected_fn_t	344
16.3.3. pmix_server_client_connected2_fn_t	345
16.3.4. pmix_server_client_finalized_fn_t	347
16.3.5. pmix_server_abort_fn_t	348
16.3.6. pmix_server_fencenb_fn_t	350
16.3.6.1. Modex Callback Function	353
16.3.7. pmix_server_dmodex_req_fn_t	353
16.3.7.1. Dmodex attributes	355
16.3.8. pmix_server_publish_fn_t	355
16.3.9. pmix_server_lookup_fn_t	357
16.3.10. pmix_server_unpublish_fn_t	360
16.3.11. pmix_server_spawn_fn_t	362
16.3.11.1. Server spawn attributes	367
16.3.12. pmix_server_connect_fn_t	367
16.3.13. pmix_server_disconnect_fn_t	368
16.3.14. pmix_server_register_events_fn_t	370
16.3.15. pmix_server_deregister_events_fn_t	372
16.3.16. pmix_server_notify_event_fn_t	374
16.3.17. pmix_server_listener_fn_t	375
16.3.17.1. PMIx Client Connection Callback Function	376
16.3.18. pmix_server_query_fn_t	377
16.3.19. pmix_server_tool_connection_fn_t	379
16.3.19.1. Tool connection attributes	382
16.3.19.2. PMIx Tool Connection Callback Function	382
16.3.20. pmix_server_log_fn_t	382
16.3.21. pmix_server_alloc_fn_t	384

16.3.22. <code>pmix_server_job_control_fn_t</code>	387
16.3.23. <code>pmix_server_monitor_fn_t</code>	390
16.3.24. <code>pmix_server_get_cred_fn_t</code>	393
16.3.24.1. Credential callback function	394
16.3.25. <code>pmix_server_validate_cred_fn_t</code>	395
16.3.26. Credential validation callback function	397
16.3.27. <code>pmix_server_iof_fn_t</code>	398
16.3.27.1. IOF delivery function	401
16.3.28. <code>pmix_server_stdin_fn_t</code>	402
16.3.29. <code>pmix_server_grp_fn_t</code>	403
16.3.29.1. Group Operation Constants	406
16.3.30. <code>pmix_server_fabric_fn_t</code>	406
17. Tools and Debuggers	408
17.1. Connection Mechanisms	408
17.1.1. Rendezvousing with a local server	411
17.1.2. Connecting to a remote server	412
17.1.3. Attaching to running jobs	413
17.1.4. Tool initialization attributes	413
17.1.5. Tool initialization environmental variables	414
17.1.6. Tool connection attributes	414
17.2. Launching Applications with Tools	415
17.2.1. Direct launch	415
17.2.2. Indirect launch	419
17.2.2.1. Initiator-based command line parsing	420
17.2.2.2. Intermediate Launcher (IL)-based command line parsing	423
17.2.3. Tool spawn-related attributes	424
17.2.4. Tool rendezvous-related events	425
17.3. IO Forwarding	425
17.3.1. Forwarding stdout/stderr	426
17.3.2. Forwarding stdin	428
17.3.3. IO Forwarding Channels	429
17.3.4. IO Forwarding constants	430
17.3.5. IO Forwarding attributes	430

17.4. Debugger Support	431
17.4.1. Co-Location of Debugger Daemons	433
17.4.2. Co-Spawn of Debugger Daemons	435
17.4.3. Debugger Agents	436
17.4.4. Tracking the job lifecycle	437
17.4.4.1. Job lifecycle events	438
17.4.4.2. Job lifecycle attributes	439
17.4.5. Debugger-related constants	439
17.4.6. Debugger attributes	439
17.5. Tool-Specific APIs	441
17.5.1. PMIx_tool_init	441
17.5.2. PMIx_tool_finalize	444
17.5.3. PMIx_tool_disconnect	445
17.5.4. PMIx_tool_attach_to_server	446
17.5.5. PMIx_tool_get_servers	447
17.5.6. PMIx_tool_set_server	448
17.5.7. PMIx_IOF_pull	449
17.5.8. PMIx_IOF_deregister	451
17.5.9. PMIx_IOF_push	452
18. Storage Support Definitions	455
18.1. Storage support constants	455
18.2. Storage support attributes	457
A. Python Bindings	459
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.3.3.3. 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. Revision History		529
B.1.	Version 1.0: June 12, 2015	529
B.2.	Version 2.0: Sept. 2018	530
B.2.1.	Removed/Modified Application Programming Interfaces (APIs)	530
B.2.2.	Deprecated constants	530
B.2.3.	Deprecated attributes	531
B.3.	Version 2.1: Dec. 2018	531
B.4.	Version 2.2: Jan 2019	532
B.5.	Version 3.0: Dec. 2018	532
B.5.1.	Removed constants	533

B.5.2.	Deprecated attributes	533
B.5.3.	Removed attributes	533
B.6.	Version 3.1: Jan. 2019	534
B.7.	Version 3.2: Oct. 2020	534
B.7.1.	Deprecated constants	535
B.7.2.	Deprecated attributes	536
B.8.	Version 4.0: Dec. 2020	537
B.8.1.	Added Constants	539
B.8.2.	Added Attributes	542
B.8.3.	Added Environmental Variables	555
B.8.4.	Added Macros	555
B.8.5.	Deprecated APIs	555
B.8.6.	Deprecated constants	556
B.8.7.	Removed constants	556
B.8.8.	Deprecated attributes	557
B.8.9.	Removed attributes	558
B.9.	Version 4.1: Oct. 2021	559
B.9.1.	Added Functions (Provisional)	559
B.9.2.	Added Data Structures (Provisional)	559
B.9.3.	Added Macros (Provisional)	559
B.9.4.	Added Constants (Provisional)	560
B.9.5.	Added Attributes (Provisional)	560
C. Acknowledgements		563
C.1.	Version 4.0	563
C.2.	Version 3.0	564
C.3.	Version 2.0	565
C.4.	Version 1.0	566
Bibliography		568
Index		569
Index of APIs		571

Index of Support Macros	579
Index of Data Structures	583
Index of Constants	585
Index of Environmental Variables	595
Index of Attributes	596

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.

13 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>

26 1.2 PMIx Architecture Overview

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

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

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

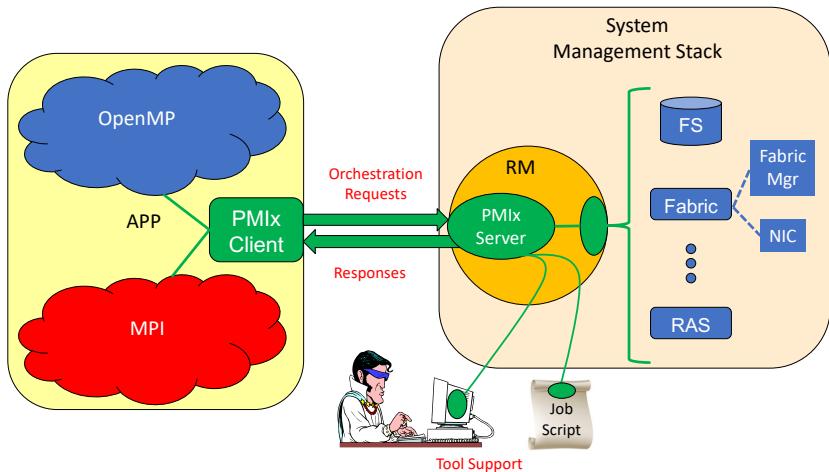


Figure 1.1.: PMIx-SMS Interactions

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

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

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

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

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

18 1.3 Portability of Functionality

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

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

36 1.3.1 Attributes in PMIx

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

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

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

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

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

21 It is therefore critical that users and application implementers:

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

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

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

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

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

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

16 **PMIX_ATTR_UNDEF** "pmix.undef" (NULL)

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

CHAPTER 2

PMIx Terms and Conventions

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

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

The following terminology is used throughout this document:

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

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

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

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

- *process* refers to an operating system process, also commonly referred to as a *heavyweight* process. A process is often comprised of multiple *lightweight threads*, commonly known as simply *threads*.
- *client* refers to a process that was registered with the PMIx server prior to being started, and connects to that PMIx server via `PMIx_Init` using its assigned namespace and rank with the information required to connect to that server being provided to the process at time of start of execution.
- *clone* refers to a process that was directly started by a PMIx client (e.g., using *fork/exec*) and calls `PMIx_Init`, thus connecting to its local PMIx server using the same namespace and rank as its parent process.
- *rank* refers to the numerical location (starting from zero) of a process within the defined scope. Thus, *job rank* is the rank of a process within its *job* and is synonymous with its unqualified *rank*, while *application rank* is the rank of that process within its *application*.
- *peer* refers to another process within the same *job*.
- *workflow* refers to an orchestrated execution plan frequently involving multiple *jobs* carried out under the control of a *workflow manager* process. An example workflow might first execute a computational job to generate the flow of liquid through a complex cavity, followed by a visualization job that takes the output of the first job as its input to produce an image output.
- *scheduler* refers to the component of the SMS responsible for scheduling of resource allocations. This is also generally referred to as the *system workflow manager* - for the purposes of this document, the *WLM* acronym will be used interchangeably to refer to the scheduler.
- *resource manager* is used in a generic sense to represent the subsystem that will host the PMIx server library. This could be a vendor-supplied resource manager or a third-party agent such as a programming model's runtime library.
- *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.
- *node* refers to a single operating system instance. Note that this may encompass one or more physical objects.
- *package* refers to a single object that is either soldered or connected to a printed circuit board via a mechanical socket. Packages may contain multiple chips that include (but are not limited to) processing units, memory, and peripheral interfaces.
- *processing unit*, or *PU*, is the electronic circuitry within a computer that executes instructions. Depending upon architecture and configuration settings, it may consist of a single hardware thread or multiple hardware threads collectively organized as a *core*.
- *fabric* is used in a generic sense to refer to the networks within the system regardless of speed or protocol. Any use of the term *network* in the document should be considered interchangeable with *fabric*.

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

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

2.1 Notational Conventions

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

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...

1 Rationale

2 Throughout this document, the rationale for the design choices made in the interface specification is
3 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.

4 Advice to users

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

7 Advice to PMIx library implementers

8 Throughout this document, material that is primarily commentary to PMIx library implementers is
9 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.

10 Advice to PMIx server hosts

11 Throughout this document, material that is primarily commentary aimed at host environments (e.g.,
12 RMIs and RunTime Environments (RTEs)) providing support for the PMIx server library is set off in
13 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.

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

17 2.2 Semantics

18 The following terms will be taken to mean:

- 19 • *shall, must* and *will* indicate that the specified behavior is *required* of all conforming
20 implementations
- 21 • *should* and *may* indicate behaviors that a complete implementation would include, but are not
22 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 • Underscores are used to separate words in a function or variable name.
- 6 • Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the
7 first letter of the word following it. For example, **PMIx_Get_version**.
- 8 • PMIx server and tool APIs are all lower case letters following the prefix - e.g.,
9 **PMIx_server_register_nspace**.
- 10 • The **PMIx_** prefix is used to denote functions.
- 11 • The **pmix_** prefix is used to denote function pointer and type definitions.

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

14 2.4 Procedure Conventions

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

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

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

29 A similar model is used for the server module functions (see 16.3.1). In this case, the PMIx library
30 is making an upcall into its host via the PMIx server module callback function and passing a
31 specific callback function pointer and callback data object. The PMIx library expects the host to
32 call the cbfunc with the necessary arguments and pass back the original callback data object upon
33 completing the operation. This gives the server-side PMIx library the ability to associate a context

1 with the call back (since multiple operations may be outstanding). The host has no visibility into
2 the contents of the callback data object object, nor is permitted to alter it in any way.

CHAPTER 3

Data Structures and Types

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

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

Advice to PMIx library implementers

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

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

Advice to users

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

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

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

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

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

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

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

9 3.1 Constants

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

15 **PMIX_MAX_NSLEN** Maximum namespace string length as an integer.

Advice to PMIx library implementers

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

19 **PMIX_MAX_KEYLEN** Maximum key string length as an integer.

Advice to PMIx library implementers

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

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

3.1.1 PMIx Return Status Constants

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

Advice to PMIx library implementers

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

Advice to users

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

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

`PMIX_SUCCESS` Success.
`PMIX_ERROR` General Error.
`PMIX_ERR_EXISTS` Requested operation would overwrite an existing value - typically returned when an operation would overwrite an existing file or directory.
`PMIX_ERR_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_INVALID_CRED` Invalid security credentials.
`PMIX_ERR_WOULD_BLOCK` Operation would block.
`PMIX_ERR_UNKNOWN_DATA_TYPE` The data type specified in an input to the PMIx library is not recognized by the implementation.
`PMIX_ERR_TYPE_MISMATCH` The data type found in an object does not match the expected data type as specified in the API call - e.g., a request to unpack a `PMIX_BOOL` value from a buffer that does not contain a value of that type in the current unpack location.
`PMIX_ERR_UNPACK_INADEQUATE_SPACE` Inadequate space to unpack data - the number of values in the buffer exceeds the specified number to unpack.
`PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER` Unpacking past the end of the provided buffer - the number of values in the buffer is less than the specified number to unpack, or a request was made to unpack a buffer beyond the buffer's end.
`PMIX_ERR_UNPACK_FAILURE` The unpack operation failed for an unspecified reason.
`PMIX_ERR_PACK_FAILURE` The pack operation failed for an unspecified reason.
`PMIX_ERR_NO_PERMISSIONS` The user lacks permissions to execute the specified operation.
`PMIX_ERR_TIMEOUT` Either a user-specified or system-internal timeout expired.

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

PMIX_ERR_BAD_PARAM One or more incorrect parameters (e.g., passing an attribute with a value of the wrong type), or multiple parameters containing conflicting directives (e.g., multiple instances of the same attribute with different values, or different attributes specifying conflicting behaviors), were passed to a PMIx API.

PMIX_ERR_EMPTY An array or list was given that has no members in it - i.e., the object is empty.

PMIX_ERR_RESOURCE_BUSY Resource busy - typically seen when an attempt to establish a connection to another process (e.g., a PMIx server) cannot be made due to a communication failure.

PMIX_ERR_OUT_OF_RESOURCE Resource exhausted.

PMIX_ERR_INIT Error during initialization.

PMIX_ERR_NOMEM Out of memory.

PMIX_ERR_NOT_FOUND The requested information was not found.

PMIX_ERR_NOT_SUPPORTED The requested operation is not supported by either the PMIx implementation or the host environment.

PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED The requested operation is supported by the PMIx implementation and (if applicable) the host environment. However, at least one supplied parameter was given an unsupported value, and the operation cannot therefore be executed as requested.

PMIX_ERR_COMM_FAILURE Communication failure - a message failed to be sent or received, but the connection remains intact.

PMIX_ERR_LOST_CONNECTION Lost connection between server and client or tool.

PMIX_ERR_INVALID_OPERATION The requested operation is supported by the implementation and host environment, but fails to meet a requirement (e.g., requesting to *disconnect* from processes without first *connecting* to them, inclusion of conflicting directives, or a request to perform an operation that conflicts with an ongoing one).

PMIX_OPERATION_IN_PROGRESS A requested operation is already in progress - the duplicate request shall therefore be ignored.

PMIX_OPERATION_SUCCEEDED The requested operation was performed atomically - no callback function will be executed.

PMIX_ERR_PARTIAL_SUCCESS The operation is considered successful but not all elements of the operation were concluded (e.g., some members of a group construct operation chose not to participate).

3.1.1.1 User-Defined Error and Event Constants

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

PMIX_EXTERNAL_ERR_BASE A starting point for user-level defined error and event constants. Negative values that are more negative than the defined constant are guaranteed not to conflict with PMIx values. Definitions should always be based on the

1 **PMIX_EXTERNAL_ERR_BASE** constant and not a specific value as the value of the constant
2 may change.

3 3.2 Data Types

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

6 3.2.1 Key Structure

7 The **pmix_key_t** structure is a statically defined character array of length
8 **PMIX_MAX_KEYLEN+1**, thus supporting keys of maximum length **PMIX_MAX_KEYLEN** while
9 preserving space for a mandatory **NULL** terminator.

PMIx v2.0

10 **typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];**

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

Advice to users

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

17 Passing a **pmix_key_t** value to the standard *sizeof* utility can result in compiler warnings of
18 incorrect returned value. Users are advised to avoid using *sizeof(pmix_key_t)* and instead rely on
19 the **PMIX_MAX_KEYLEN** constant.

20 3.2.1.1 Key support macros

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

1 **Check key macro**
2 Compare the key in a `pmix_info_t` to a given value.
3 
4 **IN a**
5 Pointer to the structure whose key is to be checked (pointer to `pmix_info_t`)
6 **IN b**
7 String value to be compared against (`char*`)
8 Returns `true` if the key matches the given value

9 **Check reserved key macro**
10 Check if the given key is a PMIx *reserved* key as described in Chapter 6.
11 
12 **IN a**
13 String value to be checked (`char*`)
14 Returns `true` if the key is reserved by the Standard.

15 **Load key macro**
16 Load a key into a `pmix_info_t`.
17 
18 **IN a**
19 Pointer to the structure whose key is to be loaded (pointer to `pmix_info_t`)
20 **IN b**
21 String value to be loaded (`char*`)
22 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.

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

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

20 `PMIX_CHECK_NSPACE(a, b)`

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

22 **IN b** String value to be compared against (`char*`)
23
24 Returns `true` if the namespace matches the given value

```

1  Check invalid namespace macro
2  Check if the provided pmix_nspace_t is invalid.
3  PMIX_NSPACE_INVALID (a)          C
4  IN a                                C
5    Pointer to the structure whose value is to be checked (pointer to pmix_nspace_t)
6  Returns true if the namespace is invalid (i.e., starts with a NULL resulting in a zero-length string
7  value)

8  Load namespace macro
9  Load a namespace into a pmix_nspace_t.
PMIx v4.0   C
10 PMIX_LOAD_NSPACE (a, b)          C
     C
11 IN a                                C
12  Pointer to the target structure (pointer to pmix_nspace_t)
13 IN b                                C
14  String value to be loaded - if NULL is given, then the target structure will be initialized to
15  zero's (char*)
16  No return value.

```

3.2.3 Rank Structure

```

18  The pmix_rank_t structure is a uint32_t type for rank values.
PMIx v1.0   C
19  typedef uint32_t pmix_rank_t;      C
     C
20  The following constants can be used to set a variable of the type pmix_rank_t. All definitions
21  were introduced in version 1 of the standard unless otherwise marked. Valid rank values start at
22  zero.
23  PMIX_RANK_UNDEF A value to request job-level data where the information itself is not
24    associated with any specific rank, or when passing a pmix_proc_t identifier to an
25    operation that only references the namespace field of that structure.
26  PMIX_RANK_WILDCARD A value to indicate that the user wants the data for the given key
27    from every rank that posted that key.
28  PMIX_RANK_LOCAL_NODE Special rank value used to define groups of ranks. This constant
29    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

```
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)
```

C

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

PMIx v4.1

```
18 PMIX_RANK_IS_VALID(a)
```

C

```
19 IN a  
20 Rank to be checked (pmix_rank_t)
```

C

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

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

C

C

3.2.4.1 Process structure support macros

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

Initialize the proc structure

Initialize the `pmix_proc_t` fields.

PMIx v1.0

C

`PMIX_PROC_CONSTRUCT (m)`

IN m

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

Destruct the proc structure

Destruct the `pmix_proc_t` fields.

C

`PMIX_PROC_DESTRUCT (m)`

IN m

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

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

Create a proc array

Allocate and initialize an array of `pmix_proc_t` structures.

PMIx v1.0

C

`PMIX_PROC_CREATE (m, n)`

INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

Free a proc structure

Release a `pmix_proc_t` structure.

PMIx v4.0

C

`PMIX_PROC_RELEASE (m)`

IN m

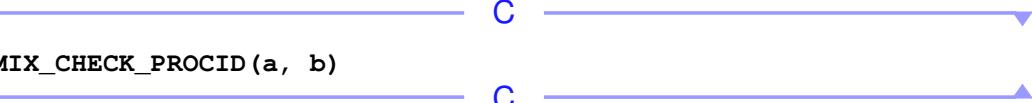
Pointer to a `pmix_proc_t` structure (handle)

1 **Free a proc array**
 2 Release an array of `pmix_proc_t` structures.
`PMIx v1.0` C
 3 `PMIX_PROC_FREE(m, n)` 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` C
 10 `PMIX_PROC_LOAD(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 No return value. Deprecated in favor of `PMIX_LOAD_PROCID`

18 **Compare identifiers**
 19 Compare two `pmix_proc_t` identifiers.
`PMIx v3.0` C
 20 `PMIX_CHECK_PROCID(a, b)` 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
- one of the ranks is `PMIX_RANK_WILDCARD`

```

1 Check if a process identifier is valid
2 Check for invalid namespace or rank value
3 PMIX_PROCID_INVALID (a)
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) nspace field or a rank
7 field of PMIX_RANK_INVALID

8 Load a procID structure
9 Load values into a pmix_proc_t.
PMIx v4.0
10 PMIX_LOAD_PROCID (m, n, r)
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 procID structure
18 Transfer contents of one pmix_proc_t value to another pmix_proc_t.
PMIx v4.1
19 PMIX_PROCID_XFER (d, s)
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)

```

1 **Construct a multi-cluster namespace**

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

3 **PMIX_MULTICLUSTER_NSPACE_CONSTRUCT (m, n, r)**

4 IN m

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

6 IN n

7 Cluster identifier (`char*`)

8 IN n

9 Namespace to be loaded (`pmix_nspace_t`)

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

11 **Parse a multi-cluster namespace**

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

PMIx v4.0

13 **PMIX_MULTICLUSTER_NSPACE_PARSE (m, n, r)**

14 IN m

15 `pmix_nspace_t` structure containing the multi-cluster identifier (pointer to
16 `pmix_nspace_t`)

17 IN n

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

19 IN n

20 Location where the namespace is to be stored (`pmix_nspace_t`)

21 **3.2.5 Process State Structure**

PMIx v2.0

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

Advice to users

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

```

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

```

32 3.2.6 Process Information Structure

33 The `pmix_proc_info_t` structure defines a set of information about a specific process
34 including it's name, location, and state.

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

```
PMIX_PROC_INFO_CONSTRUCT(m)
```

IN m

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

Destruct the process information structure

Destruct the `pmix_proc_info_t` fields.

PMIx v2.0

```
PMIX_PROC_INFO_DESTRUCT(m)
```

IN m

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

```

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

```

3.2.7 Job State Structure

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

Advice to users

The fine-grained nature of the following constants may exceed the ability of an RM to provide 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.
```

3.2.8 Value Structure

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

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

```

3.2.8.1 Value structure support macros

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

Initialize the value structure

Initialize the `pmix_value_t` fields.

PMIx v1.0

`PMIX_VALUE_CONSTRUCT (m)`

IN *m*

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

Destruct the value structure

Destruct the `pmix_value_t` fields.

PMIx v1.0

`PMIX_VALUE_DESTRUCT (m)`

IN *m*

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.  
3          PMIx v1.0   C  
4          INOUT m  
5              Address where the pointer to the array of pmix_value_t structures shall be stored (handle)  
6          IN n  
7              Number of structures to be allocated (size_t)  
  
8      Free a value structure  
9      Release a pmix_value_t structure.  
PMIx v4.0   C  
10     PMIX_VALUE_RELEASE (m)    C  
  
11     IN m  
12         Pointer to a pmix_value_t structure (handle)  
  
13     Free a value array  
14     Release an array of pmix_value_t structures.  
PMIx v1.0   C  
15     PMIX_VALUE_FREE (m, n)    C  
  
16     IN m  
17         Pointer to the array of pmix_value_t structures (handle)  
18     IN n  
19         Number of structures in the array (size_t)  
  
20     Load a value structure  
21     Load data into a pmix_value_t structure.  
PMIx v2.0
```

1 C
2 **PMIX_VALUE_LOAD**(*v*, *d*, *t*); C
3 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.

10 **Advice to users**

11 The data will be copied into the [pmix_value_t](#) - thus, any data stored in the source value can be
12 modified or free'd without affecting the copied data once the macro has completed.

12 **Unload a value structure**

13 Unload data from a [pmix_value_t](#) structure.

PMIx v2.2

14 C
15 **PMIX_VALUE_UNLOAD**(*r*, *v*, *d*, *t*); C
16 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](#).

24 **Advice to users**

25 Memory will be allocated and the data will be in the [pmix_value_t](#) returned - the source
26 [pmix_value_t](#) will not be altered.

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

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

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.

12 **Advice to users**

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

14 **Retrieve a numerical value from a value struct**
15 Retrieve a numerical value from a `pmix_value_t` structure.
PMIx v3.0

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

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
`PMIX_ERR_BAD_PARAM` if it doesn't

3.2.9 Info Structure

The `pmix_info_t` structure defines a key/value pair with associated directive. All fields were 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;
```

3.2.9.1 Info structure support macros

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

Initialize the info structure

Initialize the `pmix_info_t` fields.

PMIx v1.0

```
13     PMIX_INFO_CONSTRUCT(m)
```

IN m

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

Destruct the info structure

Destruct the `pmix_info_t` fields.

PMIx v1.0

```
18     PMIX_INFO_DESTRUCT(m)
```

IN m

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

Create an info array

Allocate and initialize an array of info structures.

PMIx v1.0

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

INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

```
1      Free an info array  
2      Release an array of pmix_info_t structures.  
3      PMIx v1.0  
4      PMIX_INFO_FREE(m, n)  
5      IN m  
6          Pointer to the array of pmix_info_t structures (handle)  
7      IN n  
8          Number of structures in the array (size_t)  
9      PMIx v1.0  
10     Load key and value data into a info struct  
11     PMIX_INFO_LOAD(v, k, d, t);  
12     IN v  
13         Pointer to the pmix_info_t into which the key and data are to be loaded (pointer to  
14             pmix_info_t)  
15     IN k  
16         String key to be loaded - must be less than or equal to PMIX_MAX_KEYLEN in length  
17             (handle)  
18     IN d  
19         Pointer to the data value to be loaded (handle)  
20     IN t  
21         Type of the provided data value (pmix_data_type_t)  
22     This macro simplifies the loading of key and data into a pmix_info_t by correctly assigning  
23     values to the structure's fields.  
24
```

Advice to users

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

1 **Copy data between info structures**

2 Copy all data (including key, value, and directives) between two `pmix_info_t` structures.

3 `PMIX_INFO_XFER(d, s);`

4 **IN d**

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

6 **IN s**

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

8 This macro simplifies the transfer of data between two `pmix_info_t` structures.

9 **Advice to users**

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

12 **Test a boolean info struct**

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

PMIx v2.0

14 `PMIX_INFO_TRUE(m)`

15 **IN m**

16 Pointer to a `pmix_info_t` structure (handle)

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

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

20 **3.2.9.2 Info structure list macros**

21 Constructing an array of `pmix_info_t` is a fairly common operation. The following macros are
22 provided to simplify this construction.

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.

4 `PMIX_INFO_LIST_START(m)`

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

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

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`)

23 **Advice to users**

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

3 `PMIX_INFO_LIST_XFER(rc, m, s)`

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`)

10 **Advice to users**

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

15 `PMIX_INFO_LIST_CONVERT(rc, m, d)`

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

25 `PMIX_INFO_LIST_RELEASE(m)`

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.

3.2.10 Info Type Directives

PMIx v2.0

The `pmix_info_directives_t` structure is a `uint32_t` type that defines the behavior of 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

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

Advice to PMIx library implementers

The top 16-bits of the `pmix_info_directives_t` are reserved for internal use by PMIx library implementers - the PMIx standard will *not* specify their intent, leaving them for customized use by implementers. Implementers 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.

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

`PMIX_INFO_REQD` The behavior defined in the `pmix_info_t` array is required, and not optional. This is a bit-mask value.

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

`PMIX_INFO_ARRAY_END` Mark that this `pmix_info_t` struct is at the end of an array created by the `PMIX_INFO_CREATE` macro. This is a bit-mask value.

`PMIX_INFO_DIR_RESERVED` A bit-mask identifying the bits reserved for internal use by implementers - these currently are set as `0xfffff0000`.

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);
```

C

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);
```

C

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);
```

C

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.

1 **Test an info structure for *optional* directive**
2 Test a `pmix_info_t` structure, returning `true` if the structure is *optional*.

3 `PMIX_INFO_IS_OPTIONAL(info);`

4 **IN info**

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

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

8 **Mark a required attribute as processed**

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

PMIx v4.0

10 `PMIX_INFO_PROCESSED(info);`

11 **IN info**

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

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

15 **Test if a required attribute has been processed**

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

PMIx v4.0

17 `PMIX_INFO_WAS_PROCESSED(info);`

18 **IN info**

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

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

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

22 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

24 `PMIX_INFO_IS_END(info);`

25 **IN info**

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

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

3.2.11 Environmental Variable Structure

PMIx v3.0

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

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

3.2.11.1 Environmental variable support macros

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

Initialize the envar structure

Initialize the **pmix_envar_t** fields.

PMIx v3.0

```
17 PMIX_ENVAR_CONSTRUCT (m)
```

IN m

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

Destruct the envar structure

Clear the **pmix_envar_t** fields.

PMIx v3.0

```
22 PMIX_ENVAR_DESTRUCT (m)
```

IN m

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

1 **Create an envar array**
2 Allocate and initialize an array of `pmix_envar_t` structures.
3  `PMIX_ENVAR_CREATE(m, n)` 
4 **INOUT m**
5 Address where the pointer to the array of `pmix_envar_t` structures shall be stored (handle)
6 **IN n**
7 Number of structures to be allocated (`size_t`)

8 **Free an envar array**
9 Release an array of `pmix_envar_t` structures.
PMIx v3.0  `PMIX_ENVAR_FREE(m, n)` 
10 **IN m**
11 Pointer to the array of `pmix_envar_t` structures (handle)
12 **IN n**
13 Number of structures in the array (`size_t`)

15 **Load an envar structure**
16 Load values into a `pmix_envar_t`.
PMIx v2.0  `PMIX_ENVAR_LOAD(m, e, v, s)` 
17 **IN m**
18 Pointer to the structure to be loaded (pointer to `pmix_envar_t`)
19 **IN e**
20 Environmental variable name (`char*`)
21 **IN v**
22 Value of variable (`char*`)
23 **IN v**
24 Separator character (`char`)
25

3.2.12 Byte Object Type

The `pmix_byte_object_t` structure describes a raw byte sequence.

```
typedef struct pmix_byte_object {
    char *bytes;
    size_t size;
} pmix_byte_object_t;
```

3.2.12.1 Byte object support macros

The following macros support the `pmix_byte_object_t` structure.

Initialize the byte object structure

Initialize the `pmix_byte_object_t` fields.

PMIx v2.0

```
PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

IN m

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

Destruct the byte object structure

Clear the `pmix_byte_object_t` fields.

PMIx v2.0

```
PMIX_BYTE_OBJECT_DESTRUCT(m)
```

IN m

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

Create a byte object structure

Allocate and intitialize an array of `pmix_byte_object_t` structures.

PMIx v2.0

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

INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

```
1  Free a byte object array  
2  Release an array of pmix\_byte\_object\_t structures.  
3  PMIX_BYTE_OBJECT_FREE(m, n)  
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  
10 PMIX_BYTE_OBJECT_LOAD(b, d, s)  
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)
```

3.2.13 Data Array Structure

The [pmix_data_array_t](#) structure defines an array data structure.

```
PMIx v2.0  
19  typedef struct pmix_data_array {  
20      pmix_data_type_t type;  
21      size_t size;  
22      void *array;  
23  } pmix_data_array_t;
```

3.2.13.1 Data array support macros

The following macros support the [pmix_data_array_t](#) structure.

```

1 Initialize a data array structure
2 Initialize the pmix_data_array_t fields, allocating memory for the array of the indicated type.
3 PMIX_DATA_ARRAY_CONSTRUCT(m, n, t) 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.
12 PMIx v2.2 PMIX_DATA_ARRAY_CONSTRUCT(m) 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.
18 PMIx v2.2 PMIX_DATA_ARRAY_CREATE(m, n, t) 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.
27 PMIx v2.2 PMIX_DATA_ARRAY_FREE(m) C
28 IN m
29 Pointer to the structure to be released (pointer to pmix_data_array_t)

```

3.2.14 Argument Array Macros

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

Argument array extension

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

```
PMIX_ARGV_APPEND(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 prepend

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

```
PMIX_ARGV_PREPEND(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

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

Argument array extension - unique

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

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

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.

Argument array release

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

19 `PMIX_ARGV_FREE(a);`

20 **IN a**

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

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

1 **Argument array split**
2 Split a string into a NULL-terminated argv array.

3 **PMIX_ARGV_SPLIT(a, b, c);**

4 **OUT a**
5 Resulting argv-style array (**char****)
6 **IN b**
7 String to be split (**char***)
8 **IN c**
9 Delimiter character (**char**)

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

12 **Advice to users**

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

15 **Argument array join**
16 Join all the elements of an argv array into a single newly-allocated string.

17 **PMIX_ARGV_JOIN(a, b, c);**

18 **OUT a**
19 Resulting string (**char***)
20 **IN b**
21 Argv-style array to be joined (**char****)
22 **IN c**
23 Delimiter character (**char**)

24 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)

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

5  **Advice to users** 

6 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.


7

3.3 Generalized Data Types Used for Packing/Unpacking

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


11 **Advice to PMIx library implementers** 

12 The following constants can be used to set a variable of the type `pmix_data_type_t`. Data
13 types in the PMIx Standard are defined in terms of the C-programming language. Implementers
14 wishing to support other languages should provide the equivalent definitions in a
15 language-appropriate manner. Additionally, a PMIx implementation may choose to add additional
types.


16	<code>PMIX_UNDEF</code>	Undefined.
17	<code>PMIX_BOOL</code>	Boolean (converted to/from native <code>true/false</code>) (<code>bool</code>).
18	<code>PMIX_BYTE</code>	A byte of data (<code>uint8_t</code>).
19	<code>PMIX_STRING</code>	<code>NULL</code> terminated string (<code>char*</code>).
20	<code>PMIX_SIZE</code>	Size <code>size_t</code> .
21	<code>PMIX_PID</code>	Operating Process IDentifier (PID) (<code>pid_t</code>).
22	<code>PMIX_INT</code>	Integer (<code>int</code>).
23	<code>PMIX_INT8</code>	8-byte integer (<code>int8_t</code>).
24	<code>PMIX_INT16</code>	16-byte integer (<code>int16_t</code>).
25	<code>PMIX_INT32</code>	32-byte integer (<code>int32_t</code>).
26	<code>PMIX_INT64</code>	64-byte integer (<code>int64_t</code>).
27	<code>PMIX_UINT</code>	Unsigned integer (<code>unsigned int</code>).
28	<code>PMIX_UINT8</code>	Unsigned 8-byte integer (<code>uint8_t</code>).
29	<code>PMIX_UINT16</code>	Unsigned 16-byte integer (<code>uint16_t</code>).
30	<code>PMIX_UINT32</code>	Unsigned 32-byte integer (<code>uint32_t</code>).
31	<code>PMIX_UINT64</code>	Unsigned 64-byte integer (<code>uint64_t</code>).
32	<code>PMIX_FLOAT</code>	Float (<code>float</code>).
33	<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    Persistance (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_NSPACE Process namespace (pmix_nspace_t). %
25  PMIX_QUERY       Query structure (pmix_query_t).
26  PMIX_COMPRESSED_STRING String compressed with zlib (char*).
27  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_cfunc_t is used by the pmix_modex_cfunc_t and  
19 pmix_info_cfunc_t operations to indicate that the callback data may be reclaimed/freed by  
20 the caller.
```

21 PMIx v1.0 Format

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

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.

3.4.2 Op Callback Function

Summary

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

```
typedef void (*pmix_op_cfunc_t)
    (pmix_status_t status, void *cbdata);
```

IN **status**
Status associated with the operation (handle)

IN **cbdata**
Callback data passed to original API call (memory reference)

Description

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

3.4.3 Value Callback Function

Summary

The `pmix_value_cfunc_t` is used by `PMIx_Get_nb` to return data.

PMIx v1.0

```
typedef void (*pmix_value_cfunc_t)
    (pmix_status_t status,
     pmix_value_t *kv, void *cbdata);
```

IN **status**
Status associated with the operation (handle)
IN **kv**
Key/value pair representing the data (`pmix_value_t`)
IN **cbdata**
Callback data passed to original API call (memory reference)

Description

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

```
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)

9 3.5 PMIx Datatype Value String Representations

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

Summary

String representation of a `pmix_status_t`.

PMIx v1.0

```
const char*
PMIx_Error_string(pmix_status_t status);
```

Summary

String representation of a pmix proc state t.

PMIx v2.0

```
const char*  
PMIx_Proc_state_string(pmix_proc_state_t state);
```

Summary

String representation of a `pmix_scope_t`.

PMIx v2.0

```
const char*
PMIx_Scope_string(pmix_scope_t scope);
```

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

```
1      Summary  
2      String representation of a pmix\_ifc\_channel\_t.  
3      const char*  
4      PMIx_IOF_channel_string(pmix_ifc_channel_t channel);  
5      Summary  
6      String representation of a pmix\_job\_state\_t.  
7      PMIx v4.0  
8      const char*  
9      PMIx_Job_state_string(pmix_job_state_t state);  
10     PMIx v4.0  
11     Summary  
12     String representation of a PMIx attribute.  
13     PMIx v4.0  
14     const char*  
15     PMIx_Get_attribute_string(char *attributename);  
16     PMIx v4.0  
17     Summary  
18     Return the PMIx attribute name corresponding to the given attribute string.  
19     const char*  
20     PMIx_Get_attribute_name(char *attributestring);
```

1 **Summary**
2 String representation of a [pmix_device_type_t](#).

3 **C**

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

6 **C**

CHAPTER 4

Client Initialization and Finalization

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

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

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

4.1 PMIx_Initialized

Summary

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

Format

```
int PMIx_Initialized(void)
```

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

Rationale

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

Description

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

1 4.2 PMIx_Get_version

2 Summary

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

5 Format

```
6 const char* PMIx_Get_version(void)
```

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

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

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

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

2 **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)

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

5 **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)

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

8 **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)

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

12 **PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)

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

16 **PMIX_TCP_IF_INCLUDE** "pmix.tcp.ifinclude" (char*)

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

20 **PMIX_TCP_IF_EXCLUDE** "pmix.tcp.ifexclude" (char*)

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

24 **PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)

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

27 **PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)

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

30 **PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)

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

33 **PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)

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

36 **PMIX_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.mdl.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).



1 **Description**

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

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

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

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

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

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

20 **4.3.1 Initialization events**

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

22 `PMIX_MODEL_DECLARED` Model declared.

23 `PMIX_MODEL_RESOURCES` Resource usage by a programming model has changed.

24 `PMIX_OPENMP_PARALLEL_ENTERED` An OpenMP parallel code region has been entered.

25 `PMIX_OPENMP_PARALLEL_EXITED` An OpenMP parallel code region has completed.

26 **4.3.2 Initialization attributes**

27 The following attributes influence the behavior of `PMIx_Init`.

28 **4.3.2.1 Connection attributes**

29 These attributes are used to describe a TCP socket for rendezvous with the local RM by passing
30 them into the relevant initialization API - thus, they are not typically accessed via the `PMIx_Get`
31 API.

32 `PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)`

33 If provided, directs that the TCP URI be reported and indicates the desired method of
34 reporting: '-' for stdout, '+' 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.

```

4.3.2.2 Programming model attributes

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.mld.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 Format

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

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 corresponding to a PMIx error constant.

▼ 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.

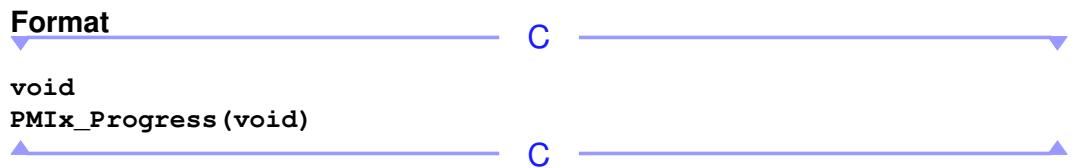
25 `PMIx_Finalize` does not include an internal barrier operation by default. This attribute
26 directs `PMIx_Finalize` to execute a barrier as part of the finalize operation.

27 4.5 PMIx_Progress

28 Summary

29 Progress the PMIx library.

1 **Format**



The diagram shows the word "Format" at the top left. A horizontal blue line extends from the right side of "Format" to the right edge of the slide. A blue downward-pointing triangle is positioned at the start of the line. A blue letter "C" is placed near the center of the line. Another blue downward-pointing triangle is located at the end of the line.

2 **void**
3 **PMIx_Progress (void)**



The diagram shows the function signature "void PMIx_Progress (void)" on three lines. A horizontal blue line starts under the first "void" and ends under the closing parenthesis. A blue upward-pointing triangle is at the start of the line, and another blue upward-pointing triangle is at the end of the line. A blue letter "C" is centered on the line.

4 **Description**

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

CHAPTER 5

Synchronization and Data Access Operations

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

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

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

11 5.1 PMIx_Fence

12 Summary

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

15 Format

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

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

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

Required Attributes

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

2 **PMIX_COLLECT_DATA** "pmix.collect" (bool)

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

8 **PMIX_COLLECT_GENERATED_JOB_INFO** "pmix.collect.gen" (bool)

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

Optional Attributes

15 The following attributes are optional for PMIx implementations:

16 **PMIX_ALL_CLONES_PARTICIPATE** "pmix.clone.part" (bool)

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

18 The following attributes are optional for host environments:

19 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

1 **Description**

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

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

8  **Advice to PMIx library implementers**

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


11  **Advice to PMIx server hosts**

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


14 **5.2 PMIx_Fence_nb**

15 **Summary**

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

1 Format

C

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

C

6 IN procs

7 Array of `pmix_proc_t` structures (array of handles)

8 IN nprocs

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

10 IN info

11 Array of info structures (array of handles)

12 IN ninfo

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

14 IN cbfunc

15 Callback function (function reference)

16 IN cbdata

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

18 Returns one of the following:

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

Required Attributes

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

28 **PMIX_COLLECT_DATA** "pmix.collect" (bool)

29 Collect all data posted by the participants using **PMIx_Put** that has been committed via
30 **PMIx_Commit**, making the collection locally available to each participant at the end of the
31 operation. By default, this will include all job-level information that was locally generated
32 by PMIx servers unless excluded using the **PMIX_COLLECT_GENERATED_JOB_INFO**
33 attribute.

34 **PMIX_COLLECT_GENERATED_JOB_INFO** "pmix.collect.gen" (bool)

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

Optional Attributes

7 The following attributes are optional for PMIx implementations:

8 **PMIX_ALL_CLONES_PARTICIPATE** "pmix.clone.part" (bool)
9 All *clones* of the calling process must participate in the collective operation.

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

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

Description

15 Nonblocking version of the **PMIx_Fence** routine. See the **PMIx_Fence** description for further
16 details.
17

5.2.1 Fence-related attributes

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

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

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

33 **PMIX_ALL_CLONES_PARTICIPATE** "pmix.clone.part" (bool)
34 All *clones* of the calling process must participate in the collective operation.

1 5.3 PMIx_Get

2 Summary

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

4 Format

```
5     pmix_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);
```

9 IN proc

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

11 IN key

12 Key to retrieve (**pmix_key_t**)

13 IN info

14 Array of info structures (array of handles)

15 IN ninfo

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

17 OUT val

18 value (handle)

19 Returns one of the following:

- 20 • **PMIX_SUCCESS** The requested data has been returned in the manner requested (i.e., in a
21 provided static memory location)
- 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 • a non-zero PMIx error constant indicating a reason for the request's failure.

29 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
the PMIx server if not found.

33 **PMIX_IMMEDIATE "pmix.immediate" (bool)**

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

3 **PMIX_DATA_SCOPE** "pmix.scope" (**pmix_scope_t**)

4 Scope of the data to be searched in a **PMIx_Get** call.

5 **PMIX_SESSION_INFO** "pmix.ssn.info" (**bool**)

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

7 **PMIX_JOB_INFO** "pmix.job.info" (**bool**)

8 Return information regarding the job realm corresponding to the namespace in the target
9 process' identifier.

10 **PMIX_APP_INFO** "pmix.app.info" (**bool**)

11 Return information regarding the application realm to which the target process belongs - the
12 namespace of the target process serves to identify the job containing the target application. If
13 information about an application other than the one containing the target process is desired,
14 then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired
15 target application. This is useful in cases where there are multiple applications and the
16 mapping of processes to applications is unclear.

17 **PMIX_NODE_INFO** "pmix.node.info" (**bool**)

18 Return information from the node realm regarding the node upon which the specified
19 process is executing. If information about a node other than the one containing the specified
20 process is desired, then the attribute array must also contain either the **PMIX_NODEID** or
21 **PMIX_HOSTNAME** attribute identifying the desired target. This is useful for requesting
22 information about a specific node even if the identity of processes running on that node are
23 not known..

24 **PMIX_GET_STATIC_VALUES** "pmix.get.static" (**bool**)

25 Request that the data be returned in the provided storage location. The caller is responsible
26 for destructing the **pmix_value_t** using the **PMIX_VALUE_DESTRUCT** macro when
27 done.

28 **PMIX_GET_POINTER_VALUES** "pmix.get.pntrs" (**bool**)

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

31 **PMIX_GET_REFRESH_CACHE** "pmix.get.refresh" (**bool**)

32 When retrieving data for a remote process, refresh the existing local data cache for the
33 process in case new values have been put and committed by the process since the last refresh.
34 Local process information is assumed to be automatically updated upon posting by the
35 process. A **NULL** key will cause all values associated with the process to be refreshed -
36 otherwise, only the indicated key will be updated. A process rank of
37 **PMIX_RANK_WILDCARD** can be used to update job-related information in dynamic
38 environments. The user is responsible for subsequently updating refreshed values they may
39 have cached in their own local memory.

Optional Attributes

1 The following attributes are optional for host environments:

2 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

Description

7 Retrieve information for the specified *key* associated with the process identified in the given
8 **pmix_proc_t**. See Chapters 6 and 7 for details on rules governing retrieval of information.
9 Information will be returned according to provided directives:

- 10 • In the absence of any directive, the returned **pmix_value_t** shall be an allocated memory
11 object. The caller is responsible for releasing the object when done.
- 12 • If **PMIX_GET_POINTER_VALUES** is given, then the function shall return a pointer to a
13 **pmix_value_t** in the PMIx library's memory that contains the requested information.
- 14 • If **PMIX_GET_STATIC_VALUES** is given, then the function shall return the information in the
15 provided **pmix_value_t** pointer. In this case, the caller must provide storage for the structure
16 and pass the pointer to that storage in the *val* parameter. If the implementation cannot return a
17 static value, then the call to **PMIx_Get** must return the **PMIX_ERR_NOT_SUPPORTED** status.

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

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

20 5.3.1 **PMIx_Get_nb**

21 Summary

22 Nonblocking **PMIx_Get** operation.

1 Format

C

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

C

6 IN proc

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

8 IN key

9 Key to retrieve (string)

10 IN info

11 Array of info structures (array of handles)

12 IN ninfo

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

14 IN cbfunc

15 Callback function (function reference)

16 IN cbdata

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

18 Returns one of the following:

- 19 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
20 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
21 function prior to returning from the API.
- 22 • a PMIx error constant indicating either an error in the input or that the request was immediately
23 processed and failed - the *cbfunc* will *not* be called.

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

- 26 • **PMIX_SUCCESS** The requested data has been returned.
- 27 • **PMIX_ERR_EXISTS_OUTSIDE_SCOPE** The requested key exists, but was posted in a *scope*
28 (see Section 7.1.1.1) that does not include the requester.
- 29 • **PMIX_ERR_NOT_FOUND** The requested data was not available.
- 30 • a non-zero PMIx error constant indicating a reason for the request's failure.

31  Required Attributes 

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

33 **PMIX_OPTIONAL "pmix.optional" (bool)**

34 Look only in the client's local data store for the requested value - do not request data from
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_POINTER_VALUES "pmix.get.pntrs" (bool)
26    Request that any pointers in the returned value point directly to values in the key-value store.
27    The user must not release any returned data pointers.
28  PMIX_GET_REFRESH_CACHE "pmix.get.refresh" (bool)
29    When retrieving data for a remote process, refresh the existing local data cache for the
30    process in case new values have been put and committed by the process since the last refresh.
31    Local process information is assumed to be automatically updated upon posting by the
32    process. A NULL key will cause all values associated with the process to be refreshed -
33    otherwise, only the indicated key will be updated. A process rank of
34    PMIX_RANK_WILDCARD can be used to update job-related information in dynamic
35    environments. The user is responsible for subsequently updating refreshed values they may
36    have cached in their own local memory.
```

37

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

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

Optional Attributes

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

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

Description

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:

```
16 PMIX_OPTIONAL "pmix.optional" (bool)
17     Look only in the client's local data store for the requested value - do not request data from
18     the PMIx server if not found.
19 PMIX_IMMEDIATE "pmix.immediate" (bool)
20     Specified operation should immediately return an error from the PMIx server if the requested
21     data cannot be found - do not request it from the host RM.
22 PMIX_GET_POINTER_VALUES "pmix.get.ptrs" (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_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_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_DATA_SCOPE "pmix.scope" (pmix_scope_t)**
10 Scope of the data to be searched in a **PMIx_Get** call.

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.

15 **PMIX_WAIT "pmix.wait" (int)**
16 Caller requests that the PMIx server wait until at least the specified number of values are
17 found (a value of zero indicates *all* and is the default).

18 5.4 Query

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

26 An example use-case for the **PMIx_Query_info_nb** API is to ensure clean job completion.
27 Time-shared systems frequently impose maximum run times when assigning jobs to resource
28 allocations. To shut down gracefully (e.g., to write a checkpoint before termination) it is necessary
29 for an application to periodically query the resource manager for the time remaining in its
30 allocation. This is especially true on systems for which allocation times may be shortened or
31 lengthened from the original time limit. Many resource managers provide APIs to dynamically
32 obtain this information, but each API is specific to the resource manager.

33 PMIx supports this use-case by defining an attribute key (**PMIX_TIME_REMAINING**) that can be
34 used with the **PMIx_Query_info_nb** interface to obtain the number of seconds remaining in
35 the current job allocation. Note that one could alternatively use the
36 **PMIx_Register_event_handler** API to register for an event indicating incipient job
37 termination, and then use the **PMIx_Job_control_nb** API to request that the host SMS
38 generate an event a specified amount of time prior to reaching the maximum run time. PMIx
39 provides such alternate methods as a means of maximizing the probability of a host system
40 supporting at least one method by which the application can obtain the desired service.

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

2 **5.4.1 PMIx_Resolve_peers**

3 **Summary**

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

5 **Format**

```
6 pmix_status_t  
7 PMIx_Resolve_peers(const char *nodename,  
8                     const pmix_nspace_t nspace,  
9                     pmix_proc_t **procs, size_t *nprocs);
```

10 **IN nodename**

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

12 **IN nspace**

13 namespace (string)

14 **OUT procs**

15 Array of process structures (array of handles)

16 **OUT nprocs**

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

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

19 **Description**

20 Given a *nodename*, return the array of processes within the specified *nspace* that are executing on
21 that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified
22 node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will
23 be zero. The caller is responsible for releasing the *procs* array when done with it. The
24 **PMIX_PROC_FREE** macro is provided for this purpose.

25 **5.4.2 PMIx_Resolve_nodes**

26 **Summary**

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

```
1 Format
2 pmix_status_t
3 PMIx_Resolve_nodes(const char *nspace, char **nodelist);
4 IN nspace
5 Namespace (string)
6 OUT nodelist
7 Comma-delimited list of nodenames (string)
8 Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

13 5.4.3 PMIx_Query_info

Summary

Query information about the system in general.

16 *PMIx v4.0*

17 **pmix_status_t**

18 **PMIx_Query_info**(**pmix_query_t** *queries*[], **size_t** *nqueries*,

19 **pmix_info_t** **info*[], **size_t** **ninfo*);

20 **IN queries**

21 Array of query structures (array of handles)

22 **IN nqueries**

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

24 **INOUT info**

25 Address where a pointer to an array of **pmix_info_t** containing the results of the query can

26 be returned (memory reference)

27 **INOUT ninfo**

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

29 Returns one of the following:

30

- **PMIX_SUCCESS** All data was found and has been returned.

31

- **PMIX_ERR_NOT_FOUND** None of the requested data was available. The *info* array will be

32 **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.proc.id" (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_NSPACE "pmix.nspace" (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_NSPACE** attribute.
16 Only required when the request is for information on a specific process.

17 **PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)**

18 Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or
19 more of **PMIX_CLIENT_FUNCTIONS**, **PMIX_SERVER_FUNCTIONS**,
20 **PMIX_TOOL_FUNCTIONS**, and **PMIX_HOST_FUNCTIONS**.

21 **PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)**

22 Request attributes supported by the PMIx client library.

23 **PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)**

24 Request attributes supported by the PMIx server library.

25 **PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)**

26 Request attributes supported by the host environment.

27 **PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)**

28 Request attributes supported by the PMIx tool library functions.

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

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



Optional Attributes

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

1 **PMIX_QUERY_NAMESPACES** "pmixqry.ns" (**char***)
2 Request a comma-delimited list of active namespaces. NO QUALIFIERS.
3
4 **PMIX_QUERY_JOB_STATUS** "pmixqry.jst" (**pmix_status_t**)
5 Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX_NSPACE**
6 indicating the namespace whose status is being queried.
7
8 **PMIX_QUERY_QUEUE_LIST** "pmixqry.qlst" (**char***)
9 Request a comma-delimited list of scheduler queues. NO QUALIFIERS.
10
11 **PMIX_QUERY_QUEUE_STATUS** "pmixqry.qst" (**char***)
12 Returns status of a specified scheduler queue, expressed as a string. OPTIONAL
13 QUALIFIERS: **PMIX_ALLOC_QUEUE** naming specific queue whose status is being
14 requested.
15
16 **PMIX_QUERY_PROC_TABLE** "pmixqry.ptable" (**char***)
17 Returns a (**pmix_data_array_t**) array of **pmix_proc_info_t**, one entry for each
18 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
19 **PMIX_NSPACE** indicating the namespace whose process table is being queried.
20
21 **PMIX_QUERY_LOCAL_PROC_TABLE** "pmixqry.lptable" (**char***)
22 Returns a (**pmix_data_array_t**) array of **pmix_proc_info_t**, one entry for each
23 process in the specified namespace executing on the same node as the requester, ordered by
24 process job rank. REQUIRED QUALIFIER: **PMIX_NSPACE** indicating the namespace
25 whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX_HOSTNAME**
26 indicating the host whose local process table is being queried. By default, the query assumes
27 that the host upon which the request was made is to be used.
28
29 **PMIX_QUERY_SPAWN_SUPPORT** "pmixqry.spawn" (**bool**)
30 Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.
31
32 **PMIX_QUERY_DEBUG_SUPPORT** "pmixqry.debug" (**bool**)
33 Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.
34
35 **PMIX_QUERY_MEMORY_USAGE** "pmixqry.mem" (**bool**)
36 Return information on memory usage for the processes indicated in the qualifiers.
37 OPTIONAL QUALIFIERS: **PMIX_NSPACE** and **PMIX_RANK**, or **PMIX_PROCID** of
38 specific process(es) whose memory usage is being requested.
39
40 **PMIX_QUERY_REPORT_AVG** "pmixqry.avg" (**bool**)
41 Report only average values for sampled information. NO QUALIFIERS.
42
43 **PMIX_QUERY_REPORT_MINMAX** "pmixqry.minmax" (**bool**)
44 Report minimum and maximum values. NO QUALIFIERS.
45
46 **PMIX_QUERY_ALLOC_STATUS** "pmixquery.alloc" (**char***)

```

1      String identifier of the allocation whose status is being requested. NO QUALIFIERS.
2      PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
3          Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
4          OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
5          requested (defaults to allocation containing the caller).
6      PMIX_SERVER_URI "pmix.srvr.uri" (char*)
7          URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
8          PMIx connection. Defaults to requesting the information for the local PMIx server.
9      PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float)
10         Average Megabytes of memory used by client processes on node. OPTIONAL
11         QUALIFIERS: PMIX_HOSTNAME or PMIX_NODEID (defaults to caller's node).
12      PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float)
13         Megabytes of memory currently used by the RM daemon on the node. OPTIONAL
14         QUALIFIERS: PMIX_HOSTNAME or PMIX_NODEID (defaults to caller's node).
15      PMIX_QUERY_AUTHORIZATIONS "pmixqry.auths" (bool)
16         Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.
17      PMIX_PROC_PID "pmix.ppid" (pid_t)
18         Operating system PID of specified process.
19      PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)
20         State of the specified process as of the last report - may not be the actual current state based
21         on update rate.

```



22 **Description**

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

27 The returned *status* indicates if requested data was found or not. The returned *info* array will
28 contain a **PMIX_QUERY_RESULTS** element for each query of the *queries* array. If qualifiers were
29 included in the query, then the first element of each results array shall contain the
30 **PMIX_QUERY_QUALIFIERS** key with a **pmix_data_array_t** containing the qualifiers. The
31 remaining **pmix_info_t** shall contain the results of the query, one entry for each key that was
32 found. Note that duplicate keys in the *queries* array shall result in duplicate responses within the
33 constraints of the accompanying qualifiers. The caller is responsible for releasing the returned array.

Advice to PMIx library implementers

Information returned from `PMIx_Query_info` shall 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.

5.4.4 `PMIx_Query_info_nb`

Summary

Query information about the system in general.

Format

```
pmix_status_t  
PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,  
                    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

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)

Returns one of the following:

- `PMIX_SUCCESS` indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed.

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

Required Attributes

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

11 **PMIX_QUERY_REFRESH_CACHE** "pmix.qry.rfsh" (bool)
 12 Retrieve updated information from server. NO QUALIFIERS.

13 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)
 14 Return information regarding the session realm of the target process.

15 **PMIX_JOB_INFO** "pmix.job.info" (bool)
 16 Return information regarding the job realm corresponding to the namespace in the target
 17 process' identifier.

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

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

32 **PMIX_PROC_INFO** "pmix.proc.info" (bool)
 33 Return information regarding the target process. This attribute is technically not required as
 34 the **PMIx_Get** API specifically identifies the target process in its parameters. However, it is
 35 included here for completeness.

36 **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_NSPACE "pmix.nspace" (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_NSPACE** attribute.
16 Only required when the request is for information on a specific process.

17 **PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)**

18 Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or
19 more of **PMIX_CLIENT_FUNCTIONS**, **PMIX_SERVER_FUNCTIONS**,
20 **PMIX_TOOL_FUNCTIONS**, and **PMIX_HOST_FUNCTIONS**.

21 **PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)**

22 Request attributes supported by the PMIx client library.

23 **PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)**

24 Request attributes supported by the PMIx server library.

25 **PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)**

26 Request attributes supported by the host environment.

27 **PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)**

28 Request attributes supported by the PMIx tool library functions.

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

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



Optional Attributes

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

1 **PMIX_QUERY_NAMESPACES** "pmix.qry.ns" (**char***)
2 Request a comma-delimited list of active namespaces. NO QUALIFIERS.
3
4 **PMIX_QUERY_JOB_STATUS** "pmix.qry.jst" (**pmix_status_t**)
5 Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX_NSPACE**
6 indicating the namespace whose status is being queried.
7
8 **PMIX_QUERY_QUEUE_LIST** "pmix.qry.qlst" (**char***)
9 Request a comma-delimited list of scheduler queues. NO QUALIFIERS.
10
11 **PMIX_QUERY_QUEUE_STATUS** "pmix.qry.qst" (**char***)
12 Returns status of a specified scheduler queue, expressed as a string. OPTIONAL
13 QUALIFIERS: **PMIX_ALLOC_QUEUE** naming specific queue whose status is being
14 requested.
15
16 **PMIX_QUERY_PROC_TABLE** "pmix.qry.ptable" (**char***)
17 Returns a (**pmix_data_array_t**) array of **pmix_proc_info_t**, one entry for each
18 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
19 **PMIX_NSPACE** indicating the namespace whose process table is being queried.
20
21 **PMIX_QUERY_LOCAL_PROC_TABLE** "pmix.qry.lptable" (**char***)
22 Returns a (**pmix_data_array_t**) array of **pmix_proc_info_t**, one entry for each
23 process in the specified namespace executing on the same node as the requester, ordered by
24 process job rank. REQUIRED QUALIFIER: **PMIX_NSPACE** indicating the namespace
25 whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX_HOSTNAME**
26 indicating the host whose local process table is being queried. By default, the query assumes
27 that the host upon which the request was made is to be used.
28
29 **PMIX_QUERY_SPAWN_SUPPORT** "pmix.qry.spawn" (**bool**)
30 Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.
31
32 **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (**bool**)
33 Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.
34
35 **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (**bool**)
36 Return information on memory usage for the processes indicated in the qualifiers.
37 OPTIONAL QUALIFIERS: **PMIX_NSPACE** and **PMIX_RANK**, or **PMIX_PROCID** of
38 specific process(es) whose memory usage is being requested.
39
40 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (**bool**)
41 Report only average values for sampled information. NO QUALIFIERS.
42
43 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (**bool**)
44 Report minimum and maximum values. NO QUALIFIERS.
45
46 **PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (**char***)

1 String identifier of the allocation whose status is being requested. NO QUALIFIERS.
2 **PMIX_TIME_REMAINING** "pmix.time.remaining" (**char***)
3 Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.
4 OPTIONAL QUALIFIERS: **PMIX_NSPACE** of the namespace whose info is being
5 requested (defaults to allocation containing the caller).
6 **PMIX_SERVER_URI** "pmix.srvr.uri" (**char***)
7 URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
8 PMIx connection. Defaults to requesting the information for the local PMIx server.
9 **PMIX_CLIENT_AVG_MEMORY** "pmix.cl.mem.avg" (**float**)
10 Average Megabytes of memory used by client processes on node. OPTIONAL
11 QUALIFIERS: **PMIX_HOSTNAME** or **PMIX_NODEID** (defaults to caller's node).
12 **PMIX_DAEMON_MEMORY** "pmix.dmn.mem" (**float**)
13 Megabytes of memory currently used by the RM daemon on the node. OPTIONAL
14 QUALIFIERS: **PMIX_HOSTNAME** or **PMIX_NODEID** (defaults to caller's node).
15 **PMIX_QUERY_AUTHORIZATIONS** "pmix.qry.auths" (**bool**)
16 Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.
17 **PMIX_PROC_PID** "pmix.ppid" (**pid_t**)
18 Operating system PID of specified process.
19 **PMIX_PROC_STATE_STATUS** "pmix.proc.state" (**pmix_proc_state_t**)
20 State of the specified process as of the last report - may not be the actual current state based
21 on update rate.



22 **Description**

23 Non-blocking form of the **PMIx_Query_info** API.

24 **5.4.5 Query-specific constants**

25 **PMIX_QUERY_PARTIAL_SUCCESS** Some, but not all, of the requested information was
26 returned.

5.4.6 Query attributes

Attributes used to direct behavior of the `PMIx_Query_info` APIs.

`PMIX_QUERY_RESULTS "pmix.qry.res" (pmix_data_array_t)`
Contains an array of query results for a given `pmix_query_t` passed to the `PMIx_Query_info` APIs. If qualifiers were included in the query, then the first element of the array shall be the `PMIX_QUERY_QUALIFIERS` attribute containing those qualifiers. Each of the remaining elements of the array is a `pmix_info_t` containing the query key and the corresponding value returned by the query. This attribute is solely for reporting purposes and cannot be used in `PMIx_Get` or other query operations.

`PMIX_QUERY_QUALIFIERS "pmix.qry.quals" (pmix_data_array_t)`
Contains an array of qualifiers that were included in the query that produced the provided results. This attribute is solely for reporting purposes and cannot be used in `PMIx_Get` or other query operations.

`PMIX_QUERY_SUPPORTED_KEYS "pmix.qry.keys" (char*)`
Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS.

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

`PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)`
Retrieve updated information from server. NO QUALIFIERS.

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

`PMIX_QUERY_NAMESPACE_INFO "pmix.qry.nsinfo" (pmix_data_array_t*)`
Return an array of active namespace information - each element will itself contain an array including the namespace plus the command line of the application executing within it.
OPTIONAL QUALIFIERS: `PMIX_NSPACE` of specific namespace whose info is being requested.

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

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

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

`PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)`
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_NSPACE` indicating the namespace whose process table is being queried.

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

```

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

7 PMIX_QUERY_AUTHORIZATIONS "pmix.qry.auths" (bool)
8     Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.
9 PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
10    Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.
11 PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
12    Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.
13 PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
14    Return information on memory usage for the processes indicated in the qualifiers.
15    OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
16    specific process(es) whose memory usage is being requested.
17 PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
18    Constrain the query to local information only. NO QUALIFIERS.
19 PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
20    Report only average values for sampled information. NO QUALIFIERS.
21 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
22    Report minimum and maximum values. NO QUALIFIERS.
23 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
24    String identifier of the allocation whose status is being requested. NO QUALIFIERS.
25 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
26    Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
27    OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
28    requested (defaults to allocation containing the caller).
29 PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
30    Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or
31    more of PMIX_CLIENT_FUNCTIONS, PMIX_SERVER_FUNCTIONS,
32    PMIX_TOOL_FUNCTIONS, and PMIX_HOST_FUNCTIONS.
33 PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
34    Return the number of process sets defined in the specified range (defaults to
35    PMIX_RANGE_SESSION).

36 PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)
37    Return a pmix_data_array_t containing an array of strings of the process set names
38    defined in the specified range (defaults to PMIX_RANGE_SESSION).

39 PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t*)
40    Return an array of pmix_proc_t containing the members of the specified process set.

41 PMIX_QUERY_AVAIL_SERVERS "pmix.qry.asrvrs" (pmix_data_array_t*)

```

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

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

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

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

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

35 5.4.7 Query Structure

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

PMIx v2.0

```
1  typedef struct pmix_query {  
2      char **keys;  
3      pmix_info_t *qualifiers;  
4      size_t nqual;  
5  } pmix_query_t;
```

6 where:

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

10 5.4.7.1 Query structure support macros

11 The following macros are provided to support the **pmix_query_t** structure.

12 Initialize the query structure

13 Initialize the **pmix_query_t** fields

PMIx v2.0

```
14 PMIX_QUERY_CONSTRUCT (m)
```

15 IN m

16 Pointer to the structure to be initialized (pointer to **pmix_query_t**)

17 Destruct the query structure

18 Destruct the **pmix_query_t** fields

PMIx v2.0

```
19 PMIX_QUERY_DESTRUCT (m)
```

20 IN m

21 Pointer to the structure to be destructed (pointer to **pmix_query_t**)

```

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

```

1 5.5 Using Get vs Query

2 Both **PMIx_Get** and **PMIx_Query_info** can be used to retrieve information about the system.
3 In general, the *get* operation should be used to retrieve:

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

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

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

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

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

29 5.6 Accessing attribute support information

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

1 the function names to the array of *keys*. Function names *must* be given as user-level API names -
2 e.g., “PMIx_Get”, “PMIx_server_setup_application”, or “PMIx_tool_attach_to_server”.

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

- 8 • **PMIX_CLIENT_FUNCTIONS** "pmix.client.fns" (bool)
9 Request a list of functions supported by the PMIx client library.
- 10 • **PMIX_CLIENT_ATTRIBUTES** "pmix.client.attrs" (bool)
11 Request attributes supported by the PMIx client library.
- 12 • **PMIX_SERVER_FUNCTIONS** "pmix.srvr.fns" (bool)
13 Request a list of functions supported by the PMIx server library.
- 14 • **PMIX_SERVER_ATTRIBUTES** "pmix.srvr.attrs" (bool)
15 Request attributes supported by the PMIx server library.
- 16 • **PMIX_HOST_FUNCTIONS** "pmix.srvr.fns" (bool)
17 Request a list of functions supported by the host environment.
- 18 • **PMIX_HOST_ATTRIBUTES** "pmix.host.attrs" (bool)
19 Request attributes supported by the host environment.
- 20 • **PMIX_TOOL_FUNCTIONS** "pmix.tool.fns" (bool)
21 Request a list of functions supported by the PMIx tool library.
- 22 • **PMIX_TOOL_ATTRIBUTES** "pmix.setup.env" (bool)
23 Request attributes supported by the PMIx tool library functions.

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

33 The array of returned structures, and their child arrays, are subject to the return rules for the
34 **PMIx_Query_info_nb** API. For example, a request for supported attributes of the **PMIx_Get**
35 function that includes the *host* level will return values for the *client* and *server* levels, plus an array
36 element with a *key* of **PMIX_HOST_ATTRIBUTES** and a value type of **PMIX_UNDEF** indicating
37 that no attributes are supported at that level.

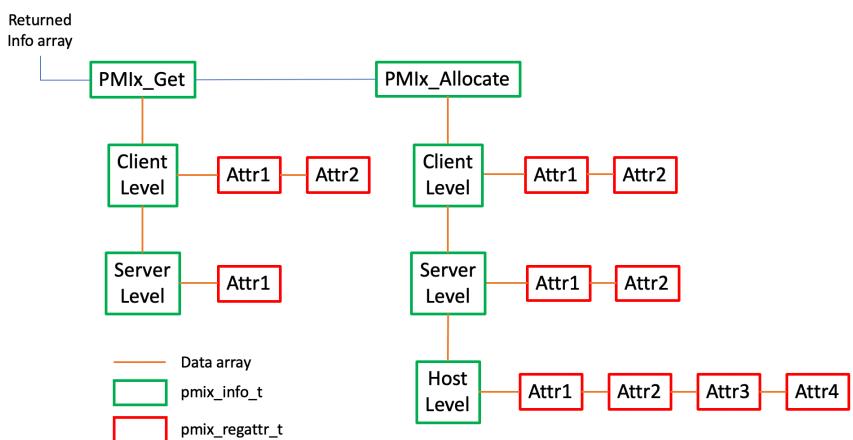


Figure 5.1.: Returned information hierarchy for attribute support request

CHAPTER 6

Reserved Keys

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

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

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

6.1 Data realms

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

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

24 **PMIX_SESSION_INFO "pmix.ssn.info" (bool)**

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

26 **PMIX_JOB_INFO "pmix.job.info" (bool)**

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

29 **PMIX_APP_INFO "pmix.app.info" (bool)**

1 Return information regarding the application realm to which the target process belongs - the
2 namespace of the target process serves to identify the job containing the target application. If
3 information about an application other than the one containing the target process is desired,
4 then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired
5 target application. This is useful in cases where there are multiple applications and the
6 mapping of processes to applications is unclear.

7 **PMIX_PROC_INFO "pmix.proc.info" (bool)**

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

11 **PMIX_NODE_INFO "pmix.node.info" (bool)**

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

Advice to users

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

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

24 **6.1.1 Session realm attributes**

25 If information about a session other than the one containing the requesting process is desired, then
26 the *info* array passed to **PMIx_Get** must contain a **PMIX_SESSION_ID** attribute identifying the
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 [16.2.3.2](#)
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 16.2.3.2
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.

```

6.1.2 Job realm attributes

Job-related information is retrieved by including the namespace of the target job and a rank of **PMIX_RANK_WILDCARD** in the *proc* argument passed to **PMIX_Get**. If desired for code clarity, the caller can also include the **PMIX_JOB_INFO** attribute in the *info* array, though this is not required. If information is requested about a namespace in a session other than the one containing the requesting process, then the *info* array must contain a **PMIX_SESSION_ID** attribute identifying the desired target session. This is required as many environments only guarantee unique namespaces within a session, and not across sessions.

Job-level information includes the following attributes:

```

22  PMIX_NSPACE "pmix.nspace" (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 [16.2.3.2](#)
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 [16.2.3.2](#)
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.nsdir" (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.

1 6.1.3 Application realm attributes

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

10 Application-level information includes the following attributes:

11 **PMIX_APPNUM** "pmix.appnum" (**uint32_t**)

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

13 **PMIX_NUM_NODES** "pmix.num.nodes" (**uint32_t**)

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

17 **PMIX_APPLDR** "pmix.aldr" (**pmix_rank_t**)

18 Lowest rank in the specified application.

19 **PMIX_APP_SIZE** "pmix.app.size" (**uint32_t**)

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

22 **PMIX_APP_ARGV** "pmix.app.argv" (**char***)

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

25 **PMIX_MAX_PROCS** "pmix.max.size" (**uint32_t**)

26 Maximum number of processes that can be executed in the specified realm. Typically, this is
27 a constraint imposed by a scheduler or by user settings in a hostfile or other resource
28 description. Defaults to the *job* realm. In this context, this is the maximum number of
29 processes that can be executed in the specified application, which may be a subset of the
30 number allocated to the overall session and job.

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. In this context, this is the number
35 of slots assigned to the specified application, which may be a subset of the slots allocated to
36 the overall session and job.

37 **PMIX_NODE_MAP** "pmix.nmap" (**char***)

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

6.1.4 Process realm attributes

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

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_CPUTSET "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_CPUTSET_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 14.3 or the distance
37       attributes of Section 11.4.11.

```

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.lldr" (**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_CPUSETS** "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 14.3.

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*.

6.2.1.1 Session-level information

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

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);
```

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

```
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);
```

1 6.2.1.2 Job-level information

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

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

19 6.2.1.3 Application-level information

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

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

```

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);

```



6.2.1.4 Process-level information

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

6.2.1.5 Node-level information

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



```

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 */

```

```
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.

7.1 Posting Key/Value Pairs

PMIx clients can post non-reserved key-value pairs associated with themselves by using **PMIx_Put**. Alternatively, PMIx clients can cache arbitrary key-value pairs accessible only by the caller via the **PMIx_Store_internal** API.

7.1.1 PMIx_Put

Summary

Post a key/value pair for distribution.

Format

```
pmix_status_t  
PMIx_Put (pmix_scope_t scope,  
          const pmix_key_t key,  
          pmix_value_t *val);
```

IN scope

Distribution scope of the provided value (handle)

IN key

key (pmix_key_t)

IN value

Reference to a pmix_value_t structure (handle)

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

Description

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

The provided *scope* determines the ability of other processes to access the posted data, as defined in Section 7.1.1.1 on page 114. Specific implementations may support different scope values, but all implementations must support at least **PMIX_GLOBAL**.

The **pmix_value_t** structure supports both string and binary values. PMIx implementations are required to support heterogeneous environments by properly converting binary values between host architectures, and will copy the provided *value* into internal memory prior to returning from **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

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

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

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

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

1 **Description**

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

5 **7.1.3 PMIx_Commit**

6 **Summary**

7 Post all previously [PMIx_Put](#) values for distribution.

8 **Format**

9 `pmix_status_t PMIx_Commit(void);`

10 Returns [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

11 **Description**

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

17 **Advice to users**

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

1 7.2 Retrieval rules for non-reserved keys

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

- 5 1. If the **PMIX_GET_REFRESH_CACHE** attribute is given, then the request is first forwarded to
6 the local PMIx server which will then update the client's cache. Note that this may not,
7 depending upon implementation details, result in any action.
- 8 2. Check the local PMIx client cache for the requested key - if not found and either the
9 **PMIX_OPTIONAL** or **PMIX_GET_REFRESH_CACHE** attribute was given, the search will stop
10 at this point and return the **PMIX_ERR_NOT_FOUND** status.
- 11 3. Request the information from the local PMIx server. The server will check its cache for the
12 specified key 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_dmode_x_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

Publish/Lookup Operations

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

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

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

8.1 PMIx_Publish

Summary

Publish data for later access via [PMIx_Lookup](#).

Format

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

IN info

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

IN ninfo

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

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

Required Attributes

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

Optional Attributes

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

- 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 **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.
- 14 **PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)
15 Declare how long the datastore shall retain the provided data. The datastore is to delete the
16 data upon reaching the persistence criterion.
- 17 **PMIX_ACCESS_PERMISSIONS** "pmix.aperms" (pmix_data_array_t)
18 Define access permissions for the published data. The value shall contain an array of
19 **pmix_info_t** structs containing the specified permissions.

Description

Publish the data in the *info* array for subsequent lookup. By default, the data will be published into the **PMIX_RANGE_SESSION** range and with **PMIX_PERSIST_APP** persistence. Changes to those values, and any additional directives, can be included in the **pmix_info_t** array. Attempts to access the data by processes outside of the provided data range shall be rejected. The **PMIX_PERSISTENCE** attribute instructs the datastore holding the published information as to how long that information is to be retained.

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

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

1 8.2 PMIx_Publish_nb

2 Summary

3 Nonblocking [PMIx_Publish](#) routine.

4 Format

```
5     pmix_status_t  
6     PMIx_Publish_nb(const pmix_info_t info[], size_t ninfo,  
7                         pmix_op_cbfunc_t cbfunc, void *cbdata);
```

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 Returns one of the following:

- 17 • [PMIX_SUCCESS](#), indicating that the request is being processed by the host environment - result
18 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
19 function prior to returning from the API.
- 20 • [PMIX_OPERATION_SUCCEEDED](#), indicating that the request was immediately processed and
21 returned *success* - the *cbfunc* will *not* be called.
- 22 • a PMIx error constant indicating either an error in the input or that the request was immediately
23 processed and failed - the *cbfunc* will *not* be called.

24 Required Attributes

25 There are no required attributes for this API. PMIx implementations that do not directly support the
26 operation but are hosted by environments that do support it must pass any attributes that are
27 provided by the client to the host environment for processing. In addition, the PMIx library is
28 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

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

2 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

6 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)

7 Define constraints on the processes that can access the provided data. Only processes that
8 meet the constraints are allowed to access it.

9 **PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)

10 Declare how long the datastore shall retain the provided data. The datastore is to delete the
11 data upon reaching the persistence criterion.

12 **PMIX_ACCESS_PERMISSIONS** "pmix.aperms" (pmix_data_array_t)

13 Define access permissions for the published data. The value shall contain an array of
14 **pmix_info_t** structs containing the specified permissions.

15 **Description**

16 Nonblocking **PMIx_Publish** routine.

17 8.3 Publish-specific constants

18 The following constants are defined for use with the **PMIx_Publish** APIs:

19 **PMIX_ERR_DUPLICATE_KEY** The provided key has already been published on the same
20 data range.

21 8.4 Publish-specific attributes

22 The following attributes are defined for use with the **PMIx_Publish** APIs:

23 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)

24 Define constraints on the processes that can access the provided data. Only processes that
25 meet the constraints are allowed to access it.

26 **PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)

27 Declare how long the datastore shall retain the provided data. The datastore is to delete the
28 data upon reaching the persistence criterion.

29 **PMIX_ACCESS_PERMISSIONS** "pmix.aperms" (pmix_data_array_t)

30 Define access permissions for the published data. The value shall contain an array of
31 **pmix_info_t** structs containing the specified permissions.

```
1 PMIX_ACCESS_USERIDS "pmix.auids" (pmix_data_array_t)
2     Array of effective User IDs (UIDs) that are allowed to access the published data.
3 PMIX_ACCESS_GRPIDS "pmix.agids" (pmix_data_array_t)
4     Array of effective Group IDs (GIDs) that are allowed to access the published data.
```

5 8.5 Publish-Lookup Datatypes

6 The following data types are defined for use with the **PMIx_Publish** APIs.

7 8.5.1 Range of Published Data

8 *PMIx v1.0* The **pmix_data_range_t** structure is a **uint8_t** type that defines a range for both data
9 *published* via the **PMIx_Publish** API and generated events. The following constants can be used
10 to set a variable of the type **pmix_data_range_t**.

11 **PMIX_RANGE_UNDEF** Undefined range.
12 **PMIX_RANGE_RM** Data is intended for the host environment, or lookup is restricted to data
13 published by the host environment.
14 **PMIX_RANGE_LOCAL** Data is only available to processes on the local node, or lookup is
15 restricted to data published by processes on the local node of the requester.
16 **PMIX_RANGE_NAMESPACE** Data is only available to processes in the same namespace, or
17 lookup is restricted to data published by processes in the same namespace as the requester.
18 **PMIX_RANGE_SESSION** Data is only available to all processes in the session, or lookup is
19 restricted to data published by other processes in the same session as the requester.
20 **PMIX_RANGE_GLOBAL** Data is available to all processes, or lookup is open to data published
21 by anyone.
22 **PMIX_RANGE_CUSTOM** Data is available only to processes as specified in the
23 **pmix_info_t** associated with this call, or lookup is restricted to data published by
24 processes as specified in the **pmix_info_t**.
25 **PMIX_RANGE_PROC_LOCAL** Data is only available to this process, or lookup is restricted to
26 data published by this process.
27 **PMIX_RANGE_INVALID** Invalid value - typically used to indicate that a range has not yet
28 been set.

29 8.5.2 Data Persistence Structure

30 *PMIx v1.0* The **pmix_persistence_t** structure is a **uint8_t** type that defines the policy for data
31 published by clients via the **PMIx_Publish** API. The following constants can be used to set a
32 variable of the type **pmix_persistence_t**.

33 **PMIX_PERSIST_INDEF** Retain data until specifically deleted.
34 **PMIX_PERSIST_FIRST_READ** Retain data until the first access, then the data is deleted.
35 **PMIX_PERSIST_PROC** Retain data until the publishing process terminates.
36 **PMIX_PERSIST_APP** Retain data until the application terminates.
37 **PMIX_PERSIST_SESSION** Retain data until the session/allocation terminates.
38 **PMIX_PERSIST_INVALID** Invalid value - typically used to indicate that a persistence has
39 not yet been set.

8.6 PMIx_Lookup

Summary

Lookup information published by this or another process with `PMIx_Publish` or `PMIx_Publish_nb`.

Format

```
pmix_status_t  
PMIx_Lookup(pmix_pdata_t data[], size_t ndata,  
             const pmix_info_t info[], size_t ninfo);
```

INOUT data

Array of publishable data structures (array of `pmix_pdata_t`)

IN ndata

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

IN info

Array of info structures (array of `pmix_info_t`)

IN ninfo

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

Returns one of the following:

- `PMIX_SUCCESS` All data was found and has been returned.
- `PMIX_ERR_NOT_FOUND` None of the requested data could be found within the requester's range.
- `PMIX_ERR_PARTIAL_SUCCESS` Some of the requested data was found. Any key that cannot be found will return with a data type of `PMIX_UNDEF` in the associated *value* struct. 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

1 PMIx libraries are not required to directly support any attributes for this function. However, any
2 provided attributes must be passed to the host environment for processing, and the PMIx library is
3 required to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is
4 requesting the info.

Optional Attributes

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

6 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

10 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)

11 Define constraints on the processes that can access the provided data. Only processes that
12 meet the constraints are allowed to access it.

13 **PMIX_WAIT** "pmix.wait" (int)

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

Description

16 Lookup information published by this or another process. By default, the search will be constrained
17 to publishers that fall within the **PMIX_RANGE_SESSION** range in case duplicate keys exist on
18 different ranges. Changes to the range (e.g., expanding the search to all potential publishers via the
20 **PMIX_RANGE_GLOBAL** constant), and any additional directives, can be provided in the
21 **pmix_info_t** array. Data is returned per the retrieval rules of Section 8.8.

22 The *data* parameter consists of an array of **pmix_pdata_t** structures with the keys specifying the
23 requested information. Data will be returned for each **key** field in the associated **value** field of
24 this structure as per the above description of return values. The **proc** field in each
25 **pmix_pdata_t** structure will contain the namespace/rank of the process that published the data.

Advice to users

26 Although this is a blocking function, it will not wait by default for the requested data to be
27 published. Instead, it will block for the time required by the datastore to lookup its current data and
28 return any found items. Thus, the caller is responsible for either ensuring that data is published
29 prior to executing a lookup, using **PMIX_WAIT** to instruct the datastore to wait for the data to be
30 published, or retrying until the requested data is found.

1 8.7 PMIx_Lookup_nb

2 Summary

3 Nonblocking version of [PMIx_Lookup](#).

4 Format

```
5     pmix_status_t  
6     PMIx_Lookup_nb(char **keys,  
7                         const pmix_info_t info[], size_t ninfo,  
8                         pmix_lookup_cbfunc_t cbfunc, void *cbdata);
```

9 **IN keys**

10 NULL-terminated array of keys (array of strings)

11 **IN info**

12 Array of info structures (array of handles)

13 **IN ninfo**

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

15 **IN cbfunc**

16 Callback function (handle)

17 **IN cbdata**

18 Callback data to be provided to the callback function (pointer)

19 Returns one of the following:

- 20 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
21 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
22 function prior to returning from the API.
- 23 • a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called.

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

- 26 • **PMIX_SUCCESS** All data was found and has been returned.
- 27 • **PMIX_ERR_NOT_FOUND** None of the requested data was available within the requester's range.
28 The *pdata* array in the callback function shall be **NULL** and the *npdata* parameter set to zero.
- 29 • **PMIX_ERR_PARTIAL_SUCCESS** Some of the requested data was found. Only found data will
30 be included in the returned *pdata* array. Note that the specific reason for a particular piece of
31 missing information (e.g., lack of permissions) cannot be communicated back to the requester in
32 this situation.
- 33 • **PMIX_ERR_NOT_SUPPORTED** There is no available datastore (either at the host environment
34 or PMIx implementation level) on this system that supports this function.

- **PMIX_ERR_NO_PERMISSIONS** All of the requested data was found and range restrictions were met for each specified key, but none of the matching data could be returned due to lack of access permissions.
 - a non-zero PMIx error constant indicating a reason for the request's failure.

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host environment for processing, and the PMIx library is required to add the `PMIX_USERID` and the `PMIX_GRP_ID` attributes of the client process that is requesting the info.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

PMIX_WAIT "pmix.wait" (int)

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

Description

Non-blocking form of the [PMIx Lookup](#) function.

8.7.1 Lookup Returned Data Structure

The `pmix_pdata_t` structure is used by `PMIx_Lookup` to describe the data being accessed.

```
3     typedef struct pmix_pdata {  
4         pmix_proc_t proc;  
5         pmix_key_t key;  
6         pmix_value_t value;  
7     } pmix_pdata_t;
```

where:

- *proc* is the process identifier of the data publisher.
- *key* is the string key of the published data.
- *value* is the value associated with the *key*.

8.7.1.1 Lookup data structure support macros

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

Initialize the pdata structure

Initialize the `pmix_pdata_t` fields

PMIx v1.0

```
16 PMIX_PDATA_CONSTRUCT (m)
```

IN *m*

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

Destruct the pdata structure

Destruct the `pmix_pdata_t` fields

PMIx v1.0

```
21 PMIX_PDATA_DESTRUCT (m)
```

IN *m*

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

```
1 Create a pdata array
2 Allocate and initialize an array of pmix_pdata_t structures
3 PMIX_PDATA_CREATE (m, n)
4 INOUT m
5 Address where the pointer to the array of pmix_pdata_t structures shall be stored (handle)
6 IN n
7 Number of structures to be allocated (size_t)
8
9 Free a pdata structure
10 Release a pmix_pdata_t structure
PMIx v4.0
11 IN m
12 Pointer to a pmix_pdata_t structure (handle)
13
14 Free a pdata array
15 Release an array of pmix_pdata_t structures
PMIx v1.0
16 IN m
17 Pointer to the array of pmix_pdata_t structures (handle)
18 IN n
19 Number of structures in the array (size_t)
20
21 Load a lookup data structure
22 This macro simplifies the loading of key, process identifier, and data into a pmix_pdata_t by
correctly assigning values to the structure's fields.
PMIx v1.0
```

```
1 PMIX_PDATA_LOAD(m, p, k, d, t); C  
2 IN m Pointer to the pmix_pdata_t structure into which the key and data are to be loaded  
3 (pointer to pmix_pdata_t)  
4 IN p Pointer to the pmix_proc_t structure containing the identifier of the process being  
5 referenced (pointer to pmix_proc_t)  
6 IN k String key to be loaded - must be less than or equal to PMIX_MAX_KEYLEN in length  
7 (handle)  
8 IN d Pointer to the data value to be loaded (handle)  
9 IN t Type of the provided data value (pmix_data_type_t)
```

Advice to users

Key, process identifier, and data will all be copied into the pmix_pdata_t - thus, the source information can be modified or free'd without affecting the copied data once the macro has completed.

Transfer a lookup data structure

This macro simplifies the transfer of key, process identifier, and data value between two pmix_pdata_t structures.

PMIx v2.0

```
21 PMIX_PDATA_XFER(d, s); C  
22 IN d Pointer to the destination pmix_pdata_t (pointer to pmix_pdata_t)  
23 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.

1 8.7.2 Lookup Callback Function

2 Summary

3 The `pmix_lookup_cbfunc_t` is used by `PMIx_Lookup_nb` to return data.

4 PMIx v1.0

C

```
5      typedef void (*pmix_lookup_cbfunc_t)
6            (pmix_status_t status,
7            pmix_pdata_t data[], size_t ndata,
8            void *cbdata);
```

C

8 IN **status**

9 Status associated with the operation (handle)

10 IN **data**

11 Array of data returned (`pmix_pdata_t`)

12 IN **ndata**

13 Number of elements in the *data* array (`size_t`)

14 IN **cbdata**

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

16 Description

17 A callback function for calls to `PMIx_Lookup_nb`. The function will be called upon completion
18 of the `PMIx_Lookup_nb` API with the *status* indicating the success or failure of the request. Any
19 retrieved data will be returned in an array of `pmix_pdata_t` structs. The namespace and rank of
20 the process that provided each data element is also returned.

21 Note that the `pmix_pdata_t` structures will be released upon return from the callback function,
22 so the receiver must copy/protect the data prior to returning if it needs to be retained.

23 8.8 Retrieval rules for published data

24 The retrieval rules for published data primarily revolve around enforcing data access permissions
25 and range constraints. The datastore shall search its stored information for each specified key
26 according to the following precedence logic:

- 27 1. If the requester specified the range, then the search shall be constrained to data where the
28 publishing process falls within the specified range.
- 29 2. If the key of the stored information does not match the specified key, then the search will
30 continue.
- 31 3. If the requester's identifier does not fall within the range specified by the publisher, then the
32 search will continue.

1 4. If the publisher specified access permissions, the effective UID and GID of the requester shall be
2 checked against those permissions, with the datastore rejecting the match if the requester fails to
3 meet the requirements.

4 5. If all of the above checks pass, then the value is added to the information that is to be returned.

5 The status returned by the datastore shall be set to:

- 6
 - **PMIX_SUCCESS** All data was found and is included in the returned information.
 - **PMIX_ERR_NOT_FOUND** None of the requested data could be found within a requester's range.
 - **PMIX_ERR_PARTIAL_SUCCESS** Some of the requested data was found. Only found data will be included in the returned information. Note that the specific reason for a particular piece of missing information (e.g., lack of permissions) cannot be communicated back to the requester in this situation.

12 • a non-zero PMIx error constant indicating a reason for the request's failure.

13 In the case where data was found and range restrictions were met for each specified key, but none of
14 the matching data could be returned due to lack of access permissions, the datastore must return the
15 **PMIX_ERR_NO_PERMISSIONS** error.

Advice to users

16 Note that duplicate keys are allowed to exist on different ranges, and that ranges do overlap each
17 other. Thus, if duplicate keys are published on overlapping ranges, it is possible for the datastore to
18 successfully find multiple responses for a given key should publisher and requester specify
19 sufficiently broad ranges. In this situation, the choice of resolving the duplication is left to the
20 datastore implementation - e.g., it may return the first value found in its search, or the value
21 corresponding to the most limited range of the found values, or it may choose to simply return an
22 error.

23 Users are advised to avoid this ambiguity by careful selection of key values and ranges - e.g., by
24 creating range-specific keys where necessary.

25 8.9 **PMIx_Unpublish**

26 Summary

27 Unpublish data posted by this process using the given keys.

1 **Format**

C

```
2      pmix_status_t  
3      PMIx_Unpublish(char **keys,  
4                    const pmix_info_t info[], size_t ninfo);
```

C

5 **IN keys**

6 NULL-terminated array of keys (array of strings)

7 **IN info**

8 Array of info structures (array of handles)

9 **IN ninfo**

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

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

▼----- Required Attributes -----▼

12 PMIx libraries are not required to directly support any attributes for this function. However, any
13 provided attributes must be passed to the host environment for processing, and the PMIx library is
14 required to add the **PMIX_USERID** and the **PMIX_GRP_ID** attributes of the client process that is
15 requesting the operation.

▼----- Optional Attributes -----▼

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

17 **PMIX_TIMEOUT "pmix.timeout" (int)**

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

21 **PMIX_RANGE "pmix.range" (pmix_data_range_t)**

22 Define constraints on the processes that can access the provided data. Only processes that
23 meet the constraints are allowed to access it.

24 **Description**

25 Unpublish data posted by this process using the given *keys*. The function will block until the data
26 has been removed by the server (i.e., it is safe to publish that key again within the specified range).
27 A value of **NULL** for the *keys* parameter instructs the server to remove all data published by this
28 process.

29 By default, the range is assumed to be **PMIX_RANGE_SESSION**. Changes to the range, and any
30 additional directives, can be provided in the *info* array.

1 8.10 PMIx_Unpublish_nb

2 Summary

3 Nonblocking version of [PMIx_Unpublish](#).

4 Format

```
5     pmix_status_t  
6     PMIx_Unpublish_nb(char **keys,  
7                         const pmix_info_t info[], size_t ninfo,  
8                         pmix_op_cbfunc_t cbfunc, void *cbdata);
```

9 **IN keys**
10 NULL-terminated array of keys (array of strings)

11 **IN info**
12 Array of info structures (array of handles)

13 **IN ninfo**
14 Number of elements in the *info* array (integer)

15 **IN cbfunc**
16 Callback function [pmix_op_cbfunc_t](#) (function reference)

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

19 Returns one of the following:

- 20 • [PMIX_SUCCESS](#), indicating that the request is being processed by the host environment - result
21 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
22 function prior to returning from the API.
- 23 • [PMIX_OPERATION_SUCCEEDED](#), indicating that the request was immediately processed and
24 returned *success* - the *cbfunc* will *not* be called.
- 25 • a PMIx error constant indicating either an error in the input or that the request was immediately
26 processed and failed - the *cbfunc* will *not* be called.

Required Attributes

27 PMIx libraries are not required to directly support any attributes for this function. However, any
28 provided attributes must be passed to the host environment for processing, and the PMIx library is
29 required to add the [PMIX_USERID](#) and the [PMIX_GRP_ID](#) attributes of the client process that is
30 requesting the operation.

Optional Attributes

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

2 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

6 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)

7 Define constraints on the processes that can access the provided data. Only processes that
8 meet the constraints are allowed to access it.

9 Description

10 Non-blocking form of the **PMIx_Unpublish** function. The callback function will be executed
11 once the server confirms removal of the specified data. The *info* array must be maintained until the
12 callback is provided.

CHAPTER 9

Event Notification

This chapter defines the PMIx event notification system. These interfaces are designed to support the reporting of events to/from clients and servers, and between library layers within a single process.

9.1 Notification and Management

PMIx event notification provides an asynchronous out-of-band mechanism for communicating events between application processes and/or elements of the SMS. Its uses span a wide range including fault notification, coordination between multiple programming libraries within a single process, and workflow orchestration for non-synchronous programming models. Events can be divided into two distinct classes:

- *Job-specific events* directly relate to a job executing within the session, such as a debugger attachment, process failure within a related job, or events generated by an application process. Events in this category are to be immediately delivered to the PMIx server library for relay to the related local processes.
- *Environment events* indirectly relate to a job but do not specifically target the job itself. This category includes SMS-generated events such as Error Check and Correction (ECC) errors, temperature excursions, and other non-job conditions that might directly affect a session's resources, but would never include an event generated by an application process. Note that although these do potentially impact the session's jobs, they are not directly tied to those jobs. Thus, events in this category are to be delivered to the PMIx server library only upon request.

Both SMS elements and applications can register for events of either type.

Advice to PMIx library implementers

Race conditions can cause the registration to come after events of possible interest (e.g., a memory ECC event that occurs after start of execution but prior to registration, or an application process generating an event prior to another process registering to receive it). SMS vendors are *requested* to cache environment events for some time to mitigate this situation, but are not *required* to do so. However, PMIx implementers are *required* to cache all events received by the PMIx server library and to deliver them to registering clients in the same order in which they were received

Advice to users

1 Applications must be aware that they may not receive environment events that occur prior to
2 registration, depending upon the capabilities of the host SMS.

3 The generator of an event can specify the *target range* for delivery of that event. Thus, the generator
4 can choose to limit notification to processes on the local node, processes within the same job as the
5 generator, processes within the same allocation, other threads within the same process, only the
6 SMS (i.e., not to any application processes), all application processes, or to a custom range based
7 on specific process identifiers. Only processes within the given range that register for the provided
8 event code will be notified. In addition, the generator can use attributes to direct that the event not
9 be delivered to any default event handlers, or to any multi-code handler (as defined below).

10 Event notifications provide the process identifier of the source of the event plus the event code and
11 any additional information provided by the generator. When an event notification is received by a
12 process, the registered handlers are scanned for their event code(s), with matching handlers
13 assembled into an *event chain* for servicing. Note that users can also specify a *source range* when
14 registering an event (using the same range designators described above) to further limit when they
15 are to be invoked. When assembled, PMIx event chains are ordered based on both the specificity of
16 the event handler and user directives at time of handler registration. By default, handlers are
17 grouped into three categories based on the number of event codes that can trigger the callback:

- 18 • *single-code* handlers are serviced first as they are the most specific. These are handlers that are
19 registered against one specific event code.
- 20 • *multi-code* handlers are serviced once all single-code handlers have completed. The handler will
21 be included in the chain upon receipt of an event matching any of the provided codes.
- 22 • *default* handlers are serviced once all multi-code handlers have completed. These handlers are
23 always included in the chain unless the generator specifically excludes them.

24 Users can specify the callback order of a handler within its category at the time of registration.
25 Ordering can be specified 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. Dereistration
36 of the declared *first* handler will re-open the position for subsequent assignment.

1 Similarly, one event handler can be declared as the *last* handler to be executed in the chain. This
2 handler will *always* be called after all other handlers have executed, regardless of category,
3 provided the incoming event matches both the specified range and event code. Note that this
4 handler will not be called if the chain is terminated by an earlier handler. Only one handler can be
5 designated as *last* — attempts to designate additional handlers as *last* will return an error.
6 Deregistration of the declared *last* handler will re-open the position for subsequent assignment.

Advice to users

7 Note that the *last* handler is called *after* all registered default handlers that match the specified
8 range of the incoming event unless a handler prior to it terminates the chain. Thus, if the application
9 intends to define a *last* handler, it should ensure that no default handler aborts the process before it.

10 Upon completing its work and prior to returning, each handler *must* call the event handler
11 completion function provided when it was invoked (including a status code plus any information to
12 be passed to later handlers) so that the chain can continue being progressed. PMIx automatically
13 aggregates the status and any results of each handler (as provided in the completion callback) with
14 status from all prior handlers so that each step in the chain has full knowledge of what preceded it.
15 An event handler can terminate all further progress along the chain by passing the
16 [PMIX_EVENT_ACTION_COMPLETE](#) status to the completion callback function.

17 9.1.1 Events versus status constants

18 Return status constants (see Section 3.1.1) represent values that can be returned from or passed into
19 PMIx APIs. These are distinct from PMIx events in that they are not values that can be registered
20 against event handlers. In general, the two types of constants are distinguished by inclusion of an
21 "ERR" in the name of error constants versus an "EVENT" in events, though there are exceptions
22 (e.g, the [PMIX_SUCCESS](#) constant).

23 9.1.2 [PMIx_Register_event_handler](#)

24 Summary

25 Register an event handler.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,  
4                               pmix_info_t info[], size_t ninfo,  
5                               pmix_notification_fn_t evhdlr,  
6                               pmix_hdlr_reg_cbfunc_t cbfunc,  
7                               void *cbdata);
```

C

8 IN **codes**9 Array of status codes (array of [pmix_status_t](#))10 IN **ncodes**11 Number of elements in the *codes* array ([size_t](#))12 IN **info**

13 Array of info structures (array of handles)

14 IN **ninfo**15 Number of elements in the *info* array ([size_t](#))16 IN **evhdlr**17 Event handler to be called [pmix_notification_fn_t](#) (function reference)18 IN **cbfunc**19 Callback function [pmix_hdlr_reg_cbfunc_t](#) (function reference)20 IN **cbdata**21 Data to be passed to the *cbfunc* callback function (memory reference)

22 If *cbfunc* is **NULL**, the function call will be treated as a *blocking* call. In this case, the returned
23 status will be either (a) the event handler reference identifier if the value is greater than or equal to
24 zero, or (b) a negative error code indicative of the reason for the failure.

25 If the *cbfunc* is non-**NULL**, the function call will be treated as a *non-blocking* call and will return
26 the following:

- 27 • **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
28 callback function will be executed upon completion of the operation. Note that the library must
29 not invoke the callback function prior to returning from the API. The result of the registration
30 operation shall be returned in the provided callback function along with the assigned event
31 handler identifier.
- 32 • **PMIX_ERR_EVENT_REGISTRATION** indicating that the registration has failed for an
33 undetermined reason.
- 34 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this
35 case, the provided callback function will not be executed.

36 The callback function must not be executed prior to returning from the API, and no events
37 corresponding to this registration may be delivered prior to the completion of the registration
38 callback function (*cbfunc*).

Required Attributes

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

1 **PMIX_EVENT_HDLR_NAME** "pmix.evname" (**char***)
2 String name identifying this handler.
3
4 **PMIX_EVENT_HDLR_FIRST** "pmix.evfist" (**bool**)
5 Invoke this event handler before any other handlers.
6
7 **PMIX_EVENT_HDLR_LAST** "pmix.evlst" (**bool**)
8 Invoke this event handler after all other handlers have been called.
9
10 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY** "pmix.evfistcat" (**bool**)
11 Invoke this event handler before any other handlers in this category.
12
13 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY** "pmix.evlstcat" (**bool**)
14 Invoke this event handler after all other handlers in this category have been called.
15
16 **PMIX_EVENT_HDLR_BEFORE** "pmix.evbefore" (**char***)
17 Put this event handler immediately before the one specified in the (**char***) value.
18
19 **PMIX_EVENT_HDLR_AFTER** "pmix.evafter" (**char***)
20 Put this event handler immediately after the one specified in the (**char***) value.
21
22 **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (**bool**)
23 Prepend this handler to the precedence list within its category.
24
25 **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (**bool**)
26 Append this handler to the precedence list within its category.
27
28
29 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrang" (**pmix_data_array_t***)
30 Array of **pmix_proc_t** defining range of event notification.
31
32 **PMIX_RANGE** "pmix.range" (**pmix_data_range_t**)
33 Define constraints on the processes that can access the provided data. Only processes that
34 meet the constraints are allowed to access it.

Host environments that implement support for PMIx event notification are required to support the following attributes when registering handlers - these attributes are used to direct that the handler should be invoked only when the event affects the indicated process(es):

35 **PMIX_EVENT_AFFECTED_PROC** "pmix.evproc" (**pmix_proc_t**)
36 The single process that was affected.
37
38 **PMIX_EVENT_AFFECTED_PROCS** "pmix.evaaffected" (**pmix_data_array_t***)

1 Array of `pmix_proc_t` defining affected processes.

2 **Description**

3 Register an event handler to report events. Note that the codes being registered do *not* need to be
4 PMIx error constants — any integer value can be registered. This allows for registration of
5 non-PMIx events such as those defined by a particular SMS vendor or by an application itself.

6 **Advice to users**

7 In order to avoid potential conflicts, users are advised to only define codes that lie outside the range
8 of the PMIx standard's error codes. Thus, SMS vendors and application developers should
9 constrain their definitions to positive values or negative values beyond the
`PMIX_EXTERNAL_ERR_BASE` boundary.

10 **Advice to users**

11 As previously stated, upon completing its work, and prior to returning, each handler *must* call the
12 event handler completion function provided when it was invoked (including a status code plus any
13 information to be passed to later handlers) so that the chain can continue being progressed. An
14 event handler can terminate all further progress along the chain by passing the
15 `PMIX_EVENT_ACTION_COMPLETE` status to the completion callback function. Note that the
16 parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once
17 the completion function has been called - thus, any information in the incoming parameters that
will be referenced following the call to the completion function must be copied.

18 **9.1.3 Event registration constants**

19 `PMIX_ERR_EVENT_REGISTRATION` Error in event registration.

20 **9.1.4 System events**

21 `PMIX_EVENT_SYS_BASE` Mark the beginning of a dedicated range of constants for system
22 event reporting.

23 `PMIX_EVENT_NODE_DOWN` A node has gone down - the identifier of the affected node will
24 be included in the notification.

25 `PMIX_EVENT_NODE_OFFLINE` A node has been marked as *offline* - the identifier of the
26 affected node will be included in the notification.

27 `PMIX_EVENT_SYS_OTHER` Mark the end of a dedicated range of constants for system event
28 reporting.

1 **Detect system event constant**
2 Test a given event constant to see if it falls within the dedicated range of constants for system event
3 reporting.

4 **PMIX_SYSTEM_EVENT (a)**

5 **IN a**

6 Error constant to be checked ([pmix_status_t](#))

7 Returns **true** if the provided values falls within the dedicated range of events for system event
8 reporting.

9 **9.1.5 Event handler registration and notification attributes**

10 Attributes to support event registration and notification.

11 **PMIX_EVENT_HDLR_NAME "pmix.evname" (char*)**

12 String name identifying this handler.

13 **PMIX_EVENT_HDLR_FIRST "pmix.evfirst" (bool)**

14 Invoke this event handler before any other handlers.

15 **PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool)**

16 Invoke this event handler after all other handlers have been called.

17 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool)**

18 Invoke this event handler before any other handlers in this category.

19 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)**

20 Invoke this event handler after all other handlers in this category have been called.

21 **PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)**

22 Put this event handler immediately before the one specified in the **(char*)** value.

23 **PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)**

24 Put this event handler immediately after the one specified in the **(char*)** value.

25 **PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)**

26 Prepend this handler to the precedence list within its category.

27 **PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool)**

28 Append this handler to the precedence list within its category.

29 **PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)**

30 Array of [pmix_proc_t](#) defining range of event notification.

31 **PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)**

32 The single process that was affected.

33 **PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)**

34 Array of [pmix_proc_t](#) defining affected processes.

35 **PMIX_EVENT_NON_DEFAULT "pmix.evnonddef" (bool)**

36 Event is not to be delivered to default event handlers.

37 **PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)**

1 Object to be returned whenever the registered callback function **cbfunc** is invoked. The
2 object will only be returned to the process that registered it.
3 **PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)**
4 Instruct the PMIx server not to cache the event.
5 **PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)**
6 PMIx server that sourced the event.
7 **PMIX_EVENT_TEXT_MESSAGE "pmix.evtext" (char*)**
8 Text message suitable for output by recipient - e.g., describing the cause of the event.
9 **PMIX_EVENT_TIMESTAMP "pmix.evtstamp" (time_t)**
10 System time when the associated event occurred.

11 9.1.5.1 Fault tolerance event attributes

12 The following attributes may be used by the host environment when providing an event notification
13 as qualifiers indicating the action it intends to take in response to the event:

14 **PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)**
15 The RM intends to terminate this session.
16 **PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)**
17 The RM intends to terminate this job.
18 **PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)**
19 The RM intends to terminate all processes on this node.
20 **PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool)**
21 The RM intends to terminate just this process.
22 **PMIX_EVENT_ACTION_TIMEOUT "pmix.evttimeout" (int)**
23 The time in seconds before the RM will execute the indicated operation.

24 9.1.5.2 Hybrid programming event attributes

25 The following attributes may be used by programming models to coordinate their use of common
26 resources within a process in conjunction with the **PMIX_OPENMP_PARALLEL_ENTERED** event:

27 **PMIX_MODEL_PHASE_NAME "pmix.mdl.phase" (char*)**
28 User-assigned name for a phase in the application execution (e.g., “cfd reduction”).
29 **PMIX_MODEL_PHASE_TYPE "pmix.mdl.ptype" (char*)**
30 Type of phase being executed (e.g., “matrix multiply”).

31 9.1.6 Notification Function

32 Summary

33 The **pmix_notification_fn_t** is called by PMIx to deliver notification of an event.

34 Advice to users

35 The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but
36 different signature than the v2.0 Standard described below. The *ad hoc* v1.0 version was removed
 from the v2.0 Standard is not included in this document to avoid confusion.

```
1  typedef void (*pmix_notification_fn_t)
2      (size_t evhdlr_registration_id,
3       pmix_status_t status,
4       const pmix_proc_t *source,
5       pmix_info_t info[], size_t ninfo,
6       pmix_info_t results[], size_t nresults,
7       pmix_event_notification_cbfunc_fn_t cbfunc,
8       void *cbdata);
```

IN **evhdlr_registration_id**
Registration number of the handler being called (**size_t**)
IN **status**
Status associated with the operation (**pmix_status_t**)
IN **source**
Identifier of the process that generated the event (**pmix_proc_t**). If the source is the SMS, then the nspace will be empty and the rank will be PMIX_RANK_UNDEF
IN **info**
Information describing the event (**pmix_info_t**). This argument will be NULL if no additional information was provided by the event generator.
IN **ninfo**
Number of elements in the info array (**size_t**)
IN **results**
Aggregated results from prior event handlers servicing this event (**pmix_info_t**). This argument will be **NULL** if this is the first handler servicing the event, or if no prior handlers provided results.
IN **nresults**
Number of elements in the results array (**size_t**)
IN **cbfunc**
pmix_event_notification_cbfunc_fn_t callback function to be executed upon completion of the handler's operation and prior to handler return (function reference).
IN **cbdata**
Callback data to be passed to cbfunc (memory reference)

32 **Description**

33 Note that different RMs may provide differing levels of support for event notification to application
34 processes. Thus, the *info* array may be **NULL** or may contain detailed information of the event. It is
35 the responsibility of the application to parse any provided info array for defined key-values if it so
36 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.

9.1.7 PMIx_Deregister_event_handler

Summary

Deregister an event handler.

Format

0

```
pmix_status_t  
PMIx_Deregister_event_handler(size_t evhdlr_ref,  
                                pmix_op_cbfunc_t cbfunc,  
                                void *cbdata);
```

0

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 and return one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
 - **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
 - a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

The returned status code will be one of the following:

- **PMIX_SUCCESS** The event handler was successfully deregistered.
 - **PMIX_ERR_BAD_PARAM** The provided *evhdlr_ref* was unrecognized.
 - **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification.

Description

Deregister an event handler. Note that no events corresponding to the referenced registration may be delivered following completion of the deregistration operation (either return from the API with `PMIX_OPERATION_SUCCEEDED` or execution of the `cbfunc`).

PMIx Notify event

Summary

Report an event for notification via any registered event handler.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Notify_event(pmix_status_t status,  
4                     const pmix_proc_t *source,  
5                     pmix_data_range_t range,  
6                     pmix_info_t info[], size_t ninfo,  
7                     pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

8 IN **status**

9 Status code of the event (**pmix_status_t**)

10 IN **source**

11 Pointer to a **pmix_proc_t** identifying the original reporter of the event (handle)

12 IN **range**

13 Range across which this notification shall be delivered (**pmix_data_range_t**)

14 IN **info**

15 Array of **pmix_info_t** structures containing any further info provided by the originator of
16 the event (array of handles)

17 IN **ninfo**

18 Number of elements in the *info* array (**size_t**)

19 IN **cbfunc**

20 Callback function to be executed upon completion of operation **pmix_op_cbfunc_t**
21 (function reference)

22 IN **cbdata**

23 Data to be passed to the *cbfunc* callback function (memory reference)

24 If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation
25 returned in the status code.

26 If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the
27 following:

- 28 • **PMIX_SUCCESS** The notification request is valid and is being processed. The callback function
29 will be called when the process-local operation is complete and will provide the resulting status
30 of that operation. Note that this does *not* reflect the success or failure of delivering the event to
31 any recipients. The callback function must not be executed 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_BAD_PARAM** The request contains at least one incorrect entry that prevents it from
35 being processed. The callback function will *not* be called.

- **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification, or in the case of a PMIx server calling the API, the range extended beyond the local node and the host SMS environment does not support event notification. The callback function will *not* be called.

Required Attributes

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

PMIX_EVENT_NON_DEFAULT "pmix.evnondef" (bool)

Event is not to be delivered to default event handlers.

PMIX_EVENT_CUSTOM_RANGE "pmix.evrangle" (pmix_data_array_t*)

Array of `pmix_proc_t` defining range of event notification.

PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)

Instruct the PMIx server not to cache the event.

PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)

PMIx server that sourced the event.

PMIX_EVENT_TEXT_MESSAGE "pmix.evtext" (char*)

Text message suitable for output by recipient - e.g., describing the cause of the event.

Host environments that implement support for PMIx event notification are required to provide the following attributes for all events generated by the environment:

PMIX EVENT AFFECTED PROC "pmix.evproc" (pmix proc t)

The single process that was affected.

PMIX EVENT AFFECTED PROCS "pmix.evaaffected" (pmix data array t*)

Array of **pmix proc t** defining affected processes.

Optional Attributes

Host environments that support PMIx event notification may offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes may optionally be included to indicate the host environment's intended response to the event:

PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)

The RM intends to terminate this session.

PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)

The RM intends to terminate this job.

PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)

The RM intends to terminate all processes on this node.

PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool)

1 The RM intends to terminate just this process.

2 **PMIX_EVENT_ACTION_TIMEOUT** "pmix.evttimeout" (int)

3 The time in seconds before the RM will execute the indicated operation.



4 **Description**

5 Report an event for notification via any registered event handler. This function can be called by any
6 PMIx process, including application processes, PMIx servers, and SMS elements. The PMIx server
7 calls this API to report events it detected itself so that the host SMS daemon distribute and handle
8 them, and to pass events given to it by its host down to any attached client processes for processing.
9 Examples might include notification of the failure of another process, detection of an impending
10 node failure due to rising temperatures, or an intent to preempt the application. Events may be
11 locally generated or come from anywhere in the system.

12 Host SMS daemons call the API to pass events down to its embedded PMIx server both for
13 transmittal to local client processes and for the host's own internal processing where the host has
14 registered its own event handlers. The PMIx server library is not allowed to echo any event given to
15 it by its host via this API back to the host through the **pmix_server_notify_event_fn_t**
16 server module function. The host is required to deliver the event to all PMIx servers where the
17 targeted processes either are currently running, or (if they haven't started yet) might be running at
18 some point in the future as the events are required to be cached by the PMIx server library.

19 Client application processes can call this function to notify the SMS and/or other application
20 processes of an event it encountered. Note that processes are not constrained to report status values
21 defined in the official PMIx standard — any integer value can be used. Thus, applications are free
22 to define their own internal events and use the notification system for their own internal purposes.



Advice to users

23 The callback function will be called upon completion of the **notify_event** function's actions.
24 At that time, any messages required for executing the operation (e.g., to send the notification to the
25 local PMIx server) will have been queued, but may not yet have been transmitted. The caller is
26 required to maintain the input data until the callback function has been executed — the sole purpose
27 of the callback function is to indicate when the input data is no longer required.



9.1.9 Notification Handler Completion Callback Function

Summary

The `pmix_event_notification_cbfunc_fn_t` is called by event handlers to indicate 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);
```

IN **status**
Status returned by the event handler's operation (`pmix_status_t`)
IN **results**
Results from this event handler's operation on the event (`pmix_info_t`)
IN **nresults**
Number of elements in the results array (`size_t`)
IN **cbfunc**
`pmix_op_cbfunc_t` function to be executed when PMIx completes processing the
callback (function reference)
IN **thiscbdata**
Callback data that was passed in to the handler (memory reference)
IN **cbdata**
Callback data to be returned when PMIx executes cbfunc (memory reference)

Description

Define a callback by which an event handler can notify the PMIx library that it has completed its response to the notification. The handler is *required* to execute this callback so the library can determine if additional handlers need to be called. The handler shall return

`PMIX_EVENT_ACTION_COMPLETE` if no further action is required. The return status of each event handler and any returned `pmix_info_t` structures will be added to the *results* array of `pmix_info_t` passed to any subsequent event handlers to help guide their operation.

If non-`NULL`, the provided callback function will be called to allow the event handler to release the provided info array and execute any other required cleanup operations.

9.1.9.1 Completion Callback Function Status Codes

The following status code may be returned indicating various actions taken by other event handlers.

<code>PMIX_EVENT_NO_ACTION_TAKEN</code>	Event handler: No action taken.
<code>PMIX_EVENT_PARTIAL_ACTION_TAKEN</code>	Event handler: Partial action taken.
<code>PMIX_EVENT_ACTION_DEFERRED</code>	Event handler: Action deferred.
<code>PMIX_EVENT_ACTION_COMPLETE</code>	Event handler: Action complete.

CHAPTER 10

Data Packing and Unpacking

1 PMIx intentionally does not include support for internode communications in the standard, instead
2 relying on its host SMS environment to transfer any needed data and/or requests between nodes.
3 These operations frequently involve PMIx-defined public data structures that include binary data.
4 Many HPC clusters are homogeneous, and so transferring the structures can be done rather simply.
5 However, greater effort is required in heterogeneous environments to ensure binary data is correctly
6 transferred. PMIx buffer manipulation functions are provided for this purpose via standardized
7 interfaces to ease adoption.

8 10.1 Data Buffer Type

9 The `pmix_data_buffer_t` structure describes a data buffer used for packing and unpacking.

PMIx v2.0

C

```
10     typedef struct pmix_data_buffer {
11         /** Start of my memory */
12         char *base_ptr;
13         /** Where the next data will be packed to
14             (within the allocated memory starting
15             at base_ptr) */
16         char *pack_ptr;
17         /** Where the next data will be unpacked
18             from (within the allocated memory
19             starting as base_ptr) */
20         char *unpack_ptr;
21         /** Number of bytes allocated (starting
22             at base_ptr) */
23         size_t bytes_allocated;
24         /** Number of bytes used by the buffer
25             (i.e., amount of data -- including
26             overhead -- packed in the buffer) */
27         size_t bytes_used;
28     } pmix_data_buffer_t;
```

C

10.2 Support Macros

PMIx provides a set of convenience macros for creating, initiating, and releasing data buffers.

PMIX_DATA_BUFFER_CREATE

Allocate memory for a `pmix_data_buffer_t` object and initialize it. This macro uses `calloc` to allocate memory for the buffer and initialize all fields in it

PMIx v2.0

```
6 PMIX_DATA_BUFFER_CREATE(buffer);
```

C

C

OUT buffer

Variable to be assigned the pointer to the allocated `pmix_data_buffer_t` (handle)

PMIX_DATA_BUFFER_RELEASE

Free a `pmix_data_buffer_t` object and the data it contains. Free's the data contained in the buffer, and then free's the buffer itself

PMIx v2.0

```
12 PMIX_DATA_BUFFER_RELEASE(buffer);
```

C

C

IN buffer

Pointer to the `pmix_data_buffer_t` to be released (handle)

PMIX_DATA_BUFFER_CONSTRUCT

Initialize a statically declared `pmix_data_buffer_t` object.

PMIx v2.0

```
17 PMIX_DATA_BUFFER_CONSTRUCT(buffer);
```

C

C

IN buffer

Pointer to the allocated `pmix_data_buffer_t` that is to be initialized (handle)

PMIX_DATA_BUFFER_DESTRUCT

Release the data contained in a `pmix_data_buffer_t` object.

PMIx v2.0

```
22 PMIX_DATA_BUFFER_DESTRUCT(buffer);
```

C

C

IN buffer

Pointer to the `pmix_data_buffer_t` whose data is to be released (handle)

```
1 PMIX_DATA_BUFFER_LOAD  
2 Load a blob into a pmix_data_buffer_t object. Load the given data into the provided  
3 pmix_data_buffer_t object, usually done in preparation for unpacking the provided data.  
4 Note that the data is not copied into the buffer - thus, the blob must not be released until after  
5 operations on the buffer have completed.
```

PMIx v2.0

C

```
6 PMIX_DATA_BUFFER_LOAD(buffer, data, size);
```

C

```
7 IN buffer  
8 Pointer to a pre-allocated pmix_data_buffer_t (handle)  
9 IN data  
10 Pointer to a blob (char*)  
11 IN size  
12 Number of bytes in the blob size_t
```

```
13 PMIX_DATA_BUFFER_UNLOAD
```

```
14 Unload the data from a pmix_data_buffer_t object. Extract the data in a buffer, assigning the  
15 pointer to the data (and the number of bytes in the blob) to the provided variables, usually done to  
16 transmit the blob to a remote process for unpacking. The buffer's internal pointer will be set to  
17 NULL to protect the data upon buffer destruct or release - thus, the user is responsible for releasing  
18 the blob when done with it.
```

PMIx v2.0

C

```
19 PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

C

```
20 IN buffer  
21 Pointer to the pmix_data_buffer_t whose data is to be extracted (handle)  
22 OUT data  
23 Variable to be assigned the pointer to the extracted blob (void*)  
24 OUT size  
25 Variable to be assigned the number of bytes in the blob size_t
```

26 10.3 General Routines

27 The following routines are provided to support internode transfers in heterogeneous environments.

28 10.3.1 PMIx_Data_pack

29 Summary

30 Pack one or more values of a specified type into a buffer, usually for transmission to another process.

1 **Format**

C

```
2       pmix_status_t  
3       PMIx_Data_Pack(const pmix_proc_t *target,  
4                     pmix_data_buffer_t *buffer,  
5                     void *src, int32_t num_vals,  
6                     pmix_data_type_t type);
```

C

7 **IN target**

8 Pointer to a `pmix_proc_t` containing the nspace/rank of the process that will be unpacking
9 the final buffer. A NULL value may be used to indicate that the target is based on the same
10 PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

11 **IN buffer**

12 Pointer to a `pmix_data_buffer_t` where the packed data is to be stored (handle)

13 **IN src**

14 Pointer to a location where the data resides. Strings are to be passed as (char **)— i.e., the
15 caller must pass the address of the pointer to the string as the (void*). This allows the caller to
16 pass multiple strings in a single call. (memory reference)

17 **IN num_vals**

18 Number of elements pointed to by the `src` pointer. A string value is counted as a single value
19 regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,
20 string arrays) should be contiguous, although the data pointed to need not be contiguous
21 across array entries. (`int32_t`)

22 **IN type**

23 The type of the data to be packed (`pmix_data_type_t`)

24 Returns one of the following:

25 **PMIX_SUCCESS** The data has been packed as requested

26 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

27 **PMIX_ERR_BAD_PARAM** The provided buffer or src is **NULL**

28 **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this
29 implementation

30 **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation

31 **PMIX_ERROR** General error

32 **Description**

33 The pack function packs one or more values of a specified type into the specified buffer. The buffer
34 must have already been initialized via the `PMIX_DATA_BUFFER_CREATE` or
35 `PMIX_DATA_BUFFER_CONSTRUCT` macros — otherwise, `PMIx_Data_Pack` will return an
36 error. Providing an unsupported type flag will likewise be reported as an error.

37 Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may
38 lose precision when unpacked by a non-homogeneous recipient. The `PMIx_Data_Pack` function

1 will do its best to deal with heterogeneity issues between the packer and unpacker in such cases.
2 Sending a number larger than can be handled by the recipient will return an error code (generated
3 upon unpacking) — the error cannot be detected during packing.

4 The namespace of the intended recipient of the packed buffer (i.e., the process that will be
5 unpacking it) is used solely to resolve any data type differences between PMIx versions. The
6 recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx
7 library is aware of the version the recipient is using. Note that all processes in a given namespace
8 are *required* to use the same PMIx version — thus, the caller must only know at least one process
9 from the target’s namespace.

10 10.3.2 PMIx_Data_unpack

11 Summary

12 Unpack values from a [pmix_data_buffer_t](#)

13 Format

```
14 pmix_status_t
15 PMIx_Data_unpack(const pmix_proc_t *source,
16                     pmix_data_buffer_t *buffer, void *dest,
17                     int32_t *max_num_values,
18                     pmix_data_type_t type);
```

20 IN source

21 Pointer to a [pmix_proc_t](#) structure containing the nspace/rank of the process that packed
22 the provided buffer. A NULL value may be used to indicate that the source is based on the
23 same PMIx version as the caller. Note that only the source’s nspace is relevant. (handle)

24 IN buffer

25 A pointer to the buffer from which the value will be extracted. (handle)

26 INOUT dest

27 A pointer to the memory location into which the data is to be stored. Note that these values
28 will be stored contiguously in memory. For strings, this pointer must be to (char**) to provide
29 a means of supporting multiple string operations. The unpack function will allocate memory
30 for each string in the array - the caller must only provide adequate memory for the array of
31 pointers. ([void*](#))

32 INOUT max_num_values

33 The number of values to be unpacked — upon completion, the parameter will be set to the
34 actual number of values unpacked. In most cases, this should match the maximum number
35 provided in the parameters — but in no case will it exceed the value of this parameter. Note
36 that unpacking fewer values than are actually available will leave the buffer in an unpackable
37 state — the function will return an error code to warn of this condition. ([int32_t](#))

1 **IN type**
2 The type of the data to be unpacked — must be one of the PMIx defined data types
3 ([pmix_data_type_t](#))
4 Returns one of the following:
5 **PMIX_SUCCESS** The data has been unpacked as requested
6 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.
7 **PMIX_ERR_BAD_PARAM** The provided buffer or dest is **NULL**
8 **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this
9 implementation
10 **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation
11 **PMIX_ERROR** General error

12 **Description**
13 The unpack function unpacks the next value (or values) of a specified type from the given buffer.
14 The buffer must have already been initialized via an **PMIX_DATA_BUFFER_CREATE** or
15 **PMIX_DATA_BUFFER_CONSTRUCT** call (and assumedly filled with some data) — otherwise, the
16 unpack_value function will return an error. Providing an unsupported type flag will likewise be
17 reported as an error, as will specifying a data type that *does not* match the type of the next item in
18 the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an
19 error.

20 Note that it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper
21 variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte
22 field in a string array that so happens to have a value that matches the specified data type flag).
23 Therefore, the data type error check is *not* completely safe.

24 Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer.
25 It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the
26 unpack_ptr.

27 Warning: The caller is responsible for providing adequate memory storage for the requested data.
28 The user must provide a parameter indicating the maximum number of values that can be unpacked
29 into the allocated memory. If more values exist in the buffer than can fit into the memory storage,
30 then the function will unpack what it can fit into that location and return an error code indicating
31 that the buffer was only partially unpacked.

32 Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may
33 lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with
34 heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than
35 can be handled by the recipient will return an error code generated upon unpacking — these errors
36 cannot be detected during packing.

37 The namespace of the process that packed the buffer is used solely to resolve any data type
38 differences between PMIx versions. The packer must, therefore, be known to the user prior to
39 calling the pack function so that the PMIx library is aware of the version the packer is using. Note

1 that all processes in a given namespace are *required* to use the same PMIx version — thus, the
2 caller must only know at least one process from the packer's namespace.

3 10.3.3 PMIx_Data_copy

4 Summary

5 Copy a data value from one location to another.

6 Format

```
7 pmix_status_t
8 PMIx_Data_copy(void **dest, void *src,
9                 pmix_data_type_t type);
```

10 IN dest

11 The address of a pointer into which the address of the resulting data is to be stored. (**void****)

12 IN src

13 A pointer to the memory location from which the data is to be copied (handle)

14 IN type

15 The type of the data to be copied — must be one of the PMIx defined data types.

16 ([pmix_data_type_t](#))

17 Returns one of the following:

18 [PMIX_SUCCESS](#) The data has been copied as requested

19 [PMIX_ERR_NOT_SUPPORTED](#) The PMIx implementation does not support this function.

20 [PMIX_ERR_BAD_PARAM](#) The provided src or dest is **NULL**

21 [PMIX_ERR_UNKNOWN_DATA_TYPE](#) The specified data type is not known to this
22 implementation

23 [PMIX_ERR_OUT_OF_RESOURCE](#) Not enough memory to support the operation

24 [PMIX_ERROR](#) General error

25 Description

26 Since registered data types can be complex structures, the system needs some way to know how to
27 copy the data from one location to another (e.g., for storage in the registry). This function, which
28 can call other copy functions to build up complex data types, defines the method for making a copy
29 of the specified data type.

30 10.3.4 PMIx_Data_print

31 Summary

32 Pretty-print a data value.

1 **Format**

C

```
2      pmix_status_t  
3      PMIx_Data_print(char **output, char *prefix,  
4                    void *src, pmix_data_type_t type);
```

C

5 **IN output**

6 The address of a pointer into which the address of the resulting output is to be stored.
7 (**char****)

8 **IN prefix**

9 String to be prepended to the resulting output (**char***)

10 **IN src**

11 A pointer to the memory location of the data value to be printed (handle)

12 **IN type**

13 The type of the data value to be printed — must be one of the PMIx defined data types.
14 (**pmix_data_type_t**)

15 Returns one of the following:

16 **PMIX_SUCCESS** The data has been printed as requested

17 **PMIX_ERR_BAD_PARAM** The provided data type is not recognized.

18 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

19 **Description**

20 Since registered data types can be complex structures, the system needs some way to know how to
21 print them (i.e., convert them to a string representation). Primarily for debug purposes.

22 **10.3.5 PMIx_Data_copy_payload**

23 **Summary**

24 Copy a payload from one buffer to another

25 *PMIx v2.0* **Format**

C

```
26    pmix_status_t  
27    PMIx_Data_copy_payload(pmix_data_buffer_t *dest,  
28                        pmix_data_buffer_t *src);
```

C

29 **IN dest**

30 Pointer to the destination **pmix_data_buffer_t** (handle)

31 **IN src**

32 Pointer to the source **pmix_data_buffer_t** (handle)

33 Returns one of the following:

1 **PMIX_SUCCESS** The data has been copied as requested
2 **PMIX_ERR_BAD_PARAM** The src and dest **pmix_data_buffer_t** types do not match
3 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

4 **Description**

5 This function will append a copy of the payload in one buffer into another buffer. Note that this is
6 not a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing
7 payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

8 **10.3.6 PMIx_Data_load**

9 **Summary**

10 Load a buffer with the provided payload

11 **Format**

12 **pmix_status_t**
13 **PMIx_Data_load**(**pmix_data_buffer_t** ***dest**,
14 **pmix_byte_object_t** ***src**);

15 **IN dest**

16 Pointer to the destination **pmix_data_buffer_t** (handle)

17 **IN src**

18 Pointer to the source **pmix_byte_object_t** (handle)

19 Returns one of the following:

20 **PMIX_SUCCESS** The data has been loaded as requested

21 **PMIX_ERR_BAD_PARAM** The *dest* structure pointer is **NULL**

22 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

23 **Description**

24 The load function allows the caller to transfer the contents of the *src* **pmix_byte_object_t** to
25 the *dest* target buffer. If a payload already exists in the buffer, the function will "free" the existing
26 data to release it, and then replace the data payload with the one provided by the caller.

27 **Advice to users**

28 The buffer must be allocated or constructed in advance - failing to do so will cause the load
function to return an error code.

29 The caller is responsible for pre-packing the provided payload. For example, the load function
30 cannot convert to network byte order any data contained in the provided payload.

1 **10.3.7 PMIx_Data_unload**

2 **Summary**

3 Unload a buffer into a byte object

4 **Format**

```
5     pmix_status_t  
6     PMIx_Data_unload(pmix_data_buffer_t *src,  
7                         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 one of the following:

13 `PMIX_SUCCESS` The data has been copied as requested

14 `PMIX_ERR_BAD_PARAM` The destination and/or source pointer is `NULL`

15 `PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

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 **10.3.8 PMIx_Data_compress**

26 **Summary**

27 Perform a lossless compression on the provided data

1 **Format**

C

2 **bool**
3 **PMIx_Data_compress**(**const uint8_t *inbytes**, **size_t size**,
4 **uint8_t **outbytes**, **size_t *nbytes**);

5 **IN inbytes**

6 Pointer to the source data (handle)

7 **IN size**

8 Number of bytes in the source data region (**size_t**)

9 **OUT outbytes**

10 Address where the pointer to the compressed data region is to be returned (handle)

11 **OUT nbytes**

12 Address where the number of bytes in the compressed data region is to be returned (handle)

13 Returns one of the following:

14 • **True** The data has been compressed as requested

15 • **False** The data has not been compressed

16 **Description**

17 Compress the provided data block. Destination memory will be allocated if operation is
18 successfully concluded. Caller is responsible for release of the allocated region. The input data
19 block will remain unaltered.

20 Note: the compress function will return **False** if the operation would not result in a smaller data
21 block.

22 **10.3.9 PMIx_Data_decompress**

23 **Summary**

24 Decompress the provided data

25 **Format**

PMIx v4.1

C

```
1  bool
2  PMIx_Data_decompress(const uint8_t *inbytes, size_t size,
3                        uint8_t **outbytes, size_t *nbytes,);
4  OUT outbytes
5  Address where the pointer to the decompressed data region is to be returned (handle)
6  OUT nbytes
7  Address where the number of bytes in the decompressed data region is to be returned (handle)
8  IN inbytes
9  Pointer to the source data (handle)
10 IN size
11 Number of bytes in the source data region (size_t)
12 Returns one of the following:
13 • True The data has been decompressed as requested
14 • False The data has not been decompressed
15 Description
16 Decompress the provided data block. Destination memory will be allocated if operation is
17 successfully concluded. Caller is responsible for release of the allocated region. The input data
18 block will remain unaltered.
19 Only data compressed by the PMIx\_Data\_compress API can be decompressed by this
20 function. Passing data that has not been compressed by PMIx\_Data\_compress will lead to
21 unexpected and potentially catastrophic results.
```

CHAPTER 11

Process Management

This chapter defines functionality processes can use to abort processes, spawn processes, and determine the relative locality of local processes.

11.1 Abort

PMIx provides a dedicated API by which an application can request that specified processes be aborted by the system.

11.1.1 PMIx_Abort

Summary

Abort the specified processes

Format

```
pmix_status_t  
PMIx_Abort(int status, const char msg[],  
            pmix_proc_t procs[], size_t nprocs)
```

IN **status**
Error code to return to invoking environment (integer)
IN **msg**
String message to be returned to user (string)
IN **procs**
Array of **pmix_proc_t** structures (array of handles)
IN **nprocs**
Number of elements in the *procs* array (integer)

Returns one of the following:

- **PMIX_SUCCESS** if the operation was successfully completed. Note that the function shall not return in this situation if the caller's own process was included in the request.
- **PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED** if the PMIx 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.
- a PMIx error constant indicating an error in the request.

1 **Description**

2 Request that the host resource manager print the provided message and abort the provided array of
3 *procs*. A Unix or POSIX environment should handle the provided status as a return error code from
4 the main program that launched the application. A **NULL** for the *procs* array indicates that all
5 processes in the caller's namespace are to be aborted, including itself - this is the equivalent of
6 passing a **pmix_proc_t** array element containing the caller's namespace and a rank value of
7 **PMIX_RANK_WILDCARD**. While it is permitted for a caller to request abort of processes from
8 namespaces other than its own, not all environments will support such requests. Passing a **NULL**
9 *msg* parameter is allowed.

10 The function shall not return until the host environment has carried out the operation on the
11 specified processes. If the caller is included in the array of targets, then the function will not return
12 unless the host is unable to execute the operation.

▼ **Advice to users** ▼

13 The response to this request is somewhat dependent on the specific RM and its configuration (e.g.,
14 some resource managers will not abort the application if the provided status is zero unless
15 specifically configured to do so, some cannot abort subsets of processes in an application, and some
16 may not permit termination of processes outside of the caller's own namespace), and thus lies
17 outside the control of PMIx itself. However, the PMIx client library shall inform the RM of the
18 request that the specified *procs* be aborted, regardless of the value of the provided status.

19 Note that race conditions caused by multiple processes calling **PMIx_Abort** are left to the server
20 implementation to resolve with regard to which status is returned and what messages (if any) are
21 printed.

22

11.2 Process Creation

23 The **PMIx_Spawn** commands spawn new processes and/or applications in the PMIx universe.
24 This may include requests to extend the existing resource allocation or obtain a new one, depending
25 upon provided and supported attributes.

26

11.2.1 PMIx_Spawn

27 **Summary**

28 Spawn a new job.

1 **Format**

C

```
2      pmix_status_t  
3      PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,  
4                 const pmix_app_t apps[], size_t napps,  
5                 char nspace[])
```

C

6 **IN job_info**

7 Array of info structures (array of handles)

8 **IN ninfo**

9 Number of elements in the *job_info* array (integer)

10 **IN apps**

11 Array of **pmix_app_t** structures (array of handles)

12 **IN napps**

13 Number of elements in the *apps* array (integer)

14 **OUT nspace**

15 Namespace of the new job (string)

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

17 **Required Attributes**

18 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.

19 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:

21 **PMIX_WDIR "pmix.wdir" (char*)**

22 Working directory for spawned processes.

23 **PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)**

24 Set the current working directory to the session working directory assigned by the RM - can
25 be assigned to the entire job (by including attribute in the *job_info* array) or on a
26 per-application basis in the *info* array for each **pmix_app_t**.

27 **PMIX_PREFIX "pmix.prefix" (char*)**

28 Prefix to use for starting spawned processes - i.e., the directory where the executables can be
29 found.

30 **PMIX_HOST "pmix.host" (char*)**

31 Comma-delimited list of hosts to use for spawned processes.

32 **PMIX_HOSTFILE "pmix.hostfile" (char*)**

33 Hostfile to use for spawned processes.

Optional Attributes

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

1 **PMIX_ADD_HOSTFILE** "pmix.addhostfile" (**char***)
2 Hostfile containing hosts to add to existing allocation.
3
4 **PMIX_ADD_HOST** "pmix.addhost" (**char***)
5 Comma-delimited list of hosts to add to the allocation.
6 **PMIX_PRELOAD_BIN** "pmix.preloadbin" (**bool**)
7 Preload executables onto nodes prior to executing launch procedure.
8 **PMIX_PRELOAD_FILES** "pmix.loadfiles" (**char***)
9 Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.
10 **PMIX_PERSONALITY** "pmix.pers" (**char***)
11 Name of personality corresponding to programming model used by application - supported
12 values depend upon PMIx implementation.
13 **PMIX_DISPLAY_MAP** "pmix.dispmap" (**bool**)
14 Display process mapping upon spawn.
15 **PMIX_PPR** "pmix.ppr" (**char***)
16 Number of processes to spawn on each identified resource.
17 **PMIX_MAPBY** "pmix.mapby" (**char***)
18 Process mapping policy - when accessed using **PMIx_Get**, use the
19 **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the
20 provided namespace. Supported values are launcher specific.
21 **PMIX_RANKBY** "pmix.rankby" (**char***)
22 Process ranking policy - when accessed using **PMIx_Get**, use the
23 **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the
24 provided namespace. Supported values are launcher specific.
25 **PMIX_BINDTO** "pmix.bindto" (**char***)
26 Process binding policy - when accessed using **PMIx_Get**, use the
27 **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the
28 provided namespace. Supported values are launcher specific.
29 **PMIX_STDIN_TGT** "pmix.stdin" (**uint32_t**)
30 Spawned process rank that is to receive any forwarded **stdin**.
31 **PMIX_TAG_OUTPUT** "pmix.tagout" (**bool**)
32 Tag **stdout/stderr** with the identity of the source process - can be assigned to the entire
33 job (by including attribute in the *job_info* array) or on a per-application basis in the *info*
34 array for each **pmix_app_t**.
35 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (**bool**)

1 Timestamp output - can be assigned to the entire job (by including attribute in the *job_info*
2 array) or on a per-application basis in the *info* array for each **pmix_app_t**.

3 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)

4 Merge **stdout** and **stderr** streams - can be assigned to the entire job (by including
5 attribute in the *job_info* array) or on a per-application basis in the *info* array for each
6 **pmix_app_t**.

7 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)

8 Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be
9 assigned to the entire job (by including attribute in the *job_info* array) or on a per-application
10 basis in the *info* array for each **pmix_app_t**.

11 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)

12 Mark the **argv** with the rank of the process.

13 **PMIX_CPUS_PER_PROC** "pmix.cpuperproc" (uint32_t)

14 Number of PUs to assign to each rank - when accessed using **PMIx_Get**, use the
15 **PMIX_RANK_WILDCARD** value for the rank to discover the PUs/process assigned to the
16 provided namespace.

17 **PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)

18 Do not place processes on the head node.

19 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)

20 Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
21 node.

22 **PMIX_REPORT_BINDINGS** "pmix.repbind" (bool)

23 Report bindings of the individual processes.

24 **PMIX_CPU_LIST** "pmix.cpulist" (char*)

25 List of PUs to use for this job - when accessed using **PMIx_Get**, use the
26 **PMIX_RANK_WILDCARD** value for the rank to discover the PU list used for the provided
27 namespace.

28 **PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)

29 Application supports recoverable operations.

30 **PMIX_JOB_CONTINUOUS** "pmix.continuous" (bool)

31 Application is continuous, all failed processes should be immediately restarted.

32 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)

33 Maximum number of times to restart a process - when accessed using **PMIx_Get**, use the
34 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
35 namespace.

36 **PMIX_SET_ENVAR** "pmix.environ.set" (pmix_envar_t*)

37 Set the envar to the given value, overwriting any pre-existing one

```

1   PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
2     Unset the environment variable specified in the string.
3   PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
4     Add the environment variable, but do not overwrite any pre-existing one
5   PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
6     Prepend the given value to the specified environmental value using the given separator
7     character, creating the variable if it doesn't already exist
8   PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
9     Append the given value to the specified environmental value using the given separator
10    character, creating the variable if it doesn't already exist
11  PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
12    Ensure the given value appears first in the specified envar using the separator character,
13    creating the envar if it doesn't already exist
14  PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
15    Name of the WLM queue to which the allocation request is to be directed, or the queue being
16    referenced in a query.
17  PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
18    Total session time (in seconds) being requested in an allocation request.
19  PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
20    The number of nodes being requested in an allocation request.
21  PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
22    Regular expression of the specific nodes being requested in an allocation request.
23  PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
24    Number of PUs being requested in an allocation request.
25  PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
26    Regular expression of the number of PUs for each node being requested in an allocation
27    request.
28  PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
29    Regular expression of the specific PUs being requested in an allocation request.
30  PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
31    Number of Megabytes[base2] of memory (per process) being requested in an allocation
32    request.
33  PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
34    Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
35    request.
36  PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)

```

1 Fabric quality of service level for the job being requested in an allocation request.
2 **PMIX_ALLOC_FABRIC_TYPE** "pmix.alloc.nettype" (**char***)
3 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
4 **PMIX_ALLOC_FABRIC_PLANE** "pmix.alloc.netplane" (**char***)
5 ID string for the *fabric plane* to be used for the requested allocation.
6 **PMIX_ALLOC_FABRIC_ENDPTS** "pmix.alloc.endpts" (**size_t**)
7 Number of endpoints to allocate per *process* in the job.
8 **PMIX_ALLOC_FABRIC_ENDPTS_NODE** "pmix.alloc.endpts.nd" (**size_t**)
9 Number of endpoints to allocate per *node* for the job.
10 **PMIX_COSPAWN_APP** "pmix.cospawn" (**bool**)
11 Designated application is to be spawned as a disconnected job - i.e., the launcher shall not
12 include the application in any of the job-level values (e.g., **PMIX_RANK** within the job)
13 provided to any other application process generated by the same spawn request. Typically
14 used to cospawn debugger daemons alongside an application.
15 **PMIX_SPAWN_TOOL** "pmix.spwn.tool" (**bool**)
16 Indicate that the job being spawned is a tool.
17 **PMIX_EVENT_SILENT_TERMINATION** "pmix.evsilentterm" (**bool**)
18 Do not generate an event when this job normally terminates.



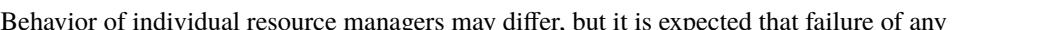
19 **Description**

20 Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspace*
21 parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the
22 namespace returned. The *nspace* array must be at least of size one more than **PMIX_MAX_NSLEN**.

23 By default, the spawned processes will be PMIx “connected” to the parent process upon successful
24 launch (see Section 11.3 for details). This includes that (a) the parent process will be given a copy
25 of the new job's information so it can query job-level info without incurring any communication
26 penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level
27 info, and (c) both the parent process and members of the child job will receive notification of errors
28 from processes in their combined assemblage.



Advice to users



29 Behavior of individual resource managers may differ, but it is expected that failure of any
30 application process to start will result in termination/cleanup of all processes in the newly spawned
31 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 17.2.2. 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.

11.2.2 `PMIx_Spawn_nb`

Summary

Nonblocking version of the `PMIx_Spawn` routine.

Format

```
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)

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the request - the `cbfunc` will *not* be called

Required Attributes

1 PMIx libraries are not required to directly support any attributes for this function. However, any
2 provided attributes must be passed to the host SMS daemon for processing.

3 Host environments are required to support the following attributes when present in either the
4 *job_info* or the *info* array of an element of the *apps* array:

5 **PMIX_WDIR "pmix.wdir" (char*)**

6 Working directory for spawned processes.

7 **PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)**

8 Set the current working directory to the session working directory assigned by the RM - can
9 be assigned to the entire job (by including attribute in the *job_info* array) or on a
10 per-application basis in the *info* array for each **pmix_app_t**.

11 **PMIX_PREFIX "pmix.prefix" (char*)**

12 Prefix to use for starting spawned processes - i.e., the directory where the executables can be
13 found.

14 **PMIX_HOST "pmix.host" (char*)**

15 Comma-delimited list of hosts to use for spawned processes.

16 **PMIX_HOSTFILE "pmix.hostfile" (char*)**

17 Hostfile to use for spawned processes.

Optional Attributes

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

19 **PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)**

20 Hostfile containing hosts to add to existing allocation.

21 **PMIX_ADD_HOST "pmix.addhost" (char*)**

22 Comma-delimited list of hosts to add to the allocation.

23 **PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)**

24 Preload executables onto nodes prior to executing launch procedure.

25 **PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)**

26 Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

27 **PMIX_PERSONALITY "pmix.pers" (char*)**

28 Name of personality corresponding to programming model used by application - supported
29 values depend upon PMIx implementation.

30 **PMIX_DISPLAY_MAP "pmix.dispmap" (bool)**

31 Display process mapping upon spawn.

32 **PMIX_PPR "pmix.ppr" (char*)**

```

1      Number of processes to spawn on each identified resource.
2      PMIX_MAPBY "pmix.mapby" (char*)
3          Process mapping policy - when accessed using PMIx_Get, use the
4          PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
5          provided namespace. Supported values are launcher specific.
6      PMIX_RANKBY "pmix.rankby" (char*)
7          Process ranking policy - when accessed using PMIx_Get, use the
8          PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the
9          provided namespace. Supported values are launcher specific.
10     PMIX_BINDTO "pmix.bindto" (char*)
11         Process binding policy - when accessed using PMIx_Get, use the
12         PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the
13         provided namespace. Supported values are launcher specific.
14     PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
15         Spawning process rank that is to receive any forwarded stdin.
16     PMIX_TAG_OUTPUT "pmix.tagout" (bool)
17         Tag stdout/stderr with the identity of the source process - can be assigned to the entire
18         job (by including attribute in the job_info array) or on a per-application basis in the info
19         array for each pmix_app_t.
20     PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
21         Timestamp output - can be assigned to the entire job (by including attribute in the job_info
22         array) or on a per-application basis in the info array for each pmix_app_t.
23     PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
24         Merge stdout and stderr streams - can be assigned to the entire job (by including
25         attribute in the job_info array) or on a per-application basis in the info array for each
26         pmix_app_t.
27     PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
28         Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be
29         assigned to the entire job (by including attribute in the job_info array) or on a per-application
30         basis in the info array for each pmix_app_t.
31     PMIX_INDEX_ARGV "pmix.indxargv" (bool)
32         Mark the argv with the rank of the process.
33     PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
34         Number of PUs to assign to each rank - when accessed using PMIx_Get, use the
35         PMIX_RANK_WILDCARD value for the rank to discover the PUs/process assigned to the
36         provided namespace.
37     PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
38         Do not place processes on the head node.

```

```

1   PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
2     Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
3     node.
4   PMIX_REPORT_BINDINGS "pmix.repbinding" (bool)
5     Report bindings of the individual processes.
6   PMIX_CPU_LIST "pmix.cplulist" (char*)
7     List of PUs to use for this job - when accessed using PMIx_Get, use the
8     PMIX_RANK_WILDCARD value for the rank to discover the PU list used for the provided
9     namespace.
10  PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
11    Application supports recoverable operations.
12  PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
13    Application is continuous, all failed processes should be immediately restarted.
14  PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
15    Maximum number of times to restart a process - when accessed using PMIx_Get, use the
16    PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
17    namespace.
18  PMIX_SET_ENVAR "pmix.environ.set" (pmix_envar_t*)
19    Set the envar to the given value, overwriting any pre-existing one
20  PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
21    Unset the environment variable specified in the string.
22  PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
23    Add the environment variable, but do not overwrite any pre-existing one
24  PMIX-prepend_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
25    Prepend 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_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
28    Append the given value to the specified environmental value using the given separator
29    character, creating the variable if it doesn't already exist
30  PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
31    Ensure the given value appears first in the specified envar using the separator character,
32    creating the envar if it doesn't already exist
33  PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
34    Name of the WLM queue to which the allocation request is to be directed, or the queue being
35    referenced in a query.
36  PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
37    Total session time (in seconds) being requested in an allocation request.

```

```

1   PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
2       The number of nodes being requested in an allocation request.
3
4   PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
5       Regular expression of the specific nodes being requested in an allocation request.
6
7   PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
8       Number of PUs being requested in an allocation request.
9
10  PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
11      Regular expression of the number of PUs for each node being requested in an allocation
12      request.
13
14  PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
15      Regular expression of the specific PUs being requested in an allocation request.
16
17  PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
18      Number of Megabytes[base2] of memory (per process) being requested in an allocation
19      request.
20
21  PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
22      Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
23      request.
24
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
28  PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
29      Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
30
31  PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
32      ID string for the fabric plane to be used for the requested allocation.
33
34  PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
35      Number of endpoints to allocate per process in the job.
36
37  PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
38      Number of endpoints to allocate per node for the job.
39
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
44      used to cospawn debugger daemons alongside an application.
45
46  PMIX_SPAWN_TOOL "pmix.spwn.tool" (bool)
47      Indicate that the job being spawned is a tool.
48
49  PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
50      Do not generate an event when this job normally terminates.

```

1 **Description**

2 Nonblocking version of the **PMIx_Spawn** routine. The provided callback function will be
3 executed upon successful start of *all* specified application processes.

4 **Advice to users**

5 Behavior of individual resource managers may differ, but it is expected that failure of any
6 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.

7 **11.2.3 Spawning-specific constants**

8 In addition to the generic error constants, the following spawning-specific error constants may be
9 returned by the spawning APIs:

10	PMIX_ERR_JOB_ALLOC_FAILED	The job request could not be executed due to failure to obtain the specified allocation
11	PMIX_ERR_JOB_APP_NOT_EXECUTABLE	The specified application executable either could not be found, or lacks execution privileges.
12	PMIX_ERR_JOB_NO_EXE_SPECIFIED	The job request did not specify an executable.
13	PMIX_ERR_JOB_FAILED_TO_MAP	The launcher was unable to map the processes for the specified job request.
14	PMIX_ERR_JOB_FAILED_TO_LAUNCH	One or more processes in the job request failed to launch
15		
16		
17		
18		

19 **11.2.4 Spawning attributes**

20 Attributes used to describe **PMIx_Spawn** behavior - they are values passed to the **PMIx_Spawn**
21 API and therefore are not accessed using the **PMIx_Get** APIs when used in that context. However,
22 some of the attributes defined in this section can be provided by the host environment for other
23 purposes - e.g., the host might provide the **PMIX_MAPBY** attribute in the job-related information so
24 that an application can use **PMIx_Get** to discover the mapping used for determining process
25 locations. Multi-use attributes and their respective access reference rank are denoted below.

26	PMIX_PERSONALITY "pmix.pers" (char*)	Name of personality corresponding to programming model used by application - supported values depend upon PMIx implementation.
27	PMIX_HOST "pmix.host" (char*)	Comma-delimited list of hosts to use for spawned processes.
28	PMIX_HOSTFILE "pmix.hostfile" (char*)	Hostfile to use for spawned processes.

```

1  PMIX_ADD_HOST "pmix.addhost" (char*)
2      Comma-delimited list of hosts to add to the allocation.
3  PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)
4      Hostfile containing hosts to add to existing allocation.
5  PMIX_PREFIX "pmix.prefix" (char*)
6      Prefix to use for starting spawned processes - i.e., the directory where the executables can be
7      found.
8  PMIX_WDIR "pmix.wdir" (char*)
9      Working directory for spawned processes.
10 PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
11     Display process mapping upon spawn.
12 PMIX_PPR "pmix.ppr" (char*)
13     Number of processes to spawn on each identified resource.
14 PMIX_MAPBY "pmix.mapby" (char*)
15     Process mapping policy - when accessed using PMIx\_Get, use the
16     PMIX\_RANK\_WILDCARD value for the rank to discover the mapping policy used for the
17     provided namespace. Supported values are launcher specific.
18 PMIX_RANKBY "pmix.rankby" (char*)
19     Process ranking policy - when accessed using PMIx\_Get, use the
20     PMIX\_RANK\_WILDCARD value for the rank to discover the ranking algorithm used for the
21     provided namespace. Supported values are launcher specific.
22 PMIX_BINDTO "pmix.bindto" (char*)
23     Process binding policy - when accessed using PMIx\_Get, use the
24     PMIX\_RANK\_WILDCARD value for the rank to discover the binding policy used for the
25     provided namespace. Supported values are launcher specific.
26 PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
27     Preload executables onto nodes prior to executing launch procedure.
28 PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
29     Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.
30 PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
31     Spawns process rank that is to receive any forwarded stdin.
32 PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
33     Set the current working directory to the session working directory assigned by the RM - can
34     be assigned to the entire job (by including attribute in the job_info array) or on a
35     per-application basis in the info array for each pmix\_app\_t.
36 PMIX_TAG_OUTPUT "pmix.tagout" (bool)
37     Tag stdout/stderr with the identity of the source process - can be assigned to the entire
38     job (by including attribute in the job_info array) or on a per-application basis in the info
39     array for each pmix\_app\_t.
40 PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
41     Timestamp output - can be assigned to the entire job (by including attribute in the job_info
42     array) or on a per-application basis in the info array for each pmix\_app\_t.
43 PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)

```

```

1 Merge stdout and stderr streams - can be assigned to the entire job (by including
2 attribute in the job_info array) or on a per-application basis in the info array for each
3 pmix_app_t.
4 PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
5 Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be
6 assigned to the entire job (by including attribute in the job_info array) or on a per-application
7 basis in the info array for each pmix_app_t.
8 PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
9 Direct output into files of form "<directory>/<jobid>/rank.<rank>/
10 stdout[err]" - can be assigned to the entire job (by including attribute in the job_info
11 array) or on a per-application basis in the info array for each pmix_app_t.
12 PMIX_INDEX_ARGV "pmix.indxargv" (bool)
13 Mark the argv with the rank of the process.
14 PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
15 Number of PUs to assign to each rank - when accessed using PMIx_Get, use the
16 PMIX_RANK_WILDCARD value for the rank to discover the PUs/process assigned to the
17 provided namespace.
18 PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
19 Do not place processes on the head node.
20 PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
21 Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
22 node.
23 PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
24 Report bindings of the individual processes.
25 PMIX_CPU_LIST "pmix.cpulist" (char*)
26 List of PUs to use for this job - when accessed using PMIx_Get, use the
27 PMIX_RANK_WILDCARD value for the rank to discover the PU list used for the provided
28 namespace.
29 PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
30 Application supports recoverable operations.
31 PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
32 Application is continuous, all failed processes should be immediately restarted.
33 PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
34 Maximum number of times to restart a process - when accessed using PMIx_Get, use the
35 PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
36 namespace.
37 PMIX_SPAWN_TOOL "pmix.spwn.tool" (bool)
38 Indicate that the job being spawned is a tool.
39 PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)
40 Include process stacktraces in timeout report from a job.
41 PMIX_TIMEOUT_REPORT_STATE "pmix.tim.state" (bool)
42 Report process states in timeout report from a job.
43 PMIX_NOTIFY_JOB_EVENTS "pmix.note.jev" (bool)

```

Requests that the launcher generate the **PMIX_EVENT_JOB_START**, **PMIX_LAUNCH_COMPLETE**, and **PMIX_EVENT_JOB_END** events. Each event is to include at least the namespace of the corresponding job and a **PMIX_EVENT_TIMESTAMP** indicating the time the event occurred. Note that the requester must register for these individual events, or capture and process them by registering a default event handler instead of individual handlers and then process the events based on the returned status code.
 Another common method is to register one event handler for all job-related events, with a separate handler for non-job events - see **PMIx_Register_event_handler** for details.

PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)

Requests that the launcher generate the **PMIX_EVENT_JOB_END** event for normal or abnormal termination of the spawned job. 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; and a **PMIX_EVENT_TIMESTAMP** indicating the time the termination occurred. Note that the requester must register for the event or capture and process it within a default event handler.

PMIX_NOTIFY_PROC_TERMINATION "pmix.noteproc" (bool)

Requests that the launcher generate the **PMIX_EVENT_PROC_TERMINATED** event whenever a process either normally or abnormally terminates.

PMIX_NOTIFY_PROC_ABNORMAL_TERMINATION "pmix.noteabproc" (bool)

Requests that the launcher generate the **PMIX_EVENT_PROC_TERMINATED** event only when a process abnormally terminates.

PMIX_LOG_PROC_TERMINATION "pmix.logproc" (bool)

Requests that the launcher log the **PMIX_EVENT_PROC_TERMINATED** event whenever a process either normally or abnormally terminates.

PMIX_LOG_PROC_ABNORMAL_TERMINATION "pmix.logabproc" (bool)

Requests that the launcher log the **PMIX_EVENT_PROC_TERMINATED** event only when a process abnormally terminates.

PMIX_LOG_JOB_EVENTS "pmix.log.jev" (bool)

Requests that the launcher log the **PMIX_EVENT_JOB_START**, **PMIX_LAUNCH_COMPLETE**, and **PMIX_EVENT_JOB_END** events using **PMIx_Log**, subject to the logging attributes of Section 12.4.3.

PMIX_LOG_COMPLETION "pmix.logcomp" (bool)

Requests that the launcher log the **PMIX_EVENT_JOB_END** event for normal or abnormal termination of the spawned job using **PMIx_Log**, subject to the logging attributes of Section 12.4.3. 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; and a **PMIX_EVENT_TIMESTAMP** indicating the time the termination occurred.

PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)

Do not generate an event when this job normally terminates.

Attributes used to adjust remote environment variables prior to spawning the specified application processes.

```
1 PMIX_SET_ENVAR "pmix.environ.set" (pmix_envar_t*)
2     Set the envar to the given value, overwriting any pre-existing one
3 PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
4     Unset the environment variable specified in the string.
5 PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
6     Add the environment variable, but do not overwrite any pre-existing one
7 PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
8     Prepend the given value to the specified environmental value using the given separator
9     character, creating the variable if it doesn't already exist
10 PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
11     Append 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_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
14     Ensure the given value appears first in the specified envar using the separator character,
15     creating the envar if it doesn't already exist
```

16 11.2.5 Application Structure

17 The `pmix_app_t` structure describes the application context for the `PMIx_Spawn` and
18 `PMIx_Spawn_nb` operations.

PMIx v1.0

C

```
19 typedef struct pmix_app {
20     /** Executable */
21     char *cmd;
22     /** Argument set, NULL terminated */
23     char **argv;
24     /** Environment set, NULL terminated */
25     char **env;
26     /** Current working directory */
27     char *cwd;
28     /** Maximum processes with this profile */
29     int maxprocs;
30     /** Array of info keys describing this application*/
31     pmix_info_t *info;
32     /** Number of info keys in 'info' array */
33     size_t ninfo;
34 } pmix_app_t;
```

C

35 11.2.5.1 App structure support macros

36 The following macros are provided to support the `pmix_app_t` structure.



1 **Free an app array**

2 Release an array of `pmix_app_t` structures

3 `PMIX_APP_FREE (m, n)`

4 **IN m**

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

6 **IN n**

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

8 **Create the info array of application directives**

9 Create an array of `pmix_info_t` structures for passing application-level directives, updating the
10 *ninfo* field of the `pmix_app_t` structure.

PMIx v2.2

11 `PMIX_APP_INFO_CREATE (m, n)`

12 **IN m**

13 Pointer to the `pmix_app_t` structure (handle)

14 **IN n**

15 Number of directives to be allocated (`size_t`)

16 **11.2.5.2 Spawn Callback Function**

17 **Summary**

18 The `pmix_spawn_cbfunc_t` is used on the PMIx client side by `PMIx_Spawn_nb` and on the
19 PMIx server side by `pmix_server_spawn_fn_t`.

PMIx v1.0

20 **typedef void (*pmix_spawn_cbfunc_t)**
21 (`pmix_status_t status,`
22 `pmix_nspace_t nspace, void *cbdata);`

23 **IN status**

24 Status associated with the operation (handle)

25 **IN nspace**

26 Namespace string (`pmix_nspace_t`)

27 **IN cbdata**

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

1 **Description**

2 The callback will be executed upon launch of the specified applications in **PMIx_Spawn_nb**, or
3 upon failure to launch any of them.

4 The *status* of the callback will indicate whether or not the spawn succeeded. The *nspace* of the
5 spawned processes will be returned, along with any provided callback data. Note that the returned
6 *nspace* value will not be protected upon return from the callback function, so the receiver must
7 copy it if it needs to be retained.

8

11.3 Connecting and Disconnecting Processes

9 This section defines functions to connect and disconnect processes in two or more separate PMIx
10 namespaces. The PMIx definition of *connected* solely implies that the host environment should
11 treat the failure of any process in the assemblage as a reportable event, taking action on the
12 assemblage as if it were a single application. For example, if the environment defaults (in the
13 absence of any application directives) to terminating an application upon failure of any process in
14 that application, then the environment should terminate all processes in the connected assemblage
15 upon failure of any member.

16 The host environment may choose to assign a new namespace to the connected assemblage and/or
17 assign new ranks for its members for its own internal tracking purposes. However, it is not required
18 to communicate such assignments to the participants (e.g., in response to an appropriate call to
19 **PMIx_Query_info_nb**). The host environment is required to generate a
20 **PMIX_ERR_PROC_TERM_WO_SYNC** event should any process in the assemblage terminate or
21 call **PMIx_Finalize** without first *disconnecting* from the assemblage. If the job including the
22 process is terminated as a result of that action, then the host environment is required to also
23 generate the **PMIX_ERR_JOB_TERM_WO_SYNC** for all jobs that were terminated as a result.

24

Advice to PMIx server hosts

25 The *connect* operation does not require the exchange of job-level information nor the inclusion of
26 information posted by participating processes via **PMIx_Put**. Indeed, the callback function
27 utilized in **pmix_server_connect_fn_t** cannot pass information back into the PMIx server
28 library. However, host environments are advised that collecting such information at the
29 participating daemons represents an optimization opportunity as participating processes are likely
 to request such information after the connect operation completes.

Advice to users

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation. While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

Neither the PMIx implementation nor host environment are required to provide any tracking support for the assemblage. Thus, the application is responsible for maintaining the membership list of the assemblage.

11.3.1 PMIx_Connect

Summary

Connect namespaces.

Format

```
pmix_status_t  
PMIx_Connect(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 a negative value corresponding to a PMIx error constant.

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

1 The following attributes are optional for PMIx implementations:

2 **PMIX_ALL_CLONES_PARTICIPATE "pmix.clone.part" (bool)**

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

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

5 **PMIX_TIMEOUT "pmix.timeout" (int)**

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

Description

10 Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The
11 function will return once all processes identified in *procs* have called either **PMIx_Connect** or its
12 non-blocking version, *and* the host environment has completed any supporting operations required
13 to meet the terms of the PMIx definition of *connected* processes.

14 A process can only engage in one connect operation involving the identical *procs* array at a time.
15 However, a process can be simultaneously engaged in multiple connect operations, each involving a
16 different *procs* array.

17 As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level
18 directives regarding timeout constraints and other options available from the host RM.

Advice to users

19 All processes engaged in a given **PMIx_Connect** operation must provide the identical *procs* array
20 as ordering of entries in the array and the method by which those processes are identified (e.g., use
21 of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host
22 environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

11.3.2 PMIx_Connect_nb

Summary

Nonblocking [PMIx_Connect_nb](#) routine.

Format

```
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)
```

IN procs

Array of proc structures (array of handles)

IN nprocs

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

IN info

Array of info structures (array of handles)

IN ninfo

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

IN cbfunc

Callback function [pmix_op_cbfunc_t](#) (function reference)

IN cbdata

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

1 The following attributes are optional for PMIx implementations:

2 **PMIX_ALL_CLONES_PARTICIPATE** "pmix.clone.part" (bool)
3 All *clones* of the calling process must participate in the collective operation.

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

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

Description

9 Nonblocking version of **PMIx_Connect**. The callback function is called once all processes
10 identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host
11 environment has completed any supporting operations required to meet the terms of the PMIx
12 definition of *connected* processes. See the advice provided in the description for **PMIx_Connect**
13 for more information.

11.3.3 PMIx_Disconnect

Summary

Disconnect a previously connected set of processes.

Format

15 *PMIx v1.0*  C
16 **pmix_status_t**
17 **PMIx_Disconnect**(**const pmix_proc_t procs[], size_t nprocs,**
18 **const pmix_info_t info[], size_t ninfo);**

22 **IN procs**
23 Array of proc structures (array of handles)
24 **IN nprocs**
25 Number of elements in the *procs* array (integer)
26 **IN info**
27 Array of info structures (array of handles)
28 **IN ninfo**
29 Number of elements in the *info* array (integer)
30 Returns one of the following:
31 • **PMIX_SUCCESS**, indicating that the request was successfully executed

- the **PMIX_ERR_INVALID_OPERATION** error indicating that the specified set of *procs* was not previously *connected* via a call to **PMIx_Connect** or its non-blocking form.
 - a PMIx error constant indicating either an error in the input or that the request failed

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

The following attributes are optional for PMIx implementations:

PMIX_ALL_CLONES_PARTICIPATE "pmix.clone.part" (bool)
All *clones* of the calling process must participate in the collective operation.

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

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

Description

Disconnect a previously connected set of processes. The function will return once all processes identified in *procs* have called either **PMIx_Disconnect** or its non-blocking version, *and* the host environment has completed any required supporting operations.

A process can only engage in one disconnect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple disconnect operations, each involving a different *procs* array.

As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

Advice to users

All processes engaged in a given **PMIx_Disconnect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

11.3.4 PMIx_Disconnect_nb

Summary

Nonblocking **PMIx_Disconnect** routine.

Format

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

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)

Returns one of the following:

- 1 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
2 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
3 function prior to returning from the API.
4 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
5 returned *success* - the *cbfunc* will *not* be called
6 • a PMIx error constant indicating either an error in the input or that the request was immediately
7 processed and failed - the *cbfunc* will *not* be called

Required Attributes

8 PMIx libraries are not required to directly support any attributes for this function. However, any
9 provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

10 The following attributes are optional for PMIx implementations:

11 **PMIX_ALL_CLONES_PARTICIPATE** "pmix.clone.part" (bool)
12 All *clones* of the calling process must participate in the collective operation.

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

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

Description

19 Nonblocking **PMIx_Disconnect** routine. The callback function is called either:

- 20 • to return the **PMIX_ERR_INVALID_OPERATION** error indicating that the specified set of
21 *procs* was not previously *connected* via a call to **PMIx_Connect** or its non-blocking form;
22 • to return a PMIx error constant indicating that the operation failed; or
23 • once all processes identified in *procs* have called either **PMIx_Disconnect_nb** or its
24 blocking version, *and* the host environment has completed any required supporting operations.

25 See the advice provided in the description for **PMIx_Disconnect** for more information.

11.4 Process Locality

The relative locality of processes is often used to optimize their interactions with the hardware and other processes. PMIx provides a means by which the host environment can communicate the locality of a given process using the `PMIx_server_generate_locality_string` to generate an abstracted representation of that value. This provides a human-readable format and allows the client to parse the locality string with a method of its choice that may differ from the one used by the server that generated it.

There are times, however, when relative locality and other PMIx-provided information doesn't include some element required by the application. In these instances, the application may need access to the full description of the local hardware topology. PMIx does not itself generate such descriptions - there are multiple third-party libraries that fulfill that role. Instead, PMIx offers an abstraction method by which users can obtain a pointer to the description. This transparently enables support for different methods of sharing the topology between the host environment (which may well have already generated it prior to local start of application processes) and the clients - e.g., through passing of a shared memory region.

11.4.1 `PMIx_Load_topology`

Summary

Load the local hardware topology description

Format

```
pmix_status_t  
PMIx_Load_topology(pmix_topology_t *topo);
```

INOUT topo

Address of a `pmix_topology_t` structure where the topology information is to be loaded (handle)

Returns `PMIX_SUCCESS`, indicating that the *topo* was successfully loaded, or an appropriate PMIx error constant.

Description

Obtain a pointer to the topology description of the local node. If the *source* field of the provided `pmix_topology_t` is set, then the PMIx library must return a description from the specified implementation or else indicate that the implementation is not available by returning the `PMIX_ERR_NOT_SUPPORTED` error constant.

The returned pointer may point to a shared memory region or an actual instance of the topology description. In either case, the description shall be treated as a "read-only" object - attempts to modify the object are likely to fail and return an error. The PMIx library is responsible for performing any required cleanup when the client library finalizes.

Advice to users

1 It is the responsibility of the user to ensure that the *topo* argument is properly initialized prior to
2 calling this API, and to check the returned *source* to verify that the returned topology description is
3 compatible with the user's code.

4 11.4.2 PMIx_Get_relative_locality

5 Summary

6 Get the relative locality of two local processes given their locality strings.

7 Format

```
8 pmix_status_t
9 PMIx_Get_relative_locality(const char *locality1,
10                      const char *locality2,
11                      pmix_locality_t *locality);
```

12 IN locality1

13 String returned by the `PMIx_server_generate_locality_string` API (handle)

14 IN locality2

15 String returned by the `PMIx_server_generate_locality_string` API (handle)

16 INOUT locality

17 Location where the relative locality bitmask is to be constructed (memory reference)

18 Returns `PMIX_SUCCESS`, indicating that the *locality* was successfully loaded, or an appropriate
19 PMIx error constant.

20 Description

21 Parse the locality strings of two processes (as returned by `PMIx_Get` using the
22 `PMIX_LOCALITY_STRING` key) and set the appropriate `pmix_locality_t` locality bits in
23 the provided memory location.

24 11.4.2.1 Topology description

25 The `pmix_topology_t` structure contains a (case-insensitive) string identifying the source of
26 the topology (e.g., "hwloc") and a pointer to the corresponding implementation-specific topology
27 description.

PMIx v4.0

```
28     typedef struct pmix_topology {
29         char *source;
30         void *topology;
31     } pmix_topoology_t;
```

11.4.2.2 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

5 `PMIX_TOPOLOGY_CONSTRUCT (m)`

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

10 `PMIX_TOPOLOGY_DESTRUCT (m)`

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

15 `PMIX_TOPOLOGY_CREATE (m, n)`

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

23 `PMIX_TOPOLOGY_FREE (m, n)`

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`)

11.4.2.3 Relative locality of two processes

The `pmix_locality_t` datatype is a `uint16_t` bitmask that defines the relative locality of two processes on a node. The following constants represent specific bits in the mask and can be used to test a locality value using standard bit-test methods.

5	<code>PMIX_LOCALITY_UNKNOWN</code>	All bits are set to zero, indicating that the relative locality of the
6		two processes is unknown
7	<code>PMIX_LOCALITY_NONLOCAL</code>	The two processes do not share any common locations
8	<code>PMIX_LOCALITY_SHARE_HWTHREAD</code>	The two processes share at least one hardware thread
9	<code>PMIX_LOCALITY_SHARE_CORE</code>	The two processes share at least one core
10	<code>PMIX_LOCALITY_SHARE_L1CACHE</code>	The two processes share at least an L1 cache
11	<code>PMIX_LOCALITY_SHARE_L2CACHE</code>	The two processes share at least an L2 cache
12	<code>PMIX_LOCALITY_SHARE_L3CACHE</code>	The two processes share at least an L3 cache
13	<code>PMIX_LOCALITY_SHARE_PACKAGE</code>	The two processes share at least a package
14	<code>PMIX_LOCALITY_SHARE_NUMA</code>	The two processes share at least one Non-Uniform
15		Memory Access (NUMA) region
16	<code>PMIX_LOCALITY_SHARE_NODE</code>	The two processes are executing on the same node

Implementers and vendors may choose to extend these definitions as needed to describe a particular system.

11.4.2.4 Locality keys

`PMIX_LOCALITY_STRING "pmix.locstr" (char*)`

String describing a process's bound location - referenced using the process's rank. The string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon. The remainder of the string represents the corresponding locality as expressed by the underlying implementation. The entire string must be passed to `PMIx_Get_relative_locality` for processing. Note that hosts are only required to provide locality strings for local client processes - thus, a call to `PMIx_Get` for the locality string of a process that returns `PMIX_ERR_NOT_FOUND` indicates that the process is not executing on the same node.

11.4.3 PMIx_Parse_cpuset_string

Summary

Parse the PU binding bitmap from its string representation.

Format

C

```
pmix_status_t  
PMIx_Parse_cpuset_string(const char *cpuset_string,  
                           pmix_cpuset_t *cpuset);
```

1 **IN cpuset_string**
2 String returned by the **PMIx_server_generate_cpuset_string** API (handle)
3 **INOUT cpuset**
4 Address of an object where the bitmap is to be stored (memory reference)
5 Returns **PMIX_SUCCESS**, indicating that the *cpuset* was successfully loaded, or an appropriate
6 PMIx error constant.

7 **Description**
8 Parse the string representation of the binding bitmap (as returned by **PMIx_Get** using the
9 **PMIX_CPUSSET** key) and set the appropriate PU binding location information in the provided
10 memory location.

11 11.4.4 PMIx_Get_cpuset

12 Summary

13 Get the PU binding bitmap of the current process.

14 PMIx v4.0 Format

```
15   pmix_status_t  
16   PMIx_Get_cpuset(pmix_cpuset_t *cpuset, pmix_bind_envelope_t ref);
```

17 INOUT cpuset

18 Address of an object where the bitmap is to be stored (memory reference)

19 IN ref

20 The binding envelope to be considered when formulating the bitmap
21 (**pmix_bind_envelope_t**)

22 Returns **PMIX_SUCCESS**, indicating that the *cpuset* was successfully loaded, or an appropriate
23 PMIx error constant.

24 Description

25 Obtain and set the appropriate PU binding location information in the provided memory location
26 based on the specified binding envelope.

27 11.4.4.1 Binding envelope

28 **PMIx v4.0**

29 The **pmix_bind_envelope_t** data type defines the envelope of threads within a possibly
30 multi-threaded process that are to be considered when getting the cpuset associated with the
process. Valid values include:

- 31 **PMIX_CPBIND_PROCESS** Use the location of all threads in the possibly multi-threaded
32 process.
33 **PMIX_CPBIND_THREAD** Use only the location of the thread calling the API.

1 11.4.5 PMIx_Compute_distances

2 Summary

3 Compute distances from specified process location to local devices.

4 Format

```
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

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 one of the following:

- 27 • **PMIX_SUCCESS** indicating that the distances were returned.
- 28 • a non-zero PMIx error constant indicating the reason the request failed.

29 Description

30 Both the minimum and maximum distance fields in the elements of the array shall be filled with the
31 respective distances between the current process location and the types of devices or specific device
32 identified in the *info* directives. In the absence of directives, distances to all supported device types
33 shall be returned.

Advice to users

A process whose threads are not all bound to the same location may return inconsistent results from calls to this API by different threads if the **PMIX_CPUBIND_THREAD** binding envelope was used when generating the *cpuset*.

11.4.6 PMIx_Compute_distances_nb

Summary

Compute distances from specified process location to local devices.

Format

```
pmix_status_t  
PMIx_Compute_distances_nb(pmix_topology_t *topo,  
                           pmix_cpuset_t *cpuset,  
                           pmix_info_t info[], size_t ninfo[],  
                           pmix_device_dist_cbfunc_t cbfunc,  
                           void *cbdata);
```

IN topo

Pointer to the topology description of the node where the process is located (**NULL** indicates the local node) (**pmix_topology_t**)

IN cpuset

Pointer to the location of the process (**pmix_cpuset_t**)

IN info

Array of **pmix_info_t** describing the devices whose distance is to be computed (handle)

IN ninfo

Number of elements in *info* (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 has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

1 **Description**

2 Non-blocking form of the [PMIx_Compute_distances](#) API.

3 **11.4.7 Device Distance Callback Function**

4 **Summary**

5 The [pmix_device_dist_cbfunc_t](#) is used to return an array of device distances.

PMIx v4.0

C

```
6      typedef void (*pmix_device_dist_cbfunc_t)
7            (pmix_status_t status,
8            pmix_device_distance_t *dist,
9            size_t ndist,
10          void *cbdata,
11          pmix_release_cbfunc_t release_fn,
12          void *release_cbdata);
```

C

13 **IN status**

14 Status associated with the operation ([pmix_status_t](#))

15 **IN dist**

16 Array of [pmix_device_distance_t](#) returned by the operation (pointer)

17 **IN ndist**

18 Number of elements in the *dist* array ([size_t](#))

19 **IN cbdata**

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

21 **IN release_fn**

22 Function to be called when done with the *dist* data (function pointer)

23 **IN release_cbdata**

24 Callback data to be passed to *release_fn* (memory reference)

25 **Description**

26 The *status* indicates if requested data was found or not. The array of

27 [pmix_device_distance_t](#) will contain the distance information.

28 **11.4.8 Device type**

29 The [pmix_device_type_t](#) is a [uint64_t](#) bitmask for identifying the type(s) whose
30 distances are being requested, or the type of a specific device being referenced (e.g., in a
31 [pmix_device_distance_t](#) object).

PMIx v1.0

C

```
32      typedef uint16_t pmix_device_type_t;
```

1 The following constants can be used to set a variable of the type `pmix_device_type_t`.

2 **PMIX_DEVTYPE_UNKNOWN** The device is of an unknown type - will not be included in
3 returned device distances.

4 **PMIX_DEVTYPE_BLOCK** Operating system block device, or non-volatile memory device
5 (e.g., "sda" or "dax2.0" on Linux).

6 **PMIX_DEVTYPE_GPU** Operating system Graphics Processing Unit (GPU) device (e.g.,
7 "card0" for a Linux Direct Rendering Manager (DRM) device).

8 **PMIX_DEVTYPE_NETWORK** Operating system network device (e.g., the "eth0" interface on
9 Linux).

10 **PMIX_DEVTYPE_OPENFABRICS** Operating system OpenFabrics device (e.g., an "mlx4_0"
11 InfiniBand Host Channel Adapter (HCA), or "hfi1_0" Omni-Path interface on Linux).

12 **PMIX_DEVTYPE_DMA** Operating system Direct Memory Access (DMA) engine device (e.g.,
13 the "dma0chan0" DMA channel on Linux).

14 **PMIX_DEVTYPE_COPROC** Operating system co-processor device (e.g., "mic0" for a Xeon Phi
15 on Linux, "opencl0d0" for a OpenCL device, or "cuda0" for a Compute Unified Device
16 Architecture (CUDA) device).

17 11.4.9 Device Distance Structure

18 The `pmix_device_distance_t` structure contains the minimum and maximum relative
19 distance from the caller to a given device.

PMIx v4.0

```
20     typedef struct pmix_device_distance {
21         char *uuid;
22         char *osname;
23         pmix_device_type_t type;
24         uint16_t mindist;
25         uint16_t maxdist;
26     } pmix_device_distance_t;
```

27 The *uuid* is a string identifier guaranteed to be unique within the cluster and is typically assembled
28 from discovered device attributes (e.g., the Internet Protocol (IP) address of the device). The
29 *osname* is the local operating system name of the device and is only unique to that node.

30 The two distance fields provide the minimum and maximum relative distance to the device from the
31 specified location of the process, expressed as a 16-bit integer value where a smaller number
32 indicates that this device is closer to the process than a device with a larger distance value. Note
33 that relative distance values are not necessarily correlated to a physical property - e.g., a device at
34 twice the distance from another device does not necessarily have twice the latency for
35 communication with it.

1 Relative distances only apply to similar devices and cannot be used to compare devices of different
2 types. Both minimum and maximum distances are provided to support cases where the process may
3 be bound to more than one location, and the locations are at different distances from the device.

4 A relative distance value of **UINT16_MAX** indicates that the distance from the process to the device
5 could not be provided. This may be due to lack of available information (e.g., the PMIx
6 library not having access to device locations) or other factors.

7 11.4.10 Device distance support macros

8 The following macros are provided to support the [pmix_device_distance_t](#) structure.

9 Initialize the device distance structure

10 Initialize the [pmix_device_distance_t](#) fields.

PMIx v4.0

11 **PMIX_DEVICE_DIST_CONSTRUCT** (m)

12 IN m

13 Pointer to the structure to be initialized (pointer to [pmix_device_distance_t](#))

14 Destruct the device distance structure

15 Destruct the [pmix_device_distance_t](#) fields.

PMIx v4.0

16 **PMIX_DEVICE_DIST_DESTRUCT** (m)

17 IN m

18 Pointer to the structure to be destructed (pointer to [pmix_device_distance_t](#))

19 Create an device distance array

20 Allocate and initialize a [pmix_device_distance_t](#) array.

PMIx v4.0

21 **PMIX_DEVICE_DIST_CREATE** (m, n)

22 **INOUT** m

23 Address where the pointer to the array of [pmix_device_distance_t](#) structures shall be
24 stored (handle)

25 IN n

26 Number of structures to be allocated ([size_t](#))

1 **Release an device distance array**
2 Release an array of `pmix_device_distance_t` structures.

3 **PMIX_DEVICE_DIST_FREE (m, n)**

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

8 11.4.11 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 12

Job Management and Reporting

The job management APIs provide an application with the ability to orchestrate its operation in partnership with the SMS. Members of this category include the **PMIx_Allocation_request**, **PMIx_Job_control**, and **PMIx_Process_monitor** APIs.

12.1 Allocation Requests

This section defines functionality to request new allocations from the RM, and request modifications to existing allocations. These are primarily used in the following scenarios:

- *Evolving* applications that dynamically request and return resources as they execute.
- *Malleable* environments where the scheduler redirects resources away from executing applications for higher priority jobs or load balancing.
- *Resilient* applications that need to request replacement resources in the face of failures.
- *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time, but realizes that they underestimated their required time while executing.

PMIx attempts to address this range of use-cases with a flexible API.

12.1.1 PMIx_Allocation_request

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v3.0



```
pmix_status_t  
PMIx_Allocation_request(pmix_alloc_directive_t directive,  
                         pmix_info_t info[], size_t ninfo,  
                         pmix_info_t *results[], size_t *nresults);
```

```

1  IN  directive          Allocation directive (pmix_alloc_directive_t)
2  IN  info               Array of pmix_info_t structures (array of handles)
3  IN  ninfo              Number of elements in the info array (integer)
4  INOUT results          Address where a pointer to an array of pmix_info_t containing the results of the request
5  can be returned (memory reference)
6  INOUT nresults         Address where the number of elements in results can be returned (handle)
7
8  Returns one of the following:
9
10 • PMIX_SUCCESS, indicating that the request was processed and returned success
11 • a PMIx error constant indicating either an error in the input or that the request was refused
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

```

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

```

PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
    User-provided string identifier for this allocation request which can later be used to query
    status of the request.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
    The number of nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
    Number of PUs being requested in an allocation request.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
    Total session time (in seconds) being requested in an allocation request.

```

Optional Attributes

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

PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (**char***)
Regular expression of the specific nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (**char***)
Regular expression of the number of PUs for each node being requested in an allocation request.

PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (**char***)
Regular expression of the specific PUs being requested in an allocation request.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (**float**)
Number of Megabytes[base2] of memory (per process) being requested in an allocation request.

PMIX_ALLOC_FABRIC "pmix.alloc.net" (**array**)
Array of **pmix_info_t** describing requested fabric resources. This must include at least:
PMIX_ALLOC_FABRIC_ID, **PMIX_ALLOC_FABRIC_TYPE**, and
PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.

PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (**char***)
The key to be used when accessing this requested fabric allocation. The fabric allocation will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** whose first element is composed of this key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a Transmission Control Protocol (TCP) allocation might consist of a comma-delimited string of socket ranges such as "32000-32100,33005,38123-38146". Additional array entries will consist of any provided resource request directives, along with their assigned values. Examples include: **PMIX_ALLOC_FABRIC_TYPE** - the type of resources provided; **PMIX_ALLOC_FABRIC_PLANE** - if applicable, what plane the resources were assigned from; **PMIX_ALLOC_FABRIC_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_FABRIC_SEC_KEY** - a security key for the requested fabric allocation. NOTE: the array contents may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (**float**)
Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (**char***)
Fabric quality of service level for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (**char***)
Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (**char***)

```
1 ID string for the fabric plane to be used for the requested allocation.  
2 PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)  
3 Number of endpoints to allocate per process in the job.  
4 PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)  
5 Number of endpoints to allocate per node for the job.  
6 PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)  
7 Request that the allocation include a fabric security key for the spawned job.
```



8 **Description**

9 Request an allocation operation from the host resource manager. Several broad categories are
10 envisioned, including the ability to:

- 11 • Request allocation of additional resources, including memory, bandwidth, and compute. This
12 should be accomplished in a non-blocking manner so that the application can continue to
13 progress while waiting for resources to become available. Note that the new allocation will be
14 disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one
15 allocation will not impact the other.
- 16 • Extend the reservation on currently allocated resources, subject to scheduling availability and
17 priorities. This includes extending the time limit on current resources, and/or requesting
18 additional resources be allocated to the requesting job. Any additional allocated resources will be
19 considered as part of the current allocation, and thus will be released at the same time.
- 20 • Return no-longer-required resources to the scheduler. This includes the “loan” of resources back
21 to the scheduler with a promise to return them upon subsequent request.

22 If successful, the returned results for a request for additional resources must include the host
23 resource manager’s identifier (**PMIX_ALLOC_ID**) that the requester can use to specify the
24 resources in, for example, a call to **PMIx_Spawn**.

25 **12.1.2 PMIx_Allocation_request_nb**

26 **Summary**

27 Request an allocation operation from the host resource manager.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,  
4                               pmix_info_t info[], size_t ninfo,  
5                               pmix_info_cbfnc_t cbfunc, void *cbdata);
```

C

6 IN directive

Allocation directive ([pmix_alloc_directive_t](#))

7 IN info

Array of [pmix_info_t](#) structures (array of handles)

8 IN ninfo

Number of elements in the info array (integer)

9 IN cbfunc

Callback function [pmix_info_cbfnc_t](#) (function reference)

10 IN cbdata

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 library must not invoke the callback
14 function 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 • a PMIx error constant indicating either an error in the input or that the request was immediately
18 processed and failed - the *cbfunc* will *not* be called

19 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 SMS daemon for processing, and the PMIx library is
22 required to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
23 the request.

24 Host environments that implement support for this operation are required to support the following
25 attributes:

26 **PMIX_ALLOC_REQ_ID** "pmix.alloc.reqid" (char*)

27 User-provided string identifier for this allocation request which can later be used to query
28 status of the request.

29 **PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (uint64_t)

30 The number of nodes being requested in an allocation request.

31 **PMIX_ALLOC_NUM_CPUS** "pmix.alloc.ncpus" (uint64_t)

1 Number of PUs being requested in an allocation request.
2 **PMIX_ALLOC_TIME** "pmix.alloc.time" (**uint32_t**)
3 Total session time (in seconds) being requested in an allocation request.



Optional Attributes

4 The following attributes are optional for host environments that support this operation:
5 **PMIX_ALLOC_NODE_LIST** "pmix.alloc.nlist" (**char***)
6 Regular expression of the specific nodes being requested in an allocation request.
7 **PMIX_ALLOC_NUM_CPU_LIST** "pmix.alloc.ncpulist" (**char***)
8 Regular expression of the number of PUs for each node being requested in an allocation
9 request.
10 **PMIX_ALLOC_CPU_LIST** "pmix.alloc.cpulist" (**char***)
11 Regular expression of the specific PUs being requested in an allocation request.
12 **PMIX_ALLOC_MEM_SIZE** "pmix.alloc.msize" (**float**)
13 Number of Megabytes[base2] of memory (per process) being requested in an allocation
14 request.
15 **PMIX_ALLOC_FABRIC** "pmix.alloc.net" (**array**)
16 Array of **pmix_info_t** describing requested fabric resources. This must include at least:
17 **PMIX_ALLOC_FABRIC_ID**, **PMIX_ALLOC_FABRIC_TYPE**, and
18 **PMIX_ALLOC_FABRIC_ENDPTS**, plus whatever other descriptors are desired.
19 **PMIX_ALLOC_FABRIC_ID** "pmix.alloc.netid" (**char***)
20 The key to be used when accessing this requested fabric allocation. The fabric allocation
21 will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** whose first
22 element is composed of this key and the allocated resource description. The type of the
23 included value depends upon the fabric support. For example, a TCP allocation might
24 consist of a comma-delimited string of socket ranges such as "**32000-32100**,
25 **33005,38123-38146**". Additional array entries will consist of any provided resource
26 request directives, along with their assigned values. Examples include:
27 **PMIX_ALLOC_FABRIC_TYPE** - the type of resources provided;
28 **PMIX_ALLOC_FABRIC_PLANE** - if applicable, what plane the resources were assigned
29 from; **PMIX_ALLOC_FABRIC_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** -
30 the allocated bandwidth; **PMIX_ALLOC_FABRIC_SEC_KEY** - a security key for the
31 requested fabric allocation. NOTE: the array contents may differ from those requested,
32 especially if **PMIX_INFO_REQD** was not set in the request.
33 **PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (**float**)
34 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
35 request.
36 **PMIX_ALLOC_FABRIC_QOS** "pmix.alloc.netqos" (**char***)

1 Fabric quality of service level for the job being requested in an allocation request.
2 **PMIX_ALLOC_FABRIC_TYPE** "pmix.alloc.nettype" (**char***)
3 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
4 **PMIX_ALLOC_FABRIC_PLANE** "pmix.alloc.netplane" (**char***)
5 ID string for the *fabric plane* to be used for the requested allocation.
6 **PMIX_ALLOC_FABRIC_ENDPTS** "pmix.alloc.endpts" (**size_t**)
7 Number of endpoints to allocate per *process* in the job.
8 **PMIX_ALLOC_FABRIC_ENDPTS_NODE** "pmix.alloc.endpts.nd" (**size_t**)
9 Number of endpoints to allocate per *node* for the job.
10 **PMIX_ALLOC_FABRIC_SEC_KEY** "pmix.alloc.nsec" (**pmix_byte_object_t**)
11 Request that the allocation include a fabric security key for the spawned job.



12 **Description**

13 Non-blocking form of the **PMIx_Allocation_request** API.

14 **12.1.3 Job Allocation attributes**

15 Attributes used to describe the job allocation - these are values passed to and/or returned by the
16 **PMIx_Allocation_request_nb** and **PMIx_Allocation_request** APIs and are not
17 accessed using the **PMIx_Get** API.

18 **PMIX_ALLOC_REQ_ID** "pmix.alloc.reqid" (**char***)
19 User-provided string identifier for this allocation request which can later be used to query
20 status of the request.
21 **PMIX_ALLOC_ID** "pmix.alloc.id" (**char***)
22 A string identifier (provided by the host environment) for the resulting allocation which can
23 later be used to reference the allocated resources in, for example, a call to **PMIx_Spawn**.
24 **PMIX_ALLOC_QUEUE** "pmix.alloc.queue" (**char***)
25 Name of the WLM queue to which the allocation request is to be directed, or the queue being
26 referenced in a query.
27 **PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (**uint64_t**)
28 The number of nodes being requested in an allocation request.
29 **PMIX_ALLOC_NODE_LIST** "pmix.alloc.nlist" (**char***)
30 Regular expression of the specific nodes being requested in an allocation request.
31 **PMIX_ALLOC_NUM_CPLUS** "pmix.alloc.ncpus" (**uint64_t**)
32 Number of PUs being requested in an allocation request.
33 **PMIX_ALLOC_NUM_CPU_LIST** "pmix.alloc.ncpulist" (**char***)
34 Regular expression of the number of PUs for each node being requested in an allocation
35 request.
36 **PMIX_ALLOC_CPU_LIST** "pmix.alloc.cpulist" (**char***)

```

1      Regular expression of the specific PUs being requested in an allocation request.
2      PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
3          Number of Megabytes[base2] of memory (per process) being requested in an allocation
4          request.
5      PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
6          Array of pmix_info_t describing requested fabric resources. This must include at least:
7              PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
8              PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
9      PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
10         The key to be used when accessing this requested fabric allocation. The fabric allocation
11         will be returned/stored as a pmix_data_array_t of pmix_info_t whose first
12         element is composed of this key and the allocated resource description. The type of the
13         included value depends upon the fabric support. For example, a TCP allocation might
14         consist of a comma-delimited string of socket ranges such as "32000-32100,
15         33005,38123-38146". Additional array entries will consist of any provided resource
16         request directives, along with their assigned values. Examples include:
17             PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
18             PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
19             from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
20             the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
21             requested fabric allocation. NOTE: the array contents may differ from those requested,
22             especially if PMIX_INFO_REQD was not set in the request.
23      PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
24         Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
25         request.
26      PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
27         Fabric quality of service level for the job being requested in an allocation request.
28      PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
29         Total session time (in seconds) being requested in an allocation request.
30      PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
31         Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
32      PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
33         ID string for the fabric plane to be used for the requested allocation.
34      PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
35         Number of endpoints to allocate per process in the job.
36      PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
37         Number of endpoints to allocate per node for the job.
38      PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
39         Request that the allocation include a fabric security key for the spawned job.

```

40 12.1.4 Job Allocation Directives

41 *PMIx v2.0*

42 The **pmix_alloc_directive_t** structure is a **uint8_t** type that defines the behavior of
allocation requests. The following constants can be used to set a variable of the type

1 **`pmix_alloc_directive_t`**. All definitions were introduced in version 2 of the standard
2 unless otherwise marked.

3 **`PMIX_ALLOC_NEW`** A new allocation is being requested. The resulting allocation will be
4 disjoint (i.e., not connected in a job sense) from the requesting allocation.

5 **`PMIX_ALLOC_EXTEND`** Extend the existing allocation, either in time or as additional
6 resources.

7 **`PMIX_ALLOC_RELEASE`** Release part of the existing allocation. Attributes in the
8 accompanying **`pmix_info_t`** array may be used to specify permanent release of the
9 identified resources, or “lending” of those resources for some period of time.

10 **`PMIX_ALLOC_REAQUIRE`** Reacquire resources that were previously “lent” back to the
11 scheduler.

12 **`PMIX_ALLOC_EXTERNAL`** A value boundary above which implementers are free to define
13 their own directive values.

14 12.2 Job Control

15 This section defines APIs that enable the application and host environment to coordinate the
16 response to failures and other events. This can include requesting termination of the entire job or a
17 subset of processes within a job, but can also be used in combination with other PMIx capabilities
18 (e.g., allocation support and event notification) for more nuanced responses. For example, an
19 application notified of an incipient over-temperature condition on a node could use the
20 **`PMIx_Allocation_request_nb`** interface to request replacement nodes while
21 simultaneously using the **`PMIx_Job_control_nb`** interface to direct that a checkpoint event be
22 delivered to all processes in the application. If replacement resources are not available, the
23 application might use the **`PMIx_Job_control_nb`** interface to request that the job continue at a
24 lower power setting, perhaps sufficient to avoid the over-temperature failure.

25 The job control APIs can also be used by an application to register itself as available for preemption
26 when operating in an environment such as a cloud or where incentives, financial or otherwise, are
27 provided to jobs willing to be preempted. Registration can include attributes indicating how many
28 resources are being offered for preemption (e.g., all or only some portion), whether the application
29 will require time to prepare for preemption, etc. Jobs that request a warning will receive an event
30 notifying them of an impending preemption (possibly including information as to the resources that
31 will be taken away, how much time the application will be given prior to being preempted, whether
32 the preemption will be a suspension or full termination, etc.) so they have an opportunity to save
33 their work. Once the application is ready, it calls the provided event completion callback function to
34 indicate that the SMS is free to suspend or terminate it, and can include directives regarding any
35 desired restart.

36 12.2.1 **`PMIx_Job_control`**

37 Summary

38 Request a job control action.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Job_Control(const pmix_proc_t targets[], size_t ntargs,  
4                   const pmix_info_t directives[], size_t ndirs,  
5                   pmix_info_t *results[], size_t *nresults);
```

C

6 IN **targets**

7 Array of proc structures (array of handles)

8 IN **ntargs**

9 Number of elements in the *targets* array (integer)

10 IN **directives**

11 Array of info structures (array of handles)

12 IN **ndirs**

13 Number of elements in the *directives* array (integer)

14 INOUT **results**

15 Address where a pointer to an array of **pmix_info_t** containing the results of the request
16 can be returned (memory reference)

17 INOUT **nresults**

18 Address where the number of elements in *results* can be returned (handle)

19 Returns one of the following:

- 20 • **PMIX_SUCCESS**, indicating that the request was processed by the host environment and
21 returned *success*. Details of the result will be returned in the *results* array
22 • a PMIx error constant indicating either an error in the input or that the request was refused

23 **Required Attributes**

24 PMIx libraries are not required to directly support any attributes for this function. However, any
25 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
26 required to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
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

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.pvnimg" (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 12.2.2 PMIx_Job_control_nb

22 Summary

23 Request a job control action.

24 Format

C

```

25     pmix_status_t
26     PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargs,
27                           const pmix_info_t directives[], size_t ndirs,
28                           pmix_info_cfunc_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 Returns one of the following:
14
15  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
16    will be returned in the provided cbfunc. Note that the library must not invoke the callback
17    function prior to returning from the API.
18
19  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
20    returned success - the cbfunc will not be called
21
22  • a PMIx error constant indicating either an error in the input or that the request was immediately
23    processed and failed - the cbfunc will not be called

```

Required Attributes

```

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

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.pvnimg" (char*)
7     Name of the image that is to be provisioned.
8 PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
9     Indicate that the job can be pre-empted.
```



10 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 12.2.3 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 12.2.4 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.

12.2.5 Job control attributes

Attributes used to request control operations on an executing application - these are values passed to the job control APIs and are not accessed using the `PMIx_Get` API.

```
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_PREEEMPTIBLE "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.lvttop" (bool)
10    When recursively cleaning subdirectories, do not remove the top-level directory (the one
11      given in the cleanup request).

```

12.3 Process and Job Monitoring

In addition to external faults, a common problem encountered in HPC applications is a failure to make progress due to some internal conflict in the computation. These situations can result in a significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the job. Various watchdog methods have been developed for detecting this situation, including requiring a periodic “heartbeat” from the application and monitoring a specified file for changes in size and/or modification time.

The following APIs allow applications to request monitoring, directing what is to be monitored, the frequency of the associated check, whether or not the application is to be notified (via the event notification subsystem) of stall detection, and other characteristics of the operation.

12.3.1 PMIx_Process_monitor

Summary

Request that application processes be monitored.

Format

```

PMIx v3.0
pmix_status_t
PMIx_Process_monitor(const pmix_info_t *monitor,
                      pmix_status_t error,
                      const pmix_info_t directives[], size_t ndirs,
                      pmix_info_t *results[], size_t *nresults);

IN  monitor
     info (handle)
IN  error
     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    Returns one of the following:
11      • PMIX_SUCCESS, indicating that the request was processed and returned success. Details of the
12        result will be returned in the results array
13      • a PMIx error constant indicating either an error in the input or that the request was refused

```

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:

```

19   PMIX_MONITOR_ID "pmix.monitor.id" (char*)
20     Provide a string identifier for this request.

21   PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)
22     Identifier to be canceled (NULL means cancel all monitoring for this process).

23   PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)
24     The application desires to control the response to a monitoring event - i.e., the application is
25       requesting that the host environment not take immediate action in response to the event (e.g.,
26         terminating the job).

27   PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)
28     Register to have the PMIx server monitor the requestor for heartbeats.

29   PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
30     Time in seconds before declaring heartbeat missed.

31   PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
32     Number of heartbeats that can be missed before generating the event.

33   PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
34     Register to monitor file for signs of life.

35   PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
36     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 PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)
10    Send heartbeat to local PMIx server.
```



11 **Description**

12 Request that application processes be monitored via several possible methods. For example, that
13 the server monitor this process for periodic heartbeats as an indication that the process has not
14 become “wedged”. When a monitor detects the specified alarm condition, it will generate an event
15 notification using the provided error code and passing along any available relevant information. It
16 is up to the caller to register a corresponding event handler.

17 The *monitor* argument is an attribute indicating the type of monitor being requested. For example,
18 **PMIX_MONITOR_FILE** to indicate that the requestor is asking that a file be monitored.

19 The *error* argument is the status code to be used when generating an event notification alerting that
20 the monitor has been triggered. The range of the notification defaults to
21 **PMIX_RANGE_NAMESPACE**. This can be changed by providing a **PMIX_RANGE** directive.

22 The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency
23 of checking to be done

24 The returned *status* indicates whether or not the request was granted, and information as to the
25 reason for any denial of the request shall be returned in the *results* array.

26 **12.3.2 PMIx_Process_monitor_nb**

27 **Summary**

28 Request that application processes be monitored.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Process_monitor_nb(const pmix_info_t *monitor,  
4                               pmix_status_t error,  
5                               const pmix_info_t directives[],  
6                               size_t ndirs,  
7                               pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

8 IN monitor
9 info (handle)
10 IN error
11 status (integer)
12 IN directives
13 Array of info structures (array of handles)
14 IN ndirs
15 Number of elements in the *directives* array (integer)
16 IN cbfunc
17 Callback function **pmix_info_cbfunc_t** (function reference)
18 IN cbdata
19 Data to be passed to the callback function (memory reference)

20 Returns one of the following:

- 21 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
22 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
23 function prior to returning from the API.
- 24 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
25 returned *success* - the *cbfunc* will *not* be called.
- 26 • a PMIx error constant indicating either an error in the input or that the request was immediately
27 processed and failed - the *cbfunc* will *not* be called.

28 Optional Attributes

29 The following attributes may be implemented by a PMIx library or by the host environment. If
30 supported by the PMIx server library, then the library must not pass the supported attributes to the
31 host environment. All attributes not directly supported by the server library must be passed to the
32 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:

```
33 PMIX_MONITOR_ID "pmix.monitor.id" (char*)  
34     Provide a string identifier for this request.  
35 PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)
```

```

1 Identifier to be canceled (NULL means cancel all monitoring for this process).
2 PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)
3 The application desires to control the response to a monitoring event - i.e., the application is
4 requesting that the host environment not take immediate action in response to the event (e.g.,
5 terminating the job).

6 PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)
7 Register to have the PMIx server monitor the requestor for heartbeats.

8 PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
9 Time in seconds before declaring heartbeat missed.

10 PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
11 Number of heartbeats that can be missed before generating the event.

12 PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
13 Register to monitor file for signs of life.

14 PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
15 Monitor size of given file is growing to determine if the application is running.

16 PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
17 Monitor time since last access of given file to determine if the application is running.

18 PMIX_MONITOR_FILE MODIFY "pmix.monitor.fmod" (char*)
19 Monitor time since last modified of given file to determine if the application is running.

20 PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
21 Time in seconds between checking the file.

22 PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
23 Number of file checks that can be missed before generating the event.

24 PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)
25 Send heartbeat to local PMIx server.

```



26 **Description**

27 Non-blocking form of the **PMIx_Process_monitor** API. The *cbfunc* function provides a
28 *status* to indicate whether or not the request was granted, and to provide some information as to the
29 reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

30 **12.3.3 PMIx_Heartbeat**

31 **Summary**

32 Send a heartbeat to the PMIx server library

1 **Format**

2 **PMIx_Heartbeat () ;**

3 **Description**

4 A simplified macro wrapping **PMIx_Process_monitor_nb** that sends a heartbeat to the PMIx
5 server library.

6 **12.3.4 Monitoring events**

7 The following monitoring events may be available for registration, depending upon implementation
8 and host environment support:

9 **PMIX_MONITOR_HEARTBEAT_ALERT** Heartbeat failed to arrive within specified window.

10 The process that triggered this alert will be identified in the event.

11 **PMIX_MONITOR_FILE_ALERT** File failed its monitoring detection criteria. The file that
12 triggered this alert will be identified in the event.

13 **12.3.5 Monitoring attributes**

14 Attributes used to control monitoring of an executing application- these are values passed to the
15 **PMIx_Process_monitor_nb** API and are not accessed using the **PMIx_Get** API.

16 **PMIX_MONITOR_ID "pmix.monitor.id" (char*)**

17 Provide a string identifier for this request.

18 **PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)**

19 Identifier to be canceled (**NULL** means cancel all monitoring for this process).

20 **PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)**

21 The application desires to control the response to a monitoring event - i.e., the application is
22 requesting that the host environment not take immediate action in response to the event (e.g.,
23 terminating the job).

24 **PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)**

25 Register to have the PMIx server monitor the requestor for heartbeats.

26 **PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)**

27 Send heartbeat to local PMIx server.

28 **PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)**

29 Time in seconds before declaring heartbeat missed.

30 **PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)**

31 Number of heartbeats that can be missed before generating the event.

32 **PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)**

33 Register to monitor file for signs of life.

34 **PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)**

35 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.
```

12.4 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.

12.4.1 PMIx_Log

Summary

Log data to a data service.

Format

```
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 Return codes are one of the following:

30 **PMIX_SUCCESS** The logging request was successful.
31 **PMIX_ERR_BAD_PARAM** The logging request contains at least one incorrect entry.
32 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation or host environment does not support
33 this function.
34 other appropriate PMIx error code

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.1sys" (**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.tstmp" (**time_t**)

31 Timestamp for log report.

32 **PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstmp" (**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.

▼ Advice to users ▼

29 It is strongly recommended that the **PMIx_Log** API not be used by applications for streaming data
30 as it is not a “performant” transport and can perturb the application since it involves the local PMIx
31 server and host SMS daemon. Note that a return of **PMIX_SUCCESS** only denotes that the data
32 was successfully handed to the appropriate system call (for local channels) or the host environment
33 and does not indicate receipt at the final destination.

12.4.2 PMIx_Log_nb

2 Summary

3 Log data to a data service.

4 PMIx v2.0 Format

```
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);
```

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 **PMIX_SUCCESS** The logging request is valid and is being processed. The resulting status from
23 the operation will be provided in the callback function. Note that the library must not invoke
24 the callback function 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_BAD_PARAM** The logging request contains at least one incorrect entry that prevents
28 it from being processed. The callback function will not be called.

29 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function. The
30 callback function will not be called.

31 other appropriate PMIx error code - the callback function will not be called.

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.1sys" (**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.tstmp" (**time_t**)

31 Timestamp for log report.

32 **PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstmp" (**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

1 It is strongly recommended that the **PMIx_Log_nb** API not be used by applications for streaming
2 data as it is not a “performant” transport and can perturb the application since it involves the local
3 PMIx server and host SMS daemon. Note that a return of **PMIX_SUCCESS** only denotes that the
4 data was successfully handed to the appropriate system call (for local channels) or the host
5 environment and does not indicate receipt at the final destination.

12.4.3 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.

```
9  PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
10 ID of source of the log request.
11 PMIX_LOG_STDERR "pmix.log.stderr" (char*)
12 Log string to stderr.
13 PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
14 Log string to stdout.
15 PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
16 Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
17 otherwise to local syslog.
18 PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)
19 Log data to local syslog. Defaults to ERROR priority.
20 PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
21 Forward data to system “gateway” and log msg to that syslog. Defaults to ERROR priority.
22 PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)
23 Syslog priority level.
24 PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
25 Timestamp for log report.
26 PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)
27 Generate timestamp for log.
28 PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
29 Label the output stream with the channel name (e.g., “stdout”).
30 PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
31 Print timestamp in output string.
32 PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
33 Print the output stream in XML format.
34 PMIX_LOG_ONCE "pmix.log.once" (bool)
35 Only log this once with whichever channel can first support it, taking the channels in priority
36 order.
37 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 13

Process Sets and Groups

1 PMIx supports two slightly related, but functionally different concepts known as *process sets* and
2 *process groups*. This chapter defines these two concepts and describes how they are utilized, along
3 with their corresponding APIs.

13.1 Process Sets

5 A PMIx *Process Set* is a user-provided or host environment assigned label associated with a given
6 set of application processes. Processes can belong to multiple process *sets* at a time. Users may
7 define a PMIx process set at time of application execution. For example, if using the command line
8 parallel launcher "prun", one could specify process sets as follows:

9 \$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp

10 In this example, the processes in the first application will be labeled with a **PMIX_PSET_NAMES**
11 attribute with a value of *ocean* while those in the second application will be labeled with an *ice*
12 value. During the execution, application processes could lookup the process set attribute for any
13 process using **PMIx_Get**. Alternatively, other executing applications could utilize the
14 **PMIx_Query_info** APIs to obtain the number of declared process sets in the system, a list of
15 their names, and other information about them. In other words, the *process set* identifier provides a
16 label by which an application can derive information about a process and its application - it does
17 *not*, however, confer any operational function.

18 Host environments can create or delete process sets at any time through the
19 **PMIx_server_define_process_set** and **PMIx_server_delete_process_set**
20 APIs. PMIx servers shall notify all local clients of process set operations via the
21 **PMIX_PROCESS_SET_DEFINE** or **PMIX_PROCESS_SET_DELETE** events.

22 Process *sets* differ from process *groups* in several key ways:

- 23 • 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.

13.1.1 Process Set Constants

PMIx v4.0

The PMIx server is required to send a notification to all local clients upon creation or deletion of process sets. Client processes wishing to receive such notifications must register for the corresponding event:

PMIX_PROCESS_SET_DEFINE The host environment has defined a new process set - the event will include the process set name (**PMIX_PSET_NAME**) and the membership (**PMIX_PSET_MEMBERS**).

PMIX_PROCESS_SET_DELETE The host environment has deleted a process set - the event will include the process set name (**PMIX_PSET_NAME**).

13.1.2 Process Set Attributes

Several attributes are provided for querying the system regarding process sets using the `PMIx_Query_info` APIs.

```
4 PMIX_QUERY_NUM_PSETS "pmixqry.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 "pmixqry.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 "pmixqry.pmems" (pmix_data_array_t*)
11     Return an array of pmix_proc_t containing the members of the specified process set.
```

The `PMIX_PROCESS_SET_DEFINE` event shall include the name of the newly defined process set and its members: `PMIX_PSET_NAME "pmix.pset.nm" (char*)`

The name of the newly defined process set.

```
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.
```

In addition, a process can request (via `PMIx_Get`) the process sets to which a given process (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.
```

13.2 Process Groups

PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for operational purposes such as passing events or participating in PMIx fence operations. As with processes that assemble via `PMIx_Connect`, each member of the group is provided with both the job-level information of any other namespace represented in the group, and the contact information for all group members.

However, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected* when constructed via `PMIx_Connect`. Thus, *groups* differ from `PMIx_Connect` assemblages in several key areas, as detailed in the following sections.

13.2.1 Relation to the host environment

Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed, the server creates a tracker that associates the specified processes with the user-provided group identifier, and assigns a new *group rank* based on their relative position in the array of processes provided in the call to `PMIx_Group_construct`. Members of the group can subsequently

1 utilize the group identifier in PMIx function calls to address the group's members, using either
2 **PMIX_RANK_WILDCARD** to refer to all of them or the group-level rank of specific members. The
3 PMIx server will translate the specified processes into their RM-assigned identifiers prior to
4 passing the request up to its host. Thus, the host environment has no visibility into the group's
5 existence or membership.

6 In contrast, calls to **PMIx_Connect** are relayed to the host environment. This means that the host
7 RM should treat the failure of any process in the specified assemblage as a reportable event and
8 take appropriate action. However, the environment is not required to define a new identifier for the
9 connected assemblage or any of its member processes, nor does it define a new rank for each
10 process within that assemblage. In addition, the PMIx server does not provide any tracking support
11 for the assemblage. Thus, the caller is responsible for addressing members of the connected
12 assemblage using their RM-provided identifiers.

Advice to users

13 User-provided group identifiers must be distinct from both other group identifiers within the system
14 and namespaces provided by the RM so as to avoid collisions between group identifiers and
15 RM-assigned namespaces. This can usually be accomplished through the use of an
16 application-specific prefix – e.g., “myapp-foo”

17 13.2.2 Construction procedure

18 **PMIx_Connect** calls require that every process call the API before completing – i.e., it is
19 modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a given
20 application thread can only be involved in one connect/accept operation at a time, and is blocked in
21 that operation until all specified processes participate. In addition, there is no provision for
22 replacing processes in the assemblage due to failure to participate, nor a mechanism by which a
23 process might decline participation.

24 In contrast, PMIx Groups are designed to be more flexible in their construction procedure by
25 relaxing these constraints. While a standard blocking form of constructing groups is provided, the
26 event notification system is utilized to provide a designated *group leader* with the ability to replace
27 participants that fail to participate within a given timeout period. This provides a mechanism by
28 which the application can, if desired, replace members on-the-fly or allow the group to proceed
29 with partial membership. In such cases, the final group membership is returned to all participants
30 upon completion of the operation.

31 Additionally, PMIx supports dynamic definition of group membership based on an invite/join
32 model. A process can asynchronously initiate construction of a group of any processes via the
33 **PMIx_Group_invite** function call. Invitations are delivered via a PMIx event (using the
34 **PMIX_GROUP_INVITED** event) to the invited processes which can then either accept or decline
35 the invitation using the **PMIx_Group_join** API. The initiating process tracks responses by
36 registering for the events generated by the call to **PMIx_Group_join**, timeouts, or process

1 terminations, optionally replacing processes that decline the invitation, fail to respond in time, or
2 terminate without responding. Upon completion of the operation, the final list of participants is
3 communicated to each member of the new group.

4 13.2.3 Destruct procedure

5 Members of a PMIx Group may depart the group at any time via the **PMIx_Group_Leave** API.
6 Other members are notified of the departure via the **PMIX_GROUP_LEFT** event to distinguish such
7 events from those reporting process termination. This leaves the remaining members free to
8 continue group operations. The **PMIx_Group_Destruct** operation offers a collective method
9 akin to **PMIx_Disconnect** for deconstructing the entire group.

10 In contrast, processes that assemble via **PMIx_Connect** must all depart the assemblage together –
11 i.e., no member can depart the assemblage while leaving the remaining members in it. Even the
12 non-blocking form of **PMIx_Disconnect** retains this requirement in that members remain a part
13 of the assemblage until all members have called **PMIx_Disconnect_nb**.

14 Note that applications supporting dynamic group behaviors such as asynchronous departure take
15 responsibility for ensuring global consistency in the group definition prior to executing group
16 collective operations - i.e., it is the application's responsibility to either ensure that knowledge of
17 the current group membership is globally consistent across the participants, or to register for
18 appropriate events to deal with the lack of consistency during the operation.

Advice to users

19 The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs
20 must register for the corresponding events. Failure to do so will likely lead to operational failures.
21 Users are recommended to utilize the **PMIX_TIMEOUT** directive (or retain an internal timer) on
22 calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have
23 not registered for required events will never respond.

24 13.2.4 Process Group Events

25 *PMIx v4.0*

26 Asynchronous process group operations rely heavily on PMIx events. The following events have
been defined for that purpose.

27 **PMIX_GROUP_INVITED** The process has been invited to join a PMIx Group - the identifier of
28 the group and the ID's of other invited (or already joined) members will be included in the
29 notification.

30 **PMIX_GROUP_LEFT** A process has asynchronously left a PMIx Group - the process identifier
31 of the departing process will be included in the notification.

32 **PMIX_GROUP_MEMBER_FAILED** A member of a PMIx Group has abnormally terminated
33 (i.e., without formally leaving the group prior to termination) - the process identifier of the
34 failed process will be included in the notification.

```

1   PMIX_GROUP_INVITE_ACCEPTED A process has accepted an invitation to join a PMIx
2     Group - the identifier of the group being joined will be included in the notification.
3   PMIX_GROUP_INVITE_DECLINED A process has declined an invitation to join a PMIx
4     Group - the identifier of the declined group will be included in the notification.
5   PMIX_GROUP_INVITE_FAILED An invited process failed or terminated prior to responding
6     to the invitation - the identifier of the failed process will be included in the notification.
7   PMIX_GROUP_MEMBERSHIP_UPDATE The membership of a PMIx group has changed - the
8     identifiers of the revised membership will be included in the notification.
9   PMIX_GROUP_CONSTRUCT_ABORT Any participant in a PMIx group construct operation
10    that returns PMIX_GROUP_CONSTRUCT_ABORT from the leader failed event handler will
11    cause all participants to receive an event notifying them of that status. Similarly, the leader
12    may elect to abort the procedure by either returning this error code from the handler assigned
13    to the PMIX_GROUP_INVITE_ACCEPTED or PMIX_GROUP_INVITE_DECLINED
14    codes, or by generating an event for the abort code. Abort events will be sent to all invited or
15    existing members of the group.
16   PMIX_GROUP_CONSTRUCT_COMPLETE The group construct operation has completed - the
17     final membership will be included in the notification.
18   PMIX_GROUP_LEADER FAILED The current leader of a group including this process has
19     abnormally terminated - the group identifier will be included in the notification.
20   PMIX_GROUP_LEADER_SELECTED A new leader of a group including this process has been
21     selected - the identifier of the new leader will be included in the notification.
22   PMIX_GROUP_CONTEXT_ID_ASSIGNED A new PGCID has been assigned by the host
23     environment to a group that includes this process - the group identifier will be included in the
24     notification.

```

13.2.5 Process Group Attributes

PMIx v4.0

Attributes for querying the system regarding process groups include:

```

26   PMIX_QUERY_NUM_GROUPS "pmixqry.pgrpnum" (size_t)
27     Return the number of process groups defined in the specified range (defaults to session).
28     OPTIONAL QUALIFERS: PMIX RANGE.
29   PMIX_QUERY_GROUP_NAMES "pmixqry.pgrp" (pmix_data_array_t*)
30     Return a pmix_data_array_t containing an array of string names of the process groups
31     defined in the specified range (defaults to session). OPTIONAL QUALIFERS:
32     PMIX RANGE.
33   PMIX_QUERY_GROUP_MEMBERSHIP
34     "pmixqry.pgrpmems" (pmix_data_array_t*)
35     Return a pmix_data_array_t of pmix_proc_t containing the members of the
36     specified process group. REQUIRED QUALIFIERS: PMIX GROUP ID.
37

```

The following attributes are used as directives in PMIx Group operations:

```
PMIX_GROUP_ID "pmix.grp.id" (char*)
```

1 User-provided group identifier - as the group identifier may be used in PMIx operations, the
 2 user is required to ensure that the provided ID is unique within the scope of the host
 3 environment (e.g., by including some user-specific or application-specific prefix or suffix to
 4 the string).
 5 **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)
 6 This process is the leader of the group.
 7 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)
 8 Participation is optional - do not return an error if any of the specified processes terminate
 9 without having joined. The default is **false**.
 10 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)
 11 Notify remaining members when another member terminates without first leaving the group.
 12 **PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)
 13 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective
 14 operation.
 15 **PMIX_GROUP_MEMBERSHIP** "pmix.grp.mbrs" (pmix_data_array_t*)
 16 Array **pmix_proc_t** identifiers identifying the members of the specified group.
 17 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)
 18 Requests that the RM assign a new context identifier to the newly created group. The
 19 identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
 20 specified in the request. Thus, the value serves as a means of identifying the group within
 21 that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.
 22 **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)
 23 Group operation only involves local processes. PMIx implementations are *required* to
 24 automatically scan an array of group members for local vs remote processes - if only local
 25 processes are detected, the implementation need not execute a global collective for the
 26 operation unless a context ID has been requested from the host environment. This can result
 27 in significant time savings. This attribute can be used to optimize the operation by indicating
 28 whether or not only local processes are represented, thus allowing the implementation to
 29 bypass the scan.

30 The following attributes are used to return information at the conclusion of a PMIx Group
 31 operation and/or in event notifications:

32 **PMIX_GROUP_CONTEXT_ID** "pmix.grp.ctxid" (**size_t**)
 33 Context identifier assigned to the group by the host RM.
 34 **PMIX_GROUP_ENDPT_DATA** "pmix.grp.endpt" (pmix_byte_object_t)
 35 Data collected during group construction to ensure communication between group members
 36 is supported upon completion of the operation.

37 In addition, a process can request (via **PMIx_Get**) the process groups to which a given process
 38 (including itself) belongs:

39 **PMIX_GROUP_NAMES** "pmix.pgrp.nm" (pmix_data_array_t*)

1 Returns an array of **char*** string names of the process groups in which the given process is
2 a member.

3 **13.2.6 PMIx_Group_construct**

4 **Summary**

5 Construct a PMIx process group.

6 **Format**

```
7 pmix_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);
```

14 **IN grp**
15 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group
16 identifier (string)

17 **IN procs**
18 Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes
19 (array of handles)

20 **IN nprocs**
21 Number of elements in the *procs* array (**size_t**)

22 **IN directives**
23 Array of **pmix_info_t** structures (array of handles)

24 **IN ndirs**
25 Number of elements in the *directives* array (**size_t**)

26 **INOUT results**
27 Pointer to a location where the array of **pmix_info_t** describing the results of the
28 operation is to be returned (pointer to handle)

29 **INOUT nresults**
30 Pointer to a **size_t** location where the number of elements in *results* is to be returned
31 (memory reference)

32 Returns one of the following:

- 33 • **PMIX_SUCCESS**, indicating that the request has been successfully completed
- 34 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
35 operation
- 36 • a PMIx error constant indicating either an error in the input or that the request failed to be
37 completed

Required Attributes

1 The following attributes are *required* to be supported by all PMIx libraries that support this
2 operation:

3 **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)

4 This process is the leader of the group.

5 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

6 Participation is optional - do not return an error if any of the specified processes terminate
7 without having joined. The default is **false**.

8 **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)

9 Group operation only involves local processes. PMIx implementations are *required* to
10 automatically scan an array of group members for local vs remote processes - if only local
11 processes are detected, the implementation need not execute a global collective for the
12 operation unless a context ID has been requested from the host environment. This can result
13 in significant time savings. This attribute can be used to optimize the operation by indicating
14 whether or not only local processes are represented, thus allowing the implementation to
15 bypass the scan.

16 **PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)

17 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective
18 operation.

19 Host environments that support this operation are *required* to support the following attributes:

20 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

21 Requests that the RM assign a new context identifier to the newly created group. The
22 identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
23 specified in the request. Thus, the value serves as a means of identifying the group within
24 that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

25 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

26 Notify remaining members when another member terminates without first leaving the group.
27

Optional Attributes

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

29 **PMIX_TIMEOUT** "pmix.timeout" (int)

30 Time in seconds before the specified operation should time out (zero indicating infinite) and
31 return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions
32 caused by multiple layers (client, server, and host) simultaneously timing the operation.

1 Description

2 Construct a new group composed of the specified processes and identified with the provided group
3 identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less
4 than or equal to **PMIX_MAX_NSLEN**. Only characters accepted by standard string comparison
5 functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group
6 construct operations so long as each is provided with a unique group ID. The *directives* array can be
7 used to pass user-level directives regarding timeout constraints and other options available from the
8 PMIx server.

9 If the **PMIX_GROUP_NOTIFY_TERMINATION** attribute is provided and has a value of **true**,
10 then either the construct leader (if **PMIX_GROUP_LEADER** is provided) or all participants who
11 register for the **PMIX_GROUP_MEMBER FAILED** event will receive events whenever a process
12 fails or terminates prior to calling **PMIx_Group_construct** – i.e. if a *group leader* is declared,
13 only that process will receive the event. In the absence of a declared leader, *all* specified group
14 members will receive the event.

15 The event will contain the identifier of the process that failed to join plus any other information that
16 the host RM provided. This provides an opportunity for the leader or the collective members to
17 react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The
18 decision is communicated to the PMIx library in the results array at the end of the event handler.
19 This allows PMIx to properly adjust accounting for procedure completion. When construct is
20 complete, the participating PMIx servers will be alerted to any change in participants and each
21 group member will receive an updated group membership (marked with the
22 **PMIX_GROUP_MEMBERSHIP** attribute) as part of the *results* array returned by this API.

23 Failure of the declared leader at any time will cause a **PMIX_GROUP LEADER FAILED** event to
24 be delivered to all participants so they can optionally declare a new leader. A new leader is
25 identified by providing the **PMIX_GROUP LEADER** attribute in the results array in the return of
26 the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the
27 new leader. Results of the leader selection will be communicated to all participants via a
28 **PMIX_GROUP LEADER SELECTED** event identifying the new leader. If no leader was selected,
29 then the **pmix_info_t** provided to that event handler will include that information so the
30 participants can take appropriate action.

31 Any participant that returns **PMIX_GROUP CONSTRUCT ABORT** from either the
32 **PMIX_GROUP MEMBER FAILED** or the **PMIX_GROUP LEADER FAILED** event handler will
33 cause the construct process to abort, returning from the call with a
34 **PMIX_GROUP CONSTRUCT ABORT** status.

35 If the **PMIX_GROUP NOTIFY TERMINATION** attribute is not provided or has a value of
36 **false**, then the **PMIx_Group_construct** operation will simply return an error whenever a
37 proposed group member fails or terminates prior to calling **PMIx_Group_construct**.

38 Providing the **PMIX_GROUP OPTIONAL** attribute with a value of **true** directs the PMIx library
39 to consider participation by any specified group member as non-required - thus, the operation will
40 return **PMIX_SUCCESS** if all members participate, or **PMIX_ERR_PARTIAL_SUCCESS** if some

1 members fail to participate. The *results* array will contain the final group membership in the latter
2 case. Note that this use-case can cause the operation to hang if the **PMIX_TIMEOUT** attribute is
3 not specified and one or more group members fail to call **PMIx_Group_construct** while
4 continuing to execute. Also, note that no leader or member failed events will be generated during
5 the operation.

6 Processes in a group under construction are not allowed to leave the group until group construction
7 is complete. Upon completion of the construct procedure, each group member will have access to
8 the job-level information of all namespaces represented in the group plus any information posted
9 via **PMIx_Put** (subject to the usual scoping directives) for every group member.

Advice to PMIx library implementers

10 At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related
11 information from each participating namespace plus any information posted by group members via
12 **PMIx_Put** (subject to scoping directives) is available to each member via calls to **PMIx_Get**.

Advice to PMIx server hosts

13 The collective nature of this API generally results in use of a fence-like operation by the backend
14 host environment. Host environments that utilize the array of process participants as a *signature* for
15 such operations may experience potential conflicts should both a **PMIx_Group_construct** and
16 a **PMIx_Fence** operation involving the same participants be simultaneously executed. As PMIx
17 allows for such use-cases, it is therefore the responsibility of the host environment to resolve any
18 potential conflicts.

13.2.7 **PMIx_Group_construct_nb**

Summary

Non-blocking form of **PMIx_Group_construct**.

1 Format

C

```
2 pmix_status_t
3 PMIx_Group_construct_nb(const char grp[],
4                           const pmix_proc_t procs[], size_t nprocs,
5                           const pmix_info_t directives[],
6                           size_t ndirs,
7                           pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

8 IN grp

9 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group
10 identifier (string)

11 IN procs

12 Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes
13 (array of handles)

14 IN nprocs

15 Number of elements in the *procs* array (**size_t**)

16 IN directives

17 Array of **pmix_info_t** structures (array of handles)

18 IN ndirs

19 Number of elements in the *directives* array (**size_t**)

20 IN cbfunc

21 Callback function **pmix_info_cbfunc_t** (function reference)

22 IN cbdata

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

24 Returns one of the following:

- 25 • **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
26 callback function will be executed upon completion of the operation. Note that the library *must*
27 not invoke the callback function prior to returning from the API.
- 28 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
29 returned *success* - the *cbfunc* will *not* be called.
- 30 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
31 will *not* be called.
- 32 • a non-zero PMIx error constant indicating a reason for the request to have been rejected - the
33 *cbfunc* will *not* be called.

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

- 36 • **PMIX_SUCCESS** The operation succeeded and all specified members participated.

- **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function.
 - **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
 - a non-zero PMIx error constant indicating a reason for the request's failure.

Required Attributes

PMIx libraries that choose not to support this operation *must* return **PMIX_ERR_NOT_SUPPORTED** when the function is called.

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)

This process is the leader of the group.

PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is **false**.

PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)

Group operation only involves local processes. PMIx implementations are *required* to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)

Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are *required* to provide the following attributes:

PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, `size_t` value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to `PMIX_RANGE_SESSION`.

PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)

Notify remaining members when another member terminates without first leaving the group.

Optional Attributes

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

2 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

6 **Description**

7 Non-blocking version of the **PMIx_Group_construct** operation. The callback function will be
8 called once all group members have called either **PMIx_Group_construct** or
9 **PMIx_Group_construct_nb**.

10 **13.2.8 PMIx_Group_destruct**

11 **Summary**

12 Destruct a PMIx process group.

13 **Format**

14 **pmix_status_t**
15 **PMIx_Group_destruct**(const char grp[],
16 const pmix_info_t directives[],
17 size_t ndirs);

18 **IN grp**
19 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
20 identifier of the group to be destructed (string)

21 **IN directives**
22 Array of **pmix_info_t** structures (array of handles)

23 **IN ndirs**
24 Number of elements in the *directives* array (**size_t**)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS**, indicating that the request has been successfully completed
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
28 operation
- 29 • a PMIx error constant indicating either an error in the input or that the request failed to be
30 completed

Required Attributes

For implementations and host environments that support the operation, there are no identified required attributes for this API.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Description

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling **PMIx_Group_destruct** or its non-blocking version unless the **PMIX_GROUP_NOTIFY_TERMINATION** attribute was provided (with a value of **false**) at time of group construction. If notification was requested, then the **PMIX_GROUP_MEMBER_FAILED** event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The **PMIx_Group_destruct** operation will subsequently return **PMIX_SUCCESS** when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

Advice to PMIx server hosts

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a **PMIx_Group_destruct** and a **PMIx_Fence** operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

13.2.9 **PMIx_Group_destruct_nb**

Summary

Non-blocking form of **PMIx_Group_destruct**.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Group_destruct_nb(const char grp[],  
4 const pmix_info_t directives[],  
5 size_t ndirs,  
6 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

7 IN grp

8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
9 identifier of the group to be destructed (string)

10 IN directives

11 Array of **pmix_info_t** structures (array of handles)

12 IN ndirs

13 Number of elements in the *directives* array (**size_t**)

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 - result will be returned in the
20 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
21 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_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
25 will *not* be called.
- 26 • a PMIx error constant indicating either an error in the input or that the request was immediately
27 processed and failed - the *cbfunc* will *not* be called.

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

- 30 • **PMIX_SUCCESS** The operation was successfully completed.
- 31 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
32 does not.
- 33 • a non-zero PMIx error constant indicating a reason for the request's failure.

Required Attributes

1 PMIx libraries that choose not to support this operation *must* return
2 **PMIX_ERR_NOT_SUPPORTED** when the function is called. For implementations and host
3 environments that support the operation, there are no identified required attributes for this API.

Optional Attributes

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

5 **PMIX_TIMEOUT "pmix.timeout" (int)**

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

Description

10 Non-blocking version of the **PMIx_Group_destruct** operation. The callback function will be
11 called once all members of the group have executed either **PMIx_Group_destruct** or
12 **PMIx_Group_destruct_nb**.

13 13.2.10 **PMIx_Group_invite**

14 Summary

15 Asynchronously construct a PMIx process group.

1 Format

C

```
2 pmix_status_t
3 PMIx_Group_invite(const char grp[],
4                     const pmix_proc_t procs[], size_t nprocs,
5                     const pmix_info_t directives[], size_t ndirs,
6                     pmix_info_t **results, size_t *nresult);
```

C

7 IN **grp**

8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group
9 identifier (string)

10 IN **procs**

11 Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be
12 invited (array of handles)

13 IN **nprocs**

14 Number of elements in the *procs* array (**size_t**)

15 IN **directives**

16 Array of **pmix_info_t** structures (array of handles)

17 IN **ndirs**

18 Number of elements in the *directives* array (**size_t**)

19 INOUT **results**

20 Pointer to a location where the array of **pmix_info_t** describing the results of the
21 operation is to be returned (pointer to handle)

22 INOUT **nresults**

23 Pointer to a **size_t** location where the number of elements in *results* is to be returned
24 (memory reference)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS**, indicating that the request has been successfully completed.
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
28 operation.
- 29 • a PMIx error constant indicating either an error in the input or that the request failed to be
30 completed.

▼----- Required Attributes -----▼

31 The following attributes are *required* to be supported by all PMIx libraries that support this
32 operation:

33 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

34 Participation is optional - do not return an error if any of the specified processes terminate
35 without having joined. The default is **false**.

36 **PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)

1 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective
2 operation.

3 Host environments that support this operation are *required* to provide the following attributes:

4 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

5 Requests that the RM assign a new context identifier to the newly created group. The
6 identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
7 specified in the request. Thus, the value serves as a means of identifying the group within
8 that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

9 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

10 Notify remaining members when another member terminates without first leaving the group.
11

Optional Attributes

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

13 **PMIX_TIMEOUT** "pmix.timeout" (int)

14 Time in seconds before the specified operation should time out (zero indicating infinite) and
15 return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions
16 caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

17 Explicitly invite the specified processes to join a group. The process making the
18 **PMIx_Group_invite** call is automatically declared to be the *group leader*. Each invited
19 process will be notified of the invitation via the **PMIX_GROUP_INVITED** event - the processes
20 being invited must therefore register for the **PMIX_GROUP_INVITED** event in order to be notified
21 of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of
22 invite versus event registration is required.

23 The invitation event will include the identity of the inviting process plus the name of the group.
24 When ready to respond, each invited process provides a response using either the blocking or
25 non-blocking form of **PMIx_Group_join**. This will notify the inviting process that the
26 invitation was either accepted (via the **PMIX_GROUP_INVITE_ACCEPTED** event) or declined
27 (via the **PMIX_GROUP_INVITE_DECLINED** event). The **PMIX_GROUP_INVITE_ACCEPTED**
28 event is captured by the PMIx client library of the inviting process – i.e., the application itself does
29 not need to register for this event. The library will track the number of accepting processes and
30 alert the inviting process (by returning from the blocking form of **PMIx_Group_invite** or
31 calling the callback function of the non-blocking form) when group construction completes.

32 The inviting process should, however, register for the **PMIX_GROUP_INVITE_DECLINED** if the
33 application allows invited processes to decline the invitation. This provides an opportunity for the
34 application to either invite a replacement, declare “abort”, or choose to remove the declining

1 process from the final group. The inviting process should also register to receive
2 **PMIX_GROUP_INVITE_FAILED** events whenever a process fails or terminates prior to
3 responding to the invitation. Actions taken by the inviting process in response to these events must
4 be communicated at the end of the event handler by returning the corresponding result so that the
5 PMIx library can adjust accordingly.

6 Upon completion of the operation, all members of the new group will receive access to the job-level
7 information of each other's namespaces plus any information posted via **PMIx_Put** by the other
8 members.

9 The inviting process is automatically considered the leader of the asynchronous group construction
10 procedure and will receive all failure or termination events for invited members prior to completion.
11 The inviting process is required to provide a **PMIX_GROUP_CONSTRUCT_COMPLETE** event once
12 the group has been fully assembled – this event is used by the PMIx library as a trigger to release
13 participants from their call to **PMIx_Group_join** and provides information (e.g., the final group
14 membership) to be returned in the *results* array.

15 Failure of the inviting process at any time will cause a **PMIX_GROUP_LEADER_FAILED** event to
16 be delivered to all participants so they can optionally declare a new leader. A new leader is
17 identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of
18 the event handler. Only one process is allowed to return that attribute, declaring itself as the new
19 leader. Results of the leader selection will be communicated to all participants via a
20 **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected,
21 then the status code provided in the event handler will provide an error value so the participants can
22 take appropriate action.

Advice to users

23 Applications are not allowed to use the group in any operations until group construction is
24 complete. This is required in order to ensure consistent knowledge of group membership across all
25 participants.

26 13.2.11 **PMIx_Group_invite_nb**

27 **Summary**

28 Non-blocking form of **PMIx_Group_invite**.

1 **Format**

C

```
2        pmix_status_t  
3        PMIx_Group_invite_nb(const char grp[],  
4                          const pmix_proc_t procs[], size_t nprocs,  
5                          const pmix_info_t directives[], size_t ndirs,  
6                          pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

7 **IN** **grp**

8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group
9 identifier (string)

10 **IN** **procs**

11 Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be
12 invited (array of handles)

13 **IN** **nprocs**

14 Number of elements in the *procs* array (**size_t**)

15 **IN** **directives**

16 Array of **pmix_info_t** structures (array of handles)

17 **IN** **ndirs**

18 Number of elements in the *directives* array (**size_t**)

19 **IN** **cbfunc**

20 Callback function **pmix_info_cbfunc_t** (function reference)

21 **IN** **cbdata**

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

23 Returns one of the following:

- 24 • **PMIX_SUCCESS**, indicating that the request is being processed - result will be returned in the
25 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
26 from the API.
- 27 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
28 returned *success* - the *cbfunc* will *not* be called.
- 29 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
30 will *not* be called.
- 31 • a PMIx error constant indicating either an error in the input or that the request was immediately
32 processed and failed - the *cbfunc* will *not* be called.

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

- 35 • **PMIX_SUCCESS** The operation succeeded and all specified members participated.
- 36 • **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members
37 participated - the final group membership is included in the callback function.

- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
 - a non-zero PMIx error constant indicating a reason for the request's failure.

Required Attributes

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is **false**.

PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)

Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are *required* to provide the following attributes:

PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)

Notify remaining members when another member terminates without first leaving the group.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Description

Non-blocking version of the `PMIx_Group_invite` operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either `PMIx_Group_join` or `PMIx_Group_join_nb`.

13.2.12 PMIx_Group_join

2 Summary

3 Accept an invitation to join a PMIx process group.

4 Format

```
5 pmix_status_t  
6 PMIx_Group_join(const char grp[],  
7                 const pmix_proc_t *leader,  
8                 pmix_group_opt_t opt,  
9                 const pmix_info_t directives[], size_t ndirs,  
10                pmix_info_t **results, size_t *nresult);
```

11 IN grp

12 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group
13 identifier (string)

14 IN leader

15 Process that generated the invitation (handle)

16 IN opt

17 Accept or decline flag ([pmix_group_opt_t](#))

18 IN directives

19 Array of [pmix_info_t](#) structures (array of handles)

20 IN ndirs

21 Number of elements in the *directives* array (**size_t**)

22 INOUT results

23 Pointer to a location where the array of [pmix_info_t](#) describing the results of the
24 operation is to be returned (pointer to handle)

25 INOUT nresults

26 Pointer to a **size_t** location where the number of elements in *results* is to be returned
27 (memory reference)

28 Returns one of the following:

- 29 • **PMIX_SUCCESS**, indicating that the request has been successfully completed.
- 30 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
31 operation.
- 32 • a PMIx error constant indicating either an error in the input or that the request failed to be
33 completed.

Required Attributes

34 There are no identified required attributes for implementers.

Optional Attributes

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

2 **PMIX_TIMEOUT "pmix.timeout" (int)**

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

Description

6 Respond to an invitation to join a group that is being asynchronously constructed. The process must
7 have registered for the **PMIX_GROUP_INVITED** event in order to be notified of the invitation.
8 When called, the event information will include the **pmix_proc_t** identifier of the process that
9 generated the invitation along with the identifier of the group being constructed. When ready to
10 respond, the process provides a response using either form of **PMIx_Group_join**.
11

Advice to users

12 Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the
13 blocking form of this call unless it first “thread shifts” out of the handler and into its own thread
14 context. Likewise, while it is safe to call the non-blocking form of the API from the event handler,
15 the process *must not* block in the handler while waiting for the callback function to be called.

16 Calling this function causes the inviting process (aka the *group leader*) to be notified that the
17 process has either accepted or declined the request. The blocking form of the API will return once
18 the group has been completely constructed or the group’s construction has failed (as described
19 below) – likewise, the callback function of the non-blocking form will be executed upon the same
20 conditions.

21 Failure of the leader during the call to **PMIx_Group_join** will cause a
22 **PMIX_GROUP_LEADER_FAILED** event to be delivered to all invited participants so they can
23 optionally declare a new leader. A new leader is identified by providing the
24 **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one
25 process is allowed to return that attribute, declaring itself as the new leader. Results of the leader
26 selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED**
27 event identifying the new leader. If no leader was selected, then the status code provided in the
28 event handler will provide an error value so the participants can take appropriate action.

29 Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from the leader failed event
30 handler will cause all participants to receive an event notifying them of that status. Similarly, the
31 leader may elect to abort the procedure by either returning **PMIX_GROUP_CONSTRUCT_ABORT**
32 from the handler assigned to the **PMIX_GROUP_INVITE_ACCEPTED** or
33 **PMIX_GROUP_INVITE_DECLINED** codes, or by generating an event for the abort code. Abort
34 events will be sent to all invited participants.

13.2.13 PMIx_Group_join_nb

2 Summary

3 Non-blocking form of [PMIx_Group_join](#)

4 Format

```
5 pmix_status_t  
6 PMIx_Group_join_nb(const char grp[],  
7                     const pmix_proc_t *leader,  
8                     pmix_group_opt_t opt,  
9                     const pmix_info_t directives[], size_t ndirs,  
10                    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

11 IN grp

12 NULL-terminated character array of maximum size [PMIX_MAX_NSLEN](#) containing the group
13 identifier (string)

14 IN leader

15 Process that generated the invitation (handle)

16 IN opt

17 Accept or decline flag ([pmix_group_opt_t](#))

18 IN directives

19 Array of [pmix_info_t](#) structures (array of handles)

20 IN ndirs

21 Number of elements in the *directives* array ([size_t](#))

22 IN cbfunc

23 Callback function [pmix_info_cbfunc_t](#) (function reference)

24 IN cbdata

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

26 Returns one of the following:

- 27 • [PMIX_SUCCESS](#), indicating that the request is being processed - result will be returned in the
28 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
29 from the API.
- 30 • [PMIX_OPERATION_SUCCEEDED](#), indicating that the request was immediately processed and
31 returned *success* - the *cbfunc* will *not* be called.
- 32 • [PMIX_ERR_NOT_SUPPORTED](#) The PMIx library does not support this operation - the *cbfunc*
33 will *not* be called.
- 34 • a PMIx error constant indicating either an error in the input or that the request was immediately
35 processed and failed - the *cbfunc* will *not* be called.

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

- 3 • **PMIX_SUCCESS** The operation succeeded and group membership is in the callback function
4 parameters.
- 5 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
6 does not.
- 7 • a non-zero PMIx error constant indicating a reason for the request's failure.

▼----- Required Attributes -----▼

8 There are no identified required attributes for implementers.
▲-----

▼----- Optional Attributes -----▼

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

10 **PMIX_TIMEOUT "pmix.timeout" (int)**

11 Time in seconds before the specified operation should time out (zero indicating infinite) and
12 return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions
13 caused by multiple layers (client, server, and host) simultaneously timing the operation.
▲-----

14 **Description**

15 Non-blocking version of the **PMIx_Group_join** operation. The callback function will be called
16 once all invited members of the group (or their substitutes) have executed either
17 **PMIx_Group_join** or **PMIx_Group_join_nb**.

18 **13.2.13.1 Group accept/decline directives**

19 *PMIx v4.0*

20 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:

21 **PMIX_GROUP_DECLINE** Decline the invitation.

22 **PMIX_GROUP_ACCEPT** Accept the invitation.

23 **13.2.14 PMIx_Group_leave**

24 **Summary**

25 Leave a PMIx process group.

1 **Format**

C

```
2      pmix_status_t  
3      PMIx_Group_leave(const char grp[],  
4                    const pmix_info_t directives[],  
5                    size_t ndirs);
```

C

6 **IN grp**

7 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group
8 identifier (string)

9 **IN directives**

10 Array of **pmix_info_t** structures (array of handles)

11 **IN ndirs**

12 Number of elements in the *directives* array (**size_t**)

13 Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server.
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation.
- a PMIx error constant indicating either an error in the input or that the request is unsupported.

▼----- **Required Attributes** -----▼

18 There are no identified required attributes for implementers.
▲-----

19 **Description**

20 Calls to **PMIx_Group_leave** (or its non-blocking form) will cause a **PMIX_GROUP_LEFT**
21 event to be generated notifying all members of the group of the caller's departure. The function will
22 return (or the non-blocking function will execute the specified callback function) once the event has
23 been locally generated and is not indicative of remote receipt.

▼----- **Advice to users** -----▼

24 The **PMIx_Group_leave** API is intended solely for asynchronous departures of individual
25 processes from a group as it is not a scalable operation – i.e., when a process determines it should
26 no longer be a part of a defined group, but the remainder of the group retains a valid reason to
27 continue in existence. Developers are advised to use **PMIx_Group_destruct** (or its
28 non-blocking form) for all other scenarios as it represents a more scalable operation.

13.2.15 PMIx_Group_leave_nb

2 Summary

3 Non-blocking form of [PMIx_Group_leave](#).

4 Format

```
5 pmix_status_t
6 PMIx_Group_leave_nb(const char grp[],           C
7             const pmix_info_t directives[],           C
8             size_t ndirs,                         C
9             pmix_op_cbfunc_t cbfunc,               C
10            void *cbdata);
```

11 IN grp

12 NULL-terminated character array of maximum size [PMIX_MAX_NSLEN](#) containing the group
13 identifier (string)

14 IN directives

15 Array of [pmix_info_t](#) structures (array of handles)

16 IN ndirs

17 Number of elements in the *directives* array ([size_t](#))

18 IN cbfunc

19 Callback function [pmix_op_cbfunc_t](#) (function reference)

20 IN cbdata

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

22 Returns one of the following:

- 23 • [PMIX_SUCCESS](#), indicating that the request is being processed - result will be returned in the
24 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
25 from the API.
- 26 • [PMIX_OPERATION_SUCCEEDED](#), indicating that the request was immediately processed and
27 returned *success* - the *cbfunc* will *not* be called.
- 28 • [PMIX_ERR_NOT_SUPPORTED](#) The PMIx library does not support this operation - the *cbfunc*
29 will *not* be called.
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately
31 processed and failed - the *cbfunc* will *not* be called.

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

- 34 • [PMIX_SUCCESS](#) The operation succeeded - i.e., the [PMIX_GROUP_LEFT](#) event was generated.

- **PMIX_ERR_NOT_SUPPORTED** While the PMIx library supports this operation, the host RM does not.
 - a non-zero PMIx error constant indicating a reason for the request's failure.

Required Attributes

There are no identified required attributes for implementers.

Description

Non-blocking version of the [PMIx_Group_leave](#) operation. The callback function will be called once the event has been locally generated and is not indicative of remote receipt.

CHAPTER 14

Fabric Support Definitions

As the drive for performance continues, interest has grown in scheduling algorithms that take into account network locality of the allocated resources and in optimizing collective communication patterns by structuring them to follow fabric topology. In addition, concerns over the time required to initiate execution of parallel applications and enable communication across them have grown as the size of those applications extends into the hundreds of thousands of individual processes spanning tens of thousands of nodes.

PMIx supports the communication part of these efforts by defining data types and attributes by which fabric endpoints and coordinates for processes and devices can be obtained from the host environment. When used in conjunction with other PMIx methods described in Chapter 16, this results in the ability of a process to obtain the fabric endpoint and coordinate of all other processes without incurring additional overhead associated with a global exchange of that information. This includes:

- Defining several interfaces specifically intended to support WLMs by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric.
- Supporting hierarchical collective operations by providing the fabric coordinates for all devices on participating nodes as well as a list of the peers sharing each fabric switch. This enables one, for example, to aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches based on detailed knowledge of the fabric location of each participant.
- Enabling the "*instant on*" paradigm to mitigate the scalable launch problem by providing each process with a rich set of information about the environment and the application, including everything required for communication between peers within the application, at time of process start of execution.

Meeting these needs in the case where only a single fabric device exists on each node is relatively straightforward - PMIx and the host environment provide a single endpoint for each process plus a coordinate for the device on each node, and there is no uncertainty regarding the endpoint each process will use. Extending this to the multiple device per node case is more difficult as the choice of endpoint by any given process cannot be known in advance, and questions arise regarding reachability between devices on different nodes. Resolving these ambiguities without requiring a global operation requires that PMIx provide both (a) an endpoint for each application process on each of its local devices; and (b) the fabric coordinates of all remote and local devices on participating nodes. It also requires that each process open all of its assigned endpoints as the endpoint selected for contact by a remote peer cannot be known in advance.

1 While these steps ensure the ability of a process to connect to a remote peer, it leaves unanswered
2 the question of selecting the *preferred* device for that communication. If multiple devices are
3 present on a node, then the application can benefit from having each process utilize its "closest"
4 fabric device (i.e., the device that minimizes the communication distance between the process'
5 location and that device) for messaging operations. In some cases, messaging libraries prefer to
6 also retain the ability to use non-nearest devices, prioritizing the devices based on distance to
7 support multi-device operations (e.g., for large message transmission in parallel).

8 PMIx supports this requirement by providing the array of process-to-device distance information
9 for each process and local fabric device at start of execution. Both minimum and maximum
10 distances are provided since a single process can occupy multiple processor locations. In addition,
11 since processes can relocate themselves by changing their processor bindings, PMIx provides an
12 API that allows the process to dynamically request an update to its distance array.

13 However, while these measures assist a process in selecting its own best endpoint, they do not
14 resolve the uncertainty over the choice of preferred device by a remote peer. There are two methods
15 by which this ambiguity can be resolved:

- 16 a) A process can select a remote endpoint to use based on its own preferred device and reachability
17 of the peer's remote devices. Once the initial connection has been made, the two processes can
18 exchange information and mutually determine their desired communication path going forward.
- 19 b) The application can use knowledge of both the local and remote distance arrays to compute the
20 best communication path and establish that connection. In some instances (e.g., a homogeneous
21 system), a PMIx server may provide distance information for both local and remote devices.
22 Alternatively, when this isn't available, an application can opt to collect the information using
23 the **PMIX_COLLECT_GENERATED_JOB_INFO** with the **PMIx_Fence** API, or can obtain it
24 on a one peer-at-a-time basis using the **PMIx_Get** API on systems where the host environment
25 supports the *Direct Mode* operation.

26 Information on fabric coordinates, endpoints, and device distances are provided as *reserved keys* as
27 detailed in Chapter 6 - i.e., they are to be available at client start of execution and are subject to the
28 retrieval rules of Section 6.2. Examples for retrieving fabric-related information include retrieval of:

- 29 • An array of information on fabric devices for a node by passing **PMIX_FABRIC_DEVICES** as
30 the key to **PMIx_Get** along with the **PMIX_HOSTNAME** of the node as a directive
- 31 • An array of information on a specific fabric device by passing **PMIX_FABRIC_DEVICE** as the
32 key to **PMIx_Get** along with the **PMIX_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.

- **PMIX_FABRIC_DEVICE_MTU** "pmix.fabdev.mtu" (**size_t**)
The maximum transfer unit of link level frames or packets, in bytes.
- **PMIX_FABRIC_DEVICE_SPEED** "pmix.fabdev.speed" (**size_t**)
The active link data rate, given in bits per second.
- **PMIX_FABRIC_DEVICE_STATE** "pmix.fabdev.state" (**pmix_link_state_t**)
The last available physical port state for the specified device. Possible values are
PMIX_LINK_STATE_UNKNOWN, **PMIX_LINK_DOWN**, and **PMIX_LINK_UP**, to indicate if the port state is unknown or not applicable (unknown), inactive (down), or active (up).
- **PMIX_FABRIC_DEVICE_TYPE** "pmix.fabdev.type" (**string**)
Specifies the type of fabric interface currently active on the device, such as Ethernet or InfiniBand.

The remainder of this chapter details the events, data types, attributes, and APIs associated with fabric-related operations.

14.1 Fabric Support Events

The following events are defined for use in fabric-related operations.

- **PMIX_FABRIC_UPDATE_PENDING** The PMIx server library has been alerted to a change in the fabric that requires updating of one or more registered **pmix_fabric_t** objects.
- **PMIX_FABRIC_UPDATED** The PMIx server library has completed updating the entries of all affected **pmix_fabric_t** objects registered with the library. Access to the entries of those objects may now resume.
- **PMIX_FABRIC_UPDATE_ENDPOINTS** Endpoint assignments have been updated, usually in response to migration or restart of a process. Clients should use **PMIx_Get** to update any internally cached connections.

14.2 Fabric Support Datatypes

Several datatype definitions have been created to support fabric-related operations and information.

14.2.1 Fabric Endpoint Structure

The **pmix_endpoint_t** structure contains an assigned endpoint for a given fabric device.

PMIx v4.0

```
1     typedef struct pmix_endpoint {  
2         char *uuid;  
3         char *osname;  
4         pmix_byte_object_t endpt;  
5     } pmix_endpoint_t;
```

6 The *uuid* field contains the UUID of the fabric device, the *osname* is the local operating system's
7 name for the device, and the *endpt* field contains a fabric vendor-specific object identifying the
8 communication endpoint assigned to the process.

14.2.2 Fabric endpoint support macros

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

11 Initialize the endpoint structure

12 Initialize the `pmix_endpoint_t` fields.
PMIx v4.0

```
13 PMIX_ENDPOINT_CONSTRUCT(m)
```

14 IN m

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

16 Destruct the endpoint structure

17 Destruct the `pmix_endpoint_t` fields.
PMIx v4.0

```
18 PMIX_ENDPOINT_DESTRUCT(m)
```

19 IN m

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

21 Create an endpoint array

22 Allocate and initialize a `pmix_endpoint_t` array.
PMIx v4.0

```
23 PMIX_ENDPOINT_CREATE(m, n)
```

24 INOUT m

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

27 IN n

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

1 **Release an endpoint array**
2 Release an array of `pmix_endpoint_t` structures.

PMIx v4.0

C

3 `PMIX_ENDPOINT_FREE (m, n)`

C

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

14.2.3 Fabric Coordinate Structure

The `pmix_coord_t` structure describes the fabric coordinates of a specified device in a given view.

```
11       typedef struct pmix_coord {  
12           pmix_coord_view_t view;  
13           uint32_t *coord;  
14           size_t dims;  
15        } pmix_coord_t;
```

All coordinate values shall be expressed as unsigned integers due to their units being defined in fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

Advice to PMIx library implementers

Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the coordinate data is to be stored within the PMIx library. Implementers are free to store the coordinate in whatever format they choose.

A fabric coordinate is associated with a given fabric device and must be unique within a given view. Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates are logically grouped within the *node* realm (as described in Section 6.1) and can be retrieved per the rules detailed in Section 6.1.5.

14.2.4 Fabric coordinate support macros

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 
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.
8 
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.
13 
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.
20 
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`)

14.2.5 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.

14.2.6 Fabric geometry support macros

21 The following macros are provided to support the `pmix_geometry_t` structure.

22 Initialize the geometry structure

23 Initialize the `pmix_geometry_t` fields.

PMIx v4.0

```
24     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)

```

14.2.7 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 14.2.8 Fabric Link State

13 The **pmix_link_state_t** is a **uint32_t** type for fabric link states.

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 14.2.9 Fabric Operation Constants

PMIx v4.0

22 The **pmix_fabric_operation_t** data type is an enumerated type for specifying fabric
 23 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.

14.2.10 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
5  PMIX_FABRIC_GROUPS "pmix.fab.grp" (string)
6      A string delineating the group membership of nodes in the overall system, where each fabric
7      group consists of the group number followed by a colon and a comma-delimited list of nodes
8      in that group, with the groups delimited by semi-colons (e.g.,
9      0:node000,node002,node004,node006;1:node001,node003,
10     node005,node007)
11
12 PMIX_FABRIC_DIMS "pmix.fab.dims" (uint32_t)
13     Number of dimensions in the specified fabric plane/view. If no plane is specified in a
14     request, then the dimensions of all planes in the overall system will be returned as a
15     pmix_data_array_t containing an array of uint32_t values. Default is to provide
16     dimensions in logical view.
17
18 PMIX_FABRIC_PLANE "pmix.fab.plane" (string)
19     ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
20     for information, specifies the plane whose information is to be returned. When used directly
21     as a key in a request, returns a pmix_data_array_t of string identifiers for all fabric
22     planes in the overall system.
23
24 PMIX_FABRIC_SHAPE "pmix.fab.shape" (pmix_data_array_t*)
25     The size of each dimension in the specified fabric plane/view, returned in a
26     pmix_data_array_t containing an array of uint32_t values. The size is defined as
27     the number of elements present in that dimension - e.g., the number of devices in one
28     dimension of a physical view of a fabric plane. If no plane is specified, then the shape of
29     each plane in the overall system will be returned in a pmix_data_array_t array where
30     each element is itself a two-element array containing the PMIX_FABRIC_PLANE followed
31     by that plane's fabric shape. Default is to provide the shape in logical view.
32
33 PMIX_FABRIC_SHAPE_STRING "pmix.fab.shapestr" (string)
34     Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the
35     shape of each plane in the overall system will be returned in a pmix_data_array_t array
36     where each element is itself a two-element array containing the PMIX_FABRIC_PLANE
37     followed by that plane's fabric shape string. Default is to provide the shape in logical view.

While unusual due to scaling issues, implementations may include an array of
PMIX_FABRIC_DEVICE elements describing the device information for each device in the
overall system. Each element shall contain a pmix_data_array_t of pmix_info_t values
describing the device. Each array may contain one or more of the following (ordering is arbitrary):
    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.



14.2.10.1 Initialize the fabric structure

2 Initialize the `pmix_fabric_t` fields.

PMIx v4.0

C

3 `PMIX_FABRIC_CONSTRUCT (m)`

C

4 IN `m`

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

14.3 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.grp" (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 14.4 Fabric Support Functions

23 The following APIs allow the WLM to request specific services from the fabric subsystem via the
24 PMIx library.

Advice to PMIx server hosts

25 Due to their high cost in terms of execution, memory consumption, and interactions with other
26 SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation
27 of these APIs be restricted to a single PMIx server in a system that is supporting the SMS
28 component responsible for the scheduling of allocations (i.e., the system *scheduler*). The
29 **PMIX_SERVER_SCHEDULER** attribute can be used for this purpose to control the execution path.
30 Clients, tools, and other servers utilizing these functions are advised to have their requests
31 forwarded to the server supporting the scheduler using the **pmix_server_fabric_fn_t**
32 server module function, as needed.

14.4.1 PMIx_Fabric_register

Summary

Register for access to fabric-related information.

Format

```
5 pmix_status_t  
6 PMIx_Fabric_register(pmix_fabric_t *fabric,  
7                         const pmix_info_t directives[],  
8                         size_t ndirs);
```

INOUT fabric

address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

IN directives

an optional array of values indicating desired behaviors and/or fabric to be accessed. If `NULL`, then the highest priority available fabric will be used (array of handles)

IN ndirs

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

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

Required Attributes

The following directives are required to be supported by all PMIx libraries to aid users in identifying the fabric whose data is being sought:

PMIX_FABRIC_PLANE "pmix.fab.plane" (string)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in the overall system.

PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)

An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string)

Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

1 **Description**

2 Register for access to fabric-related information, including the communication cost matrix. This
3 call must be made prior to requesting information from a fabric. The caller may request access to a
4 particular fabric using the vendor, type, or identifier, or to a specific *fabric plane* via the
5 **PMIX_FABRIC_PLANE** attribute - otherwise, information for the default fabric will be returned.
6 Upon successful completion of the call, information will have been filled into the fields of the
7 provided *fabric* structure.

8 For performance reasons, the PMIx library does not provide thread protection for accessing the
9 information in the **pmix_fabric_t** structure. Instead, the PMIx implementation shall provide
10 two methods for coordinating updates to the provided fabric information:

- 11 • Users may periodically poll for updates using the **PMIx_Fabric_update** API
- 12 • Users may register for **PMIX_FABRIC_UPDATE_PENDING** events indicating that an update to
13 the cost matrix is pending. When received, users are required to terminate or pause any actions
14 involving access to the cost matrix before returning from the event. Completion of the
15 **PMIX_FABRIC_UPDATE_PENDING** event handler indicates to the PMIx library that the fabric
16 object's entries are available for updating. This may include releasing and re-allocating memory
17 as the number of vertices may have changed (e.g., due to addition or removal of one or more
18 devices). When the update has been completed, the PMIx library will generate a
19 **PMIX_FABRIC_UPDATED** event indicating that it is safe to begin using the updated fabric
20 object(s).

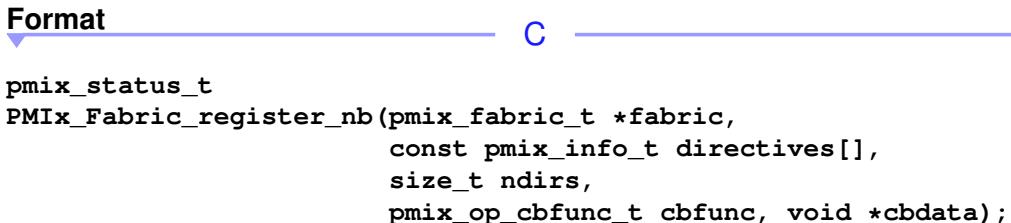
21 There is no requirement that the caller exclusively use either one of these options. For example, the
22 user may choose to both register for fabric update events, but poll for an update prior to some
23 critical operation.

24 **14.4.2 PMIx_Fabric_register_nb**

25 **Summary**

26 Register for access to fabric-related information.

27 **Format**

28 
29 **pmix_status_t**
30 **PMIx_Fabric_register_nb**(**pmix_fabric_t** ***fabric**,
31 **const pmix_info_t** **directives**[],
32 **size_t** **ndirs**,
 pmix_op_cbfunc_t **cfunc**, **void** ***cbdata**);

33 **INPUT fabric**

34 address of a **pmix_fabric_t** (backed by storage). User may populate the "name" field at
35 will - PMIx does not utilize this field (handle)

1 **IN** **directives**
2 an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**,
3 then the highest priority available fabric will be used (array of handles)
4 **IN** **ndirs**
5 Number of elements in the *directives* array (integer)
6 **IN** **cbfunc**
7 Callback function **pmix_op_cbfunc_t** (function reference)
8 **IN** **cbdata**
9 Data to be passed to the callback function (memory reference)

10 >Returns one of the following:

11 • **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
12 callback function will be executed upon completion of the operation. Note that the library must
13 not invoke the callback function prior to returning from the API.

14 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this
15 case, the provided callback function will not be executed

16 **Description**

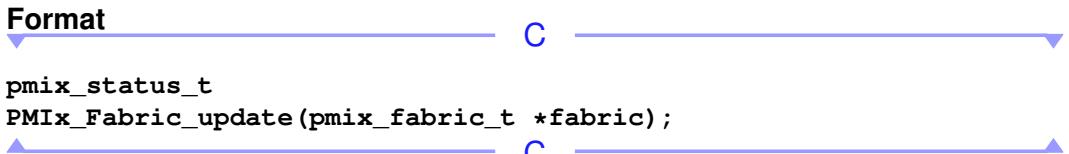
17 Non-blocking form of **PMIx_Fabric_register**. The caller is not allowed to access the
18 provided **pmix_fabric_t** until the callback function has been executed, at which time the fabric
19 information will have been loaded into the provided structure.

20 **14.4.3 PMIx_Fabric_update**

21 **Summary**

22 Update fabric-related information.

23 **Format**

24 
25 **pmix_status_t**
26 **PMIx_Fabric_update(pmix_fabric_t *fabric);**

26 **INPUT** **fabric**

27 address of a **pmix_fabric_t** (backed by storage) (handle)

28 >Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

29 **Description**

30 Update fabric-related information. This call can be made at any time to request an update of the
31 fabric information contained in the provided **pmix_fabric_t** object. The caller is not allowed to
32 access the provided **pmix_fabric_t** until the call has returned. Upon successful return, the
33 information fields in the *fabric* structure will have been updated.

14.4.4 PMIx_Fabric_update_nb

Summary

Update fabric-related information.

Format

```
pmix_status_t  
PMIx_Fabric_update_nb(pmix_fabric_t *fabric,  
                      pmix_op_cbfunc_t cbfunc, void *cbdata);
```

INOUT fabric

address of a `pmix_fabric_t` (handle)

IN cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

IN cbdata

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

Returns one of the following:

- `PMIX_SUCCESS` indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

Description

Non-blocking form of `PMIx_Fabric_update`. The caller is not allowed to access the provided `pmix_fabric_t` until the callback function has been executed, at which time the fields in the provided *fabric* structure will have been updated.

14.4.5 PMIx_Fabric_deregister

Summary

Deregister a fabric object.

Format

```
pmix_status_t  
PMIx_Fabric_deregister(pmix_fabric_t *fabric);
```

IN fabric

address of a `pmix_fabric_t` (handle)

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

Description

Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information (e.g., cost matrix) associated with it. Contents of the provided `pmix_fabric_t` will be invalidated upon function return.

14.4.6 PMIx_Fabric_deregister_nb

Summary

Deregister a fabric object.

Format

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)

Returns one of the following:

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

Description

Non-blocking form of [PMIx_Fabric_deregister](#). Provided *fabric* must not be accessed until after callback function has been executed.

CHAPTER 15

Security

1 PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes.
2 By definition, *client* processes must be preregistered with the PMIx server library via the
3 **PMIx_server_register_client** API before they are spawned. This API requires that the
4 host pass the expected effective UID/GID of the client process.

5 When the client attempts to connect to the PMIx server, the server shall use available standard 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 15.1 Obtaining Credentials

31 Applications and tools often interact with the host environment in ways that require security beyond
32 just verifying the user's identity - e.g., access to that user's relevant authorizations. This is

1 particularly important when tools connect directly to a system-level PMIx server that may be
2 operating at a privileged level. A variety of system management software packages provide
3 authorization services, but the lack of standardized interfaces makes portability problematic.

4 This section defines two PMIx client-side APIs for this purpose. These are most likely to be used
5 by user-space applications/tools, but are not restricted to that realm.

6 15.1.1 PMIx_Get_credential

7 Summary

8 Request a credential from the PMIx server library or the host environment.

9 Format

```
10 pmix_status_t  
11 PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,  
12                         pmix_byte_object_t *credential);
```

13 IN **info**
14 Array of **pmix_info_t** structures (array of handles)
15 IN **ninfo**
16 Number of elements in the *info* array (**size_t**)
17 IN **credential**
18 Address of a **pmix_byte_object_t** within which to return credential (handle)

19 Returns one of the following:

- 20 • **PMIX_SUCCESS**, indicating that the credential has been returned in the provided
21 **pmix_byte_object_t**
- 22 • a PMIx error constant indicating either an error in the input or that the request is unsupported

Required Attributes

23 There are no required attributes for this API. Note that implementations may choose to internally
24 execute integration for some security environments (e.g., directly contacting a *munge* server).

25 Implementations that support the operation but cannot directly process the client's request must
26 pass any attributes that are provided by the client to the host environment for processing. In
27 addition, the following attributes are required to be included in the *info* array passed from the PMIx
28 library to the host environment:

```
29 PMIX_USERID "pmix.euid" (uint32_t)  
30     Effective user ID of the connecting process.  
31 PMIX_GRPID "pmix.egid" (uint32_t)  
32     Effective group ID of the connecting process.
```

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (**int**)

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

Description

Request a credential from the PMIx server library or the host environment. The credential is returned as a **pmix_byte_object_t** to support potential binary formats - it is therefore opaque to the caller. No information as to the source of the credential is provided.

15.1.2 PMIx_Get_credential_nb

Summary

Request a credential from the PMIx server library or the host environment.

Format

```
14     pmix_status_t  
15     PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,  
16                               pmix_credential_cbfunc_t cbfunc,  
17                               void *cbdata);
```

IN **info**
Array of **pmix_info_t** structures (array of handles)
IN **ninfo**
Number of elements in the *info* array (**size_t**)
IN **cbfunc**
Callback function to return credential (**pmix_credential_cbfunc_t** function reference)
IN **cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided *cbfunc*
- a PMIx error constant indicating either an error in the input or that the request is unsupported - the *cbfunc* will *not* be called

Required Attributes

1 There are no required attributes for this API. Note that implementations may choose to internally
2 execute integration for some security environments (e.g., directly contacting a *munge* server).

3 Implementations that support the operation but cannot directly process the client's request must
4 pass any attributes that are provided by the client to the host environment for processing. In
5 addition, the following attributes are required to be included in the *info* array passed from the PMIx
6 library to the host environment:

7 **PMIX_USERID** "pmix.euid" (**uint32_t**)
8 Effective user ID of the connecting process.
9 **PMIX_GRP_ID** "pmix.egid" (**uint32_t**)
10 Effective group ID of the connecting process.

Optional Attributes

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

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

16 **Description**

17 Request a credential from the PMIx server library or the host environment. This version of the API
18 is generally preferred in scenarios where the host environment may have to contact a remote
19 credential service. Thus, provision is made for the system to return additional information (e.g., the
20 identity of the issuing agent) outside of the credential itself and visible to the application.

21 **15.1.3 Credential Attributes**

22 The following attributes are defined to support credential operations:

23 **PMIX_CRED_TYPE** "pmix.sec.ctype" (**char***)
24 When passed in **PMIx_Get_credential**, a prioritized, comma-delimited list of desired
25 credential types for use in environments where multiple authentication mechanisms may be
26 available. When returned in a callback function, a string identifier of the credential type.
27 **PMIX_CRYPTO_KEY** "pmix.sec.key" (**pmix_byte_object_t**)
28 Blob containing crypto key.

15.2 Validating Credentials

Given a credential, PMIx provides two methods by which a caller can request that the system validate it, returning any additional information (e.g., authorizations) conveyed within the credential.

15.2.1 PMIx_Validate_credential

Summary

Request validation of a credential by the PMIx server library or the host environment.

Format

```
pmix_status_t  
PMIx_Validate_credential(const pmix_byte_object_t *cred,  
                         const pmix_info_t info[], size_t ninfo,  
                         pmix_info_t **results, size_t *nresults);
```

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`)

INOUT results

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

INOUT nresults

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

Returns 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 *results* array
- a PMIx error constant indicating either an error in the parsing of the credential or that the request was refused

Required Attributes

1 There are no required attributes for this API. Note that implementations may choose to internally
2 execute integration for some security environments (e.g., directly contacting a *munge* server).

3 Implementations that support the operation but cannot directly process the client's request must
4 pass any attributes that are provided by the client to the host environment for processing. In
5 addition, the following attributes are required to be included in the *info* array passed from the PMIx
6 library to the host environment:

7 **PMIX_USERID** "pmix.euid" (**uint32_t**)

8 Effective user ID of the connecting process.

9 **PMIX_GRP_ID** "pmix.egid" (**uint32_t**)

10 Effective group ID of the connecting process.

Optional Attributes

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

12 **PMIX_TIMEOUT** "pmix.timeout" (**int**)

13 Time in seconds before the specified operation should time out (zero indicating infinite) and
14 return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions
15 caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

16 Request validation of a credential by the PMIx server library or the host environment.
17

18 15.2.2 PMIx_Validate_credential_nb

19 Summary

20 Request validation of a credential by the PMIx server library or the host environment. Provision is
21 made for the system to return additional information regarding possible authorization limitations
22 beyond simple authentication.

1 Format

C

```
2 pmix_status_t  
3 PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,  
4                               const pmix_info_t info[], size_t ninfo,  
5                               pmix_validation_cbfunc_t cbfunc,  
6                               void *cbdata);
```

C

7 IN cred

8 Pointer to `pmix_byte_object_t` containing the credential (handle)

9 IN info

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

11 IN ninfo

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

13 IN cbfunc

14 Callback function to return result (`pmix_validation_cbfunc_t` function reference)

15 IN cbdata

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

17 Returns one of the following:

- 18 • `PMIX_SUCCESS`, indicating that the request has been communicated to the local PMIx server -
19 result will be returned in the provided *cbfunc*
- 20 • a PMIx error constant indicating either an error in the input or that the request is unsupported -
21 the *cbfunc* will *not* be called

22 Upon completion of processing the callback function will be executed. Note that the callback
23 function must not be executed prior to return from the API.

▼----- Required Attributes -----▼

24 There are no required attributes for this API. Note that implementations may choose to internally
25 execute integration for some security environments (e.g., directly contacting a *munge* server).

26 Implementations that support the operation but cannot directly process the client's request must
27 pass any attributes that are provided by the client to the host environment for processing. In
28 addition, the following attributes are required to be included in the *info* array passed from the PMIx
29 library to the host environment:

30 **PMIX_USERID** "pmix.euid" (`uint32_t`)

31 Effective user ID of the connecting process.

32 **PMIX_GRP_ID** "pmix.egid" (`uint32_t`)

33 Effective group ID of the connecting process.

Optional Attributes

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

2 **PMIX_TIMEOUT** "pmix.timeout" (int)

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

Description

6 Request validation of a credential by the PMIx server library or the host environment. This version
7 of the API is generally preferred in scenarios where the host environment may have to contact a
8 remote credential service. Provision is made for the system to return additional information (e.g.,
9 possible authorization limitations) beyond simple authentication.
10

CHAPTER 16

Server-Specific Interfaces

The process that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes:

- collecting inventory to support scheduling algorithms,
- providing subsystems with an opportunity to precondition their resources for optimized application support,
- generating regular expressions,
- registering information to be passed to client processes, and
- requesting information on behalf of a remote process.

Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

16.1 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers.

16.1.1 PMIx_server_init

Summary

Initialize the PMIx server.

Format

```
pmix_status_t  
PMIx_server_init(pmix_server_module_t *module,  
                  pmix_info_t info[], size_t ninfo);
```

C

```

1 INOUT module
2     pmix_server_module_t structure (handle)
3 IN info
4     Array of pmix_info_t structures (array of handles)
5 IN ninfo
6     Number of elements in the info array (size_t)
7 Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

Required Attributes

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

```

9 PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
10    Name of the namespace to use for this PMIx server.

11 PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
12    Rank of this PMIx server.

13 PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)
14    Top-level temporary directory for all client processes connected to this server, and where the
15    PMIx server will place its tool rendezvous point and contact information.

16 PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
17    Temporary directory for this system, and where a PMIx server that declares itself to be a
18    system-level server will place a tool rendezvous point and contact information.

19 PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)
20    The host RM wants to declare itself as willing to accept tool connection requests.

21 PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)
22    The host RM wants to declare itself as being the local system server for PMIx connection
23    requests.

24 PMIX_SERVER_SESSION_SUPPORT "pmix.srvr.sess" (bool)
25    The host RM wants to declare itself as being the local session server for PMIx connection
26    requests.

27 PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)
28    Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
29    (e.g., logging to email).

30 PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
31    Server is supporting system scheduler and desires access to appropriate WLM-supporting
32    features. Indicates that the library is to be initialized for scheduler support.

```

Optional Attributes

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

2 **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)

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

5 **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)

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

8 **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)

9 Use only one rendezvous socket, letting priorities and/or environment parameters select the
10 active transport.

11 **PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)

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

15 **PMIX_TCP_IF_INCLUDE** "pmix.tcp.ifinclude" (char*)

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

19 **PMIX_TCP_IF_EXCLUDE** "pmix.tcp.ifexclude" (char*)

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

23 **PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)

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

26 **PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)

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

29 **PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)

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

32 **PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)

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

35 **PMIX_SERVER_REMOTE_CONNECTIONS** "pmix.srvr.remote" (bool)

```

1      Allow connections from remote tools. Forces the PMIx server to not exclusively use
2      loopback device. If the library supports connections from remote tools, this attribute may
3      be supported for enabling or disabling it.
4      PMIX_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.

Advice to PMIx server hosts

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.

16.1.2 PMIx_server_finalize

Summary

Finalize the PMIx server library.

Format

```
pmix_status_t  
PMIx_server_finalize(void);
```

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

Description

Finalize the PMIx server support library, terminating all connections to attached tools and any local clients. All memory usage is released.

16.1.3 Server Initialization Attributes

These attributes are used to direct the configuration and operation of the PMIx server library by passing them into **PMIx_server_init**.

PMIX_TOPOLOGY2 "pmix.topo2" (pmix_topology_t)

Provide a pointer to an implementation-specific description of the local node topology.

PMIX_SERVER_SHARE_TOPOLOGY "pmix.srvr.share" (bool)

The PMIx server is to share its copy of the local node topology (whether given to it or self-discovered) with any clients.

PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)

Disable legacy UNIX socket (usock) support.

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)

POSIX *mode_t* (9 bits valid).

PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)

Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)

The host RM wants to declare itself as willing to accept tool connection requests.

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_NSPACE "pmix.srv.nspace" (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 16.2 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 16.2.1 **PMIx_generate_regex**

39 Summary

40 Generate a compressed representation of the input string.

Format

C

```
2 pmix_status_t
3 PMIx_generate_regex(const char *input, char **output);
```

C

4 **IN** **input**

5 String to process (string)

6 **OUT** **output**

7 Compressed representation of *input* (array of bytes)

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

9 Description

10 Given a comma-separated list of *input* values, generate a reduced size representation of the input
 11 that can be passed down to the PMIx server library's **PMIx_server_register_nspace** API
 12 for parsing. The order of the individual values in the *input* string is preserved across the operation.
 13 The caller is responsible for releasing the returned data.

14 The precise compressed representations will be implementation specific. The regular expression
 15 itself is not required to be a printable string nor to obey typical string constraints (e.g., include a
 16 **NULL** terminator byte). However, all PMIx implementations are required to include a
 17 colon-delimited **NULL**-terminated string at the beginning of the output representation that can be
 18 printed for diagnostic purposes and identifies the method used to generate the representation. The
 19 following identifiers are reserved by the PMIx Standard:

- 20 • "**raw**: \0" - indicates that the expression following the identifier is simply the
 21 comma-delimited input string (no processing was performed).
- 22 • "**pmix**: \0" - a PMIx-unique regular expression represented as a **NULL**-terminated string
 23 following the identifier.
- 24 • "**blob**: \0" - a PMIx-unique regular expression that is not represented as a **NULL**-terminated
 25 string following the identifier. Additional implementation-specific metadata may follow the
 26 identifier along with the data itself. For example, a compressed binary array format based on the
 27 *zlib* compression package, with the size encoded in the space immediately following the
 28 identifier.

29 Communicating the resulting output should be done by first packing the returned expression using
 30 the **PMIx_Data_pack**, declaring the input to be of type **PMIX_REGEX**, and then obtaining the
 31 resulting blob to be communicated using the **PMIX_DATA_BUFFER_UNLOAD** macro. The
 32 reciprocal method can be used on the remote end prior to passing the regex into
 33 **PMIx_server_register_nspace**. The pack/unpack routines will ensure proper handling of
 34 the data based on the regex prefix.

1 **16.2.2 PMIx_generate_ppn**

2 **Summary**

3 Generate a compressed representation of the input identifying the processes on each node.

4 **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 corresponding to a PMIx error constant.

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 **16.2.3 PMIx_server_register_nspace**

22 **Summary**

23 Setup the data about a particular namespace.

24 **Format**

```
25 pmix_status_t  
26 PMIx_server_register_nspace(const pmix_nspace_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 Returns one of the following:
17
18  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
19    will be returned in the provided cbfunc. Note that the library must not invoke the callback
    function prior to returning from the API.
20
21  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
    returned success - the cbfunc will not be called
22
23  • a PMIx error constant indicating either an error in the input or that the request was immediately
    processed and failed - the cbfunc will not be called

```

----- Required Attributes -----

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

```

25  PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)
26    Registration is for this namespace only, do not copy job data.

27  PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)
28    Provide an array of pmix_info_t containing session-realm information. The
29    PMIX_SESSION_ID attribute is required to be included in the array.

30  PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
31    Provide an array of pmix_info_t containing job-realm information. The
32    PMIX_SESSION_ID attribute of the session containing the job is required to be included in
33    the array whenever the PMIx server library may host multiple sessions (e.g., when executing
34    with a host RM daemon). As information is registered one job (aka namespace) at a time via
35    the PMIx_server_register_nspace API, there is no requirement that the array
36    contain either the PMIX_NAMESPACE or PMIX_JOBID attributes when used in that context

```

(though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is ambiguous.

PMIX_APP_INFO_ARRAY "pmix.app.arr" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing application-realm information. The `PMIX_NSPACE` or `PMIX_JOBID` attributes of the *job* containing the application, plus its `PMIX_APPNUM` attribute, must to be included in the array when the array is *not* included as part of a call to `PMIX_server_register_nspace` - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

PMIX_PROC_INFO_ARRAY "pmix_pdata" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing process-realm information. The `PMIX_RANK` and `PMIX_NSPACE` attributes, or the `PMIX_PROCID` attribute, are required to be included in the array when the array is not included as part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the process is ambiguous. All three may be included if desired. When the array is included in some broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID` attribute must be included (the others are optional).

PMIX_NODE_INFO_ARRAY "pmix.node.arr" (**pmix_data_array_t**)

Provide an array of `pmix_info_t` containing node-realm information. At a minimum, either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the array, though both may be included.

Host environments are required to provide a wide range of session-, job-, application-, node-, and process-realm information, and may choose to provide a similarly wide range of optional information. The information is broadly separated into categories based on the *data realm* definitions explained in Section 6.1, and retrieved according to the rules detailed in Section 6.2.

Session-realm information may be passed as individual `pmix_info_t` entries, or as part of a `pmix_data_array_t` using the `PMIX_SESSION_INFO_ARRAY` attribute. The list of data referenced in this way shall include:

- **PMIX UNIV SIZE** "pmix.univ.size" (uint32 t)

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 PMIx Standard for historical reasons.

- **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)

Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. 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_NSPACE** "pmix.srv.nspace" (**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_NSPACE** "pmix.nspace" (**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 [16.2.3.2](#) 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 [16.2.3.2](#)
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 the
28 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
4 • PMIX_CRYPTO_KEY "pmix.sec.key" (pmix_byte_object_t)
5 Blob containing crypto key.
6
7 If more than one application is included in the namespace, then the host environment is also
8 required to supply data consisting of the following items for each application in the job, passed as a
9 pmix_data_array_t using the PMIX_APP_INFO_ARRAY attribute:
10
11 • PMIX_APPNUM "pmix.appnum" (uint32_t)
12 The application number within the job in which the specified process is a member. This
13 attribute must appear at the beginning of the array.
14
15 • PMIX_APP_SIZE "pmix.app.size" (uint32_t)
16 Number of processes in the specified application, regardless of their execution state - i.e.,
17 this number may include processes that either failed to start or have already terminated.
18
19 • PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
20 Maximum number of processes that can be executed in the specified realm. Typically, this
21 is a constraint imposed by a scheduler or by user settings in a hostfile or other resource
22 description. Defaults to the job realm. Requires use of the PMIX_APP_INFO attribute
23 to avoid ambiguity when retrieving it.
24
25 • PMIX_APPLDR "pmix.apldr" (pmix_rank_t)
26 Lowest rank in the specified application.
27
28 • PMIX_WDIR "pmix.wdir" (char*)
29 Working directory for spawned processes. This attribute is required for all registrations,
30 but may be provided as an individual pmix_info_t entry if only one application is
31 included in the namespace.
32
33 • PMIX_APP_ARGV "pmix.app.argv" (char*)
34 Consolidated argv passed to the spawn command for the given application (e.g., "./myapp
35 arg1 arg2 arg3"). This attribute is required for all registrations, but may be provided as an
36 individual pmix_info_t entry if only one application is included in the namespace.
37 plus the following optional information:
38
39 • PMIX_PSET_NAMES "pmix.pset.nms" (pmix_data_array_t*)
40 Returns an array of char* string names of the process sets in which the given process is
41 a member.
42
43 • PMIX_APP_MAP_TYPE "pmix.apmap.type" (char*)
44 Type of mapping used to layout the application (e.g., cyclic). This attribute may be
45 provided as an individual pmix_info_t entry if only one application is included in the
46 namespace.
47
48 • 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`)

Lowest rank within the specified job on the node (defaults to current node in absence of `PMIX_HOSTNAME` or `PMIX_NODEID` qualifier).

- **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.

plus the following information for the server's own node:

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

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

- **PMIX\_NSDIR** "pmix.nsdir" (`char*`)

Full path to the temporary directory assigned to the specified job, under **PMIX\_TMPDIR**.

- **PMIX LOCAL PROCS** "pmix.lprocs" (pmix proc t array)

Array of `pmix_proc_t` of all processes executing on the local node – shortcut for `PMIx_Resolve_peers` for the local node and a `NULL` namespace argument. The process identifier is ignored for this attribute.

The data may also include the following optional information for the server's own node:

- **PMIX LOCAL CPUSETS** "pmix.lcpus" (pmix data array t)

A `pmix_data_array_t` array of string representations of the PU binding bitmaps applied to each local `peer` on the caller's node upon launch. Each string shall begin with the name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap string itself. The array shall be in the same order as the processes returned by `PMIX_LOCAL_PEERS` for that namespace.

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

Total available physical memory on a node. As this information is not related to the namespace, it can be passed using the [PMIx server register resources API](#).

and the following optional information for other nodes:

- **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. Requires use of the **PMIX\_NODE\_INFO** attribute to avoid ambiguity when retrieving it.

Process-realm information shall include the following data for each process in the job, passed as a `pmix_data_array_t` using the `PMIX PROC INFO ARRAY` attribute:

- **PMIX\_RANK** "pmix.rank" (pmix rank t)

Process rank within the job, starting from zero.

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

The application number within the job in which the specified process is a member. This attribute may be omitted if only one application is present in the namespace.

- **PMIX\_APP\_RANK** "pmix.apprank" (**pmix\_rank\_t**)  
Rank of the specified process within its application. This attribute may be omitted if only one application is present in the namespace.
  - **PMIX\_GLOBAL\_RANK** "pmix.grank" (**pmix\_rank\_t**)  
Rank of the specified process spanning across all jobs in this session, starting with zero. Note that no ordering of the jobs is implied when computing this value. As jobs can start and end at random times, this is defined as a continually growing number - i.e., it is not dynamically adjusted as individual jobs and processes are started or terminated.
  - **PMIX\_LOCAL\_RANK** "pmix.lrank" (**uint16\_t**)  
Rank of the specified process on its node - refers to the numerical location (starting from zero) of the process on its node when counting only those processes from the same job that share the node, ordered by their overall rank within that job.
  - **PMIX\_NODE\_RANK** "pmix.nrank" (**uint16\_t**)  
Rank of the specified process on its node spanning all jobs- refers to the numerical location (starting from zero) of the process on its node when counting all processes (regardless of job) that share the node, ordered by their overall rank within the job. The value represents a snapshot in time when the specified process was started on its node and is not dynamically adjusted as processes from other jobs are started or terminated on the node.
  - **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\_REINCarnation** "pmix.reinc" (**uint32\_t**)  
Number of times this process has been re-instantiated - i.e, a value of zero indicates that the process has never been restarted. 5
  - **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.
- plus the following information for processes that are local to the server:
- **PMIX\_LOCALITY\_STRING** "pmix.locstr" (**char\***)  
String describing a process's bound location - referenced using the process's rank. The string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon. The remainder of the string represents the corresponding locality as expressed by the underlying implementation. The entire string must be passed to **PMIx\_Get\_relative\_locality** for processing. Note that hosts are only required to provide locality strings for local client processes - thus, a call to **PMIx\_Get** for the locality string of a process that returns **PMIX\_ERR\_NOT\_FOUND** indicates that the 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.

---

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.

### 16.2.3.1 Namespace registration attributes

2 The following attributes are defined specifically for use with the  
3 **PMIx\_server\_register\_nspace** 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 16.2.3.2 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\_nspace** API, there is no requirement that the array  
18 contain either the **PMIX\_NSPACE** 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\_NSPACE** or **PMIX\_JOBID** attributes of the *job* containing the application, plus its  
24 **PMIX\_APPNUM** attribute, must be included in the array when the array is *not* included as  
25 part of a call to **PMIx\_server\_register\_nspace** - 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\_NSPACE** 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\_nspace** - 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.

- 1     • a **PMIX\_JOB\_INFO\_ARRAY** could contain a **PMIX\_NODE\_INFO\_ARRAY** for each node  
2       hosting processes from that job, each array describing job-level values for that node.  
3     • a **PMIX\_SESSION\_INFO\_ARRAY** might contain multiple **PMIX\_JOB\_INFO\_ARRAY**  
4       elements, each describing a job executing within the session. Each job array could, in turn,  
5       contain both application and node arrays, thus providing a complete picture of the active  
6       operations within the allocation.

---

### Advice to PMIx library implementers

---

7     PMIx implementations must be capable of properly parsing and storing any hierarchical depth of  
8       information arrays. The resulting stored values are must to be accessible via both **PMIx\_Get** and  
9       **PMIx\_Query\_info\_nb** APIs, assuming appropriate directives are provided by the caller.

---

#### 16.2.3.2 Assembling the registration information

11    The following description is not intended to represent the actual layout of information in a given  
12      PMIx library. Instead, it is describes how information provided in the *info* parameter of the  
13      **PMIx\_server\_register\_nspace** shall be organized for proper processing by a PMIx server  
14      library. The ordering of the various information elements is arbitrary - they are presented in a  
15      top-down hierarchical form solely for clarity in reading.

---

### Advice to PMIx server hosts

---

16    Creating the *info* array of data requires knowing in advance the number of elements required for the  
17      array. This can be difficult to compute and somewhat fragile in practice. One method for resolving  
18      the problem is to create a linked list of objects, each containing a single **pmix\_info\_t** structure.  
19      Allocation and manipulation of the list can then be accomplished using existing standard methods.  
20      Upon completion, the final *info* array can be allocated based on the number of elements on the list,  
21      and then the values in the list object **pmix\_info\_t** structures transferred to the corresponding  
22      array element utilizing the **PMIX\_INFO\_XFER** macro.

---

23    A common building block used in several areas is the construction of a regular expression  
24      identifying the nodes involved in that area - e.g., the nodes in a *session* or *job*. PMIx provides  
25      several tools to facilitate this operation, beginning by constructing an argv-like array of node  
26      names. This array is then passed to the **PMIx\_generate\_regex** function to create a regular  
27      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, ®ex);
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_NSPACE(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, ®ex);
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);

```

C

Note that the **PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes are linked in that the order of entries in the process map must match the ordering of nodes in the node map - i.e., there is no provision in the PMIx process map regular expression generator/parser pair supporting an out-of-order node or a node that has no corresponding process map entry (e.g., a node with no processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

- Session-level information includes all session-specific values. In many cases, only two values (**PMIX\_SESSION\_ID** and **PMIX\_UNIV\_SIZE**) are included in the registration array. Since both of these values are session-specific, they can be specified independently - i.e., in their own **pmix\_info\_t** elements of the *info* array. Alternatively, they can be provided as a **pmix\_data\_array\_t** array of **pmix\_info\_t** using the **PMIX\_SESSION\_INFO\_ARRAY** attribute and identified by including the **PMIX\_SESSION\_ID** attribute in the array - this is required in cases where non-specific attributes (e.g., **PMIX\_NUM\_NODES** or **PMIX\_NODE\_MAP**) are passed to describe aspects of the session. Note that the node map can include nodes not used by the job being registered as no corresponding process map is specified.

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):

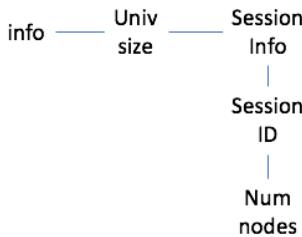


Figure 16.1.: Session-level information elements

- 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

that since the invocation only involves a single namespace, there is no need to include the **PMIX\_NSPACE** attribute in the array.

Upon conclusion of this step, the *info* array might look like:

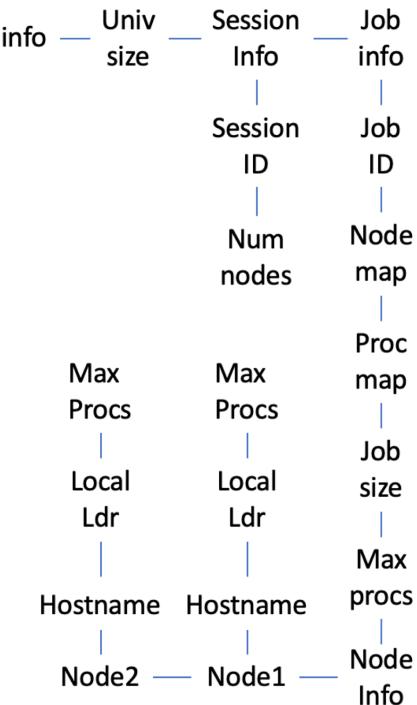


Figure 16.2.: Job-level information elements

Note that in this example, **PMIX\_NUM\_NODES** is not required as that information is contained in the **PMIX\_NODE\_MAP** attribute. Similarly, **PMIX\_JOB\_SIZE** is not technically required as that information is contained in the **PMIX\_PROC\_MAP** when combined with the corresponding node map - however, there is no issue with including the job size as a separate entry.

The example also illustrates the hierarchical use of the `PMIX_NODE_INFO_ARRAY` attribute. In this case, we have chosen to pass several job-related values for each node - since those values are non-unique across the job, they must be passed in a node-info container. Note that the choice of what information to pass into the PMIx server library versus what information to derive from other values at time of request is left to the host environment. PMIx implementors in turn may, if they choose, pre-parse registration data to create expanded views (thus enabling faster response to requests at the expense of memory footprint) or to compress views into tighter representations (thus trading minimized footprint for longer response times).

- Application-level information includes all application-specific values such as **PMIX\_APP\_SIZE**

1 and **PMIX\_APPLDR**. If the *job* contains only a *application*, then the application-specific  
 2 values can be specified independently - i.e., in their own **pmix\_info\_t** elements of the *info*  
 3 array - or as a **pmix\_data\_array\_t** array of **pmix\_info\_t** using the  
 4 **PMIX\_APP\_INFO\_ARRAY** attribute and identified by including the **PMIX\_APPNUM** attribute in  
 5 the array. Use of the array format is must in cases where non-specific attributes (e.g.,  
 6 **PMIX\_NODE\_MAP**) are passed to describe aspects of the application.

7 However, in the case of a job consisting of multiple applications, all application-specific values  
 8 for each application must be provided using the **PMIX\_APP\_INFO\_ARRAY** format, each  
 9 identified by its **PMIX\_APPNUM** value.

10 Upon conclusion of this step, the *info* array might look like that shown in 16.3, assuming there  
 11 are two applications in the job being registered:

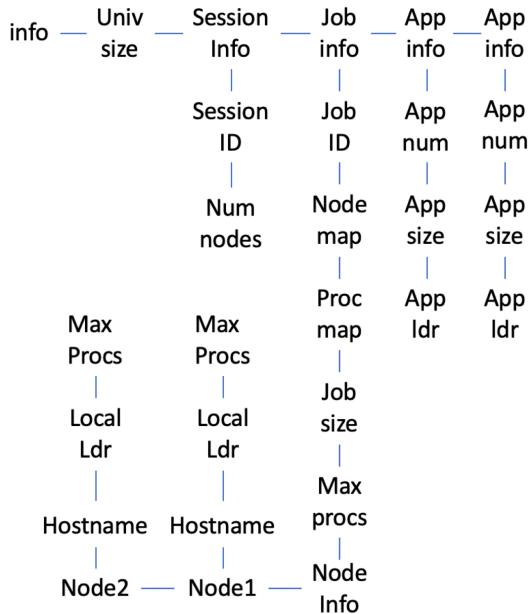


Figure 16.3.: Application-level information elements

- 12 Process-level information includes an entry for each process in the job being registered, each  
 13 entry marked with the **PMIX\_PROC\_INFO\_ARRAY** attribute. The *rank* of the process must be  
 14 the first entry in the array - this provides efficiency when storing the data. Upon conclusion of  
 15 this step, the *info* array might look like the diagram in 16.4:
- 16 • For purposes of this example, node-level information only includes values describing the local  
 17 node - i.e., it does not include information about other nodes in the job or session. In many cases,  
 18 the values included in this level are unique to it and can be specified independently - i.e., in their  
 19 own **pmix\_info\_t** elements of the *info* array. Alternatively, they can be provided as a

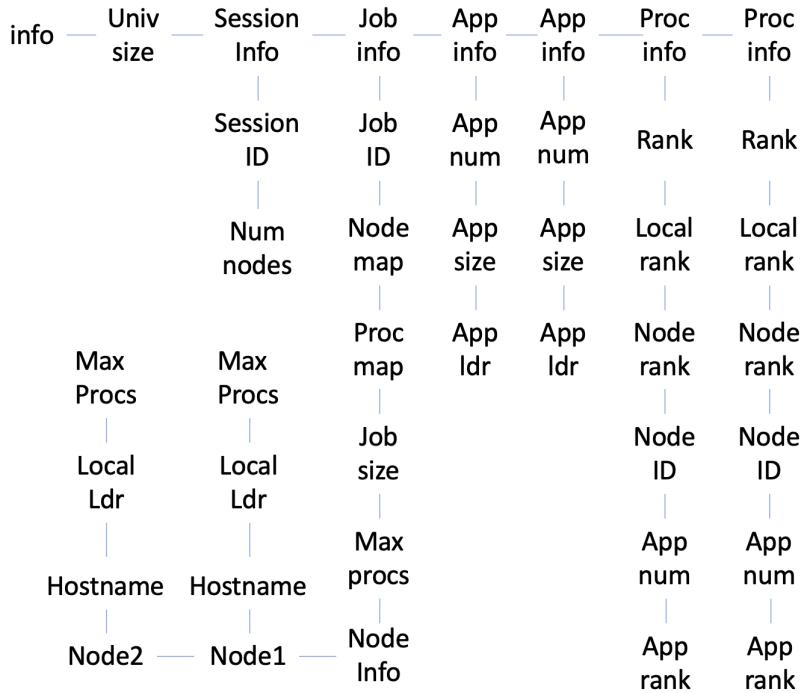


Figure 16.4.: Process-level information elements

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:

```

1 char **ndppn = NULL;
2 char rank[30];
3 char *localranks;
4 size_t m;
5 pmix_info_t info;
6
7 for (m=0; m < mynode->num_procs; m++) {
8 /* ignore processes that are not part of the target job */
9 if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace)) {
10 continue;
11 }
12 sprintf(rank, 30, "%d", mynode->proc[m].rank);
13 PMIX_ARGV_APPEND(&ndppn, rank);
14 }
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);

```

The **PMIX\_LOCAL\_CPUSETS** value is constructed in a similar manner. In the provided example, it is assumed that an Hardware Locality (HWLOC) cpuset representation (a comma-delimited string of processor IDs) of the processors assigned to each process has previously been generated and stored on the process description. Thus, the value can be constructed as shown below:

```

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_NSPACE(targetjob,mynode->proc[m].nspace)) {
38 continue;

```

```

1
2 }
3 PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
4 }
5 /* convert the array into a colon-delimited string */
6 localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
7 /* release the local array */
8 PMIX_ARGV_FREE(ndcpus);

9 /* pass the string as the value to the PMIX_LOCAL_CPUESTS key */
10 PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUESTS, 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 [16.5](#):

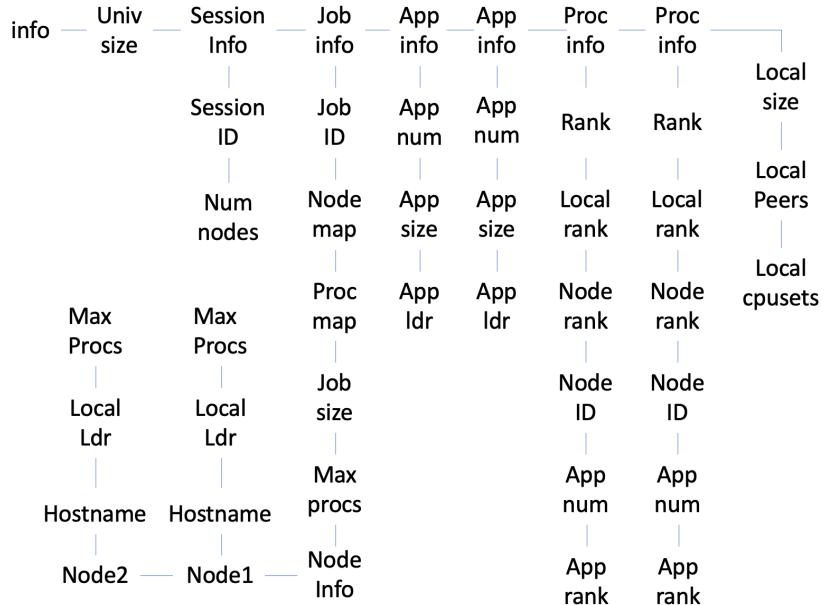


Figure 16.5.: Final information array

## 16.2.4 PMIx\_server\_deregister\_nspace

### Summary

Deregister a namespace.

### Format

```
void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

#### IN nspace

Namespace (string)

#### 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

Deregister the specified *nspace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library must not invoke the callback function prior to returning from the API, and that a **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

## 16.2.5 PMIx\_server\_register\_resources

### Summary

Register non-namespace related information with the local PMIx server library.

### Format

```
pmix_status_t
PMIx_server_register_resources(pmix_info_t info[], size_t ninfo,
 pmix_op_cbfunc_t cbfunc,
 void *cbdata);
```

#### 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_cfunc_t`. A `NULL` function reference indicates that the function is to be executed as a blocking operation (function reference)

IN chdata

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

## Description

Pass information about resources not associated with a given namespace to the PMIx server library for distribution to local client processes. This includes information on fabric devices, GPUs, and other resources. All information provided through this API shall be made available to each job as part of its job-level information. Duplicate information provided with the [PMIx\\_server\\_register\\_nspace](#) API shall override any information provided by this function for that namespace, but only for that specific namespace.

## Advice to PMIx server hosts

Note that information passed in this manner could also have been included in a call to `PMIx_server_register_nspace` - e.g., as part of a `PMIX_NODE_INFO_ARRAY` array. This API is provided as a logical alternative for code clarity, especially where multiple jobs may be supported by a single PMIx server library instance, to avoid multiple registration of static resource information.

A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

### 16.2.6 PMIx server deregister resources

## Summary

Remove specified non-namespace related information from the local PMIx server library.

## Format

```
pmix_status_t
PMIx_server_deregister_resources(pmix_info_t info[], size_t ninfo,
 pmix_op_cfunc_t cfunc,
 void *cbdata);
```

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_cfunc_t`. A `NULL` function reference indicates that the function is to be executed as a blocking operation (function reference)

IN cbdata

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

1           **Description**

2         Remove information about resources not associated with a given namespace from the PMIx server  
3         library. Only the *key* fields of the provided *info* array shall be used for the operation - the associated  
4         values shall be ignored except where they serve as qualifiers to the request. For example, to remove  
5         a specific fabric device from a given node, the *info* array might include a  
6         **PMIX\_NODE\_INFO\_ARRAY** containing the **PMIX\_NODEID** or **PMIX\_HOSTNAME** identifying  
7         the node hosting the device, and the **PMIX\_FABRIC\_DEVICE\_NAME** specifying the device to be  
8         removed. Alternatively, the device could be removed using only the **PMIX\_DEVICE\_ID** as this is  
9         unique across the overall system.

---

Advice to PMIx server hosts

---

10      As information not related to namespaces is considered *static*, there is no requirement that the host  
11     environment deregister resources prior to finalizing the PMIx server library. The server library  
12     shall properly cleanup as part of its normal finalize operations. Deregistration of resources is only  
13     required, therefore, when the host environment determines that client processes should no longer  
14     have access to that information.

15      A **NULL** *cbfunc* reference indicates that the function is to be executed as a blocking operation.

16     **16.2.7 PMIx\_server\_register\_client**

17       **Summary**

18       Register a client process with the PMIx server library.

19       **Format**

20       

```
pmix_status_t
21 PMIx_server_register_client(const pmix_proc_t *proc,
22 uid_t uid, gid_t gid,
23 void *server_object,
24 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

25       **IN proc**
26           **pmix\_proc\_t** structure (handle)
27       **IN uid**
28           user id (integer)
29       **IN gid**
30           group id (integer)
31       **IN server\_object**
32           (memory reference)

1   **IN** **`cbfunc`**  
2   Callback function `pmix_op_cbfunc_t`. 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   Returns one of the following:

- 7   • **`PMIX_SUCCESS`**, indicating that the request is being processed by the host environment - result  
8   will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
9   function prior to returning from the API.
- 10   • **`PMIX_OPERATION_SUCCEEDED`**, indicating that the request was immediately processed and  
11   returned *success* - the *cbfunc* will not be called
- 12   • a PMIx error constant indicating either an error in the input or that the request was immediately  
13   processed and failed - the *cbfunc* will not be called

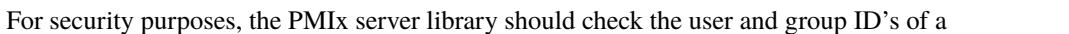
14   **Description**

15   Register a client process with the PMIx server library.

16   The host server can also, if it desires, provide an object it wishes to be returned when a server  
17   function is called that relates to a specific process. For example, the host server may have an object  
18   that tracks the specific client. Passing the object to the library allows the library to provide that  
19   object to the host server during subsequent calls related to that client, such as a  
20   `pmix_server_client_connected2_fn_t` function. This allows the host server to access  
21   the object without performing a lookup based on the client's namespace and rank.

22   **Advice to PMIx server hosts**

23   Host environments are required to execute this operation prior to starting the client process. The  
24   expected user ID and group ID of the child process allows the server library to properly authenticate  
25   clients as they connect by requiring the two values to match. Accordingly, the detected user and  
26   group ID's of the connecting process are not included in the  
`pmix_server_client_connected2_fn_t` server module function.  


27   **Advice to PMIx library implementers**

28   For security purposes, the PMIx server library should check the user and group ID's of a  
29   connecting process against those provided for the declared client process identifier via the  
`PMIx_server_register_client` prior to completing the connection.  


## 16.2.8 PMIx\_server\_deregister\_client

## Summary

Deregister a client and purge all data relating to it.

## Format

```
void
PMIx_server_deregister_client(const pmix_proc_t *proc,
 pmix_op cbfunc_t cbfunc, void *cbdata);
```

**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_nspace` API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library must not invoke the callback function prior to returning from the API.

## 16.2.9 PMIx\_server\_setup\_fork

## Summary

Setup the environment of a child process to be forked by the host.

## **Format**

```
pmix_status_t
PMIx_server_setup_fork(const pmix_proc_t *proc,
 char ***env);
```

**IN proc**  
pmix proc\_t structure (handle)

**IN env** Environment array (array of strings)

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

## Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via `PMIx_Spawn`) plus other values (e.g., variables required to properly initialize the client's fabric library).

## Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process.

## 16.2.10 PMIx server dmodex request

## Summary

Define a function by which the host server can request modeX data from the local PMIx server.

## Format

```
pmix_status_t
PMIx_server_dmodex_request(const pmix_proc_t *proc,
 pmix_dmodex_response_fn_t cbfunc,
 void *cbdata);
```

- IN proc**  
`pmix_proc_t` structure (handle)
- IN cfunc**  
Callback function `pmix_dmodex_response_fn_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.
  - a PMIx error constant indicating an error in the input - the *cbfunc* will not be called

1           **Description**

2       Define a function by which the host server can request modex data from the local PMIx server.  
3       Traditional wireup procedures revolve around the per-process posting of data (e.g., location and  
4       endpoint information) via the **PMIx\_Put** and **PMIx\_Commit** functions followed by a  
5       **PMIx\_Fence** barrier that globally exchanges the posted information. However, the barrier  
6       operation represents a significant time impact at large scale.

7       PMIx supports an alternative wireup method known as *Direct Modex* that replaces the  
8       barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In  
9       place of the barrier operation, data posted by each process is cached on the local PMIx server.  
10      When a process requests the information posted by a particular peer, it first checks the local cache  
11      to see if the data is already available. If not, then the request is passed to the local PMIx server,  
12      which subsequently requests that its RM host request the data from the RM daemon on the node  
13      where the specified peer process is located. Upon receiving the request, the RM daemon passes the  
14      request into its PMIx server library using the **PMIx\_server\_dmodex\_request** function,  
15      receiving the response in the provided *cbfunc* once the indicated process has posted its information.  
16      The RM daemon then returns the data to the requesting daemon, who subsequently passes the data  
17      to its PMIx server library for transfer to the requesting client.

---

Advice to users

---

18      While direct modex allows for faster launch times by eliminating the barrier operation, per-peer  
19      retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by  
20      returning posted information from all processes on a node upon first request - but in general direct  
21      modex remains best suited for sparsely connected applications.

22     **16.2.10.1 Server Direct Modex Response Callback Function**

23      The **PMIx\_server\_dmodex\_request** callback function.

24     **Summary**

25      Provide a function by which the local PMIx server library can return connection and other data  
26      posted by local application processes to the host resource manager.

## Format

```
typedef void (*pmix_dmodex_response_fn_t)(
 pmix_status_t status,
 char *data, size_t sz,
 void *cbdata);
```

**IN** **status**

Returned status of the request (**pmix status t**)

IN data

Pointer to a data "blob" containing the requested information (handle)

IN sz

Number of bytes in the *data* blob (integer)

IN cbdata

Data passed into the initial call to `PMIx_server_dmode_x_request` (memory reference)

## Description

Define a function to be called by the PMIx server library for return of information posted by a local application process (via [PMIx\\_Put](#) with subsequent [PMIx\\_Commit](#)) in response to a request from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd upon return from the function.

## 16.2.11 PMIx\_server\_setup\_application

## Summary

Provide a function by which a launcher can request application-specific setup data prior to launch of a *job*.

## **Format**

```
pmix_status_t
PMIx_server_setup_application(const pmix_nspace_t nspace,
 pmix_info_t info[], size_t ninfo,
 pmix_setup_application_cbfunc_t cbfunc,
 void *cbdata);
```

IN nspace

namespace (string)

IN info

Array of info structures (array of handles)

IN ninfo

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

```

1 IN cbfunc
2 Callback function pmix_setup_application_cbfunc_t (function reference)
3 IN cbdata
4 Data to be passed to the cbfunc callback function (memory reference)
5 Returns one of the following:
6
7 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
8 will be returned in the provided cbfunc. Note that the library must not invoke the callback
9 function prior to returning from the API.
10
11 • a PMIx error constant indicating either an error in the input - the cbfunc will not be called

```

### Required Attributes

PMIx libraries that support this operation are required to support the following:

```

11 PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)
12 Harvest and include relevant environmental variables.

13 PMIX_SETUP_APP_NONENVARS ""pmix.setup.nenv"" (bool)
14 Include all relevant data other than environmental variables.

15 PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)
16 Include all relevant data.

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_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
36 Request that the allocation include a fabric security key for the spawned job.

```

```
1 PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
2 Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
3 PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
4 ID string for the fabric plane to be used for the requested allocation.
5 PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
6 Number of endpoints to allocate per process in the job.
7 PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
8 Number of endpoints to allocate per node for the job.
9 PMIX_PROC_MAP "pmix.pmap" (char*)
10 Regular expression describing processes on each node in the specified realm - see 16.2.3.2
11 for an explanation of its generation. Defaults to the job realm.
12 PMIX_NODE_MAP "pmix.nmap" (char*)
13 Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2
14 for an explanation of its generation. Defaults to the job realm.
```



## Optional Attributes



PMIx libraries that support this operation may support the following:

```
16 PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
17 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
18 request.
19 PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
20 Fabric quality of service level for the job being requested in an allocation request.
21 PMIX_SESSION_INFO "pmix.ssn.info" (bool)
22 Return information regarding the session realm of the target process. In this context,
23 indicates that the information provided in the PMIX_NODE_MAP is for the entire session and
24 not just the indicated namespace. Thus, subsequent calls to this API may omit node-level
25 information - e.g., the library may not need to include information on the devices on each
26 node in a subsequent call.
```

The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the application. The PMIx server library may utilize this information to harvest/forward model-specific environmental variables, record the programming model associated with the application, etc.

- **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (char\*)  
Programming model being initialized (e.g., "MPI" or "OpenMP").
- **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (char\*)  
Programming model implementation ID (e.g., "OpenMPI" or "MPICH").

- 1     • **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (**char\***)  
2                 Programming model version string (e.g., "2.1.1").
- 

3     **Description**

4     Provide a function by which the RM can request application-specific setup data (e.g., environmental  
5     variables, fabric configuration and security credentials) from supporting PMIx server library  
6     subsystems prior to initiating launch of a job.

7     This is defined as a non-blocking operation in case contributing subsystems need to perform some  
8     potentially time consuming action (e.g., query a remote service) before responding. The returned  
9     data must be distributed by the host environment and subsequently delivered to the local PMIx  
10    server on each node where application processes will execute, prior to initiating execution of those  
11    processes.



Advice to PMIx server hosts

12    Host environments are required to execute this operation prior to launching a job. In addition to  
13    supported directives, the *info* array must include a description of the *job* using the  
14    **PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes.

15    Note that the function can be called on a per-application basis if the **PMIX\_PROC\_MAP** and  
16    **PMIX\_NODE\_MAP** are provided only for the corresponding application (as opposed to the entire  
17    job) each time.



Advice to PMIx library implementers

18    Support for harvesting of environmental variables and providing of local configuration information  
19    by the PMIx implementation is optional.



20    **16.2.11.1 Server Setup Application Callback Function**

21    The **PMIx\_server\_setup\_application** callback function.

22    **Summary**

23    Provide a function by which the resource manager can receive application-specific environmental  
24    variables and other setup data prior to launch of an application.

## Format

```
typedef void (*pmix_setup_application_cbfunc_t)(
 pmix_status_t status,
 pmix_info_t info[], size_t ninfo,
 void *provided_cbdata,
 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

|           |                        |                                                                                                       |
|-----------|------------------------|-------------------------------------------------------------------------------------------------------|
| <b>IN</b> | <b>status</b>          | returned status of the request ( <a href="#">pmix_status_t</a> )                                      |
| <b>IN</b> | <b>info</b>            | Array of info structures (array of handles)                                                           |
| <b>IN</b> | <b>ninfo</b>           | Number of elements in the <i>info</i> array (integer)                                                 |
| <b>IN</b> | <b>provided_cbdata</b> | Data originally passed to call to <a href="#">PMIx_server_setup_application</a> (memory reference)    |
| <b>IN</b> | <b>cbfunc</b>          | <a href="#">pmix_op_cbfunc_t</a> function to be called when processing completed (function reference) |
| <b>IN</b> | <b>cbdata</b>          | Data to be passed to the <i>cbfunc</i> callback function (memory reference)                           |

## Description

Define a function to be called by the PMIx server library for return of application-specific setup data in response to a request from the host RM. The returned *info* array is owned by the PMIx server library and will be free'd when the provided *cbfunc* is called.

## 24 16.2.11.2 Server Setup Application Attributes

25 *PMIx v3.0* Attributes specifically defined for controlling contents of application setup data.

**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.

## 32 16.2.12 PMIx\_Register\_attributes

## Summary

Register host environment attribute support for a function.

1           **Format**

C

```
2 pmix_status_t
3 PMIx_Register_attributes(char *function,
4 pmix_regattr_t attrs[],
5 size_t nattrs);
```

C

6     **IN**    **function**

7       String name of function (string)

8     **IN**    **attrs**

9       Array of **pmix\_regattr\_t** describing the supported attributes (handle)

10    **IN**    **nattrs**

11      Number of elements in *attrs* (**size\_t**)

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

13           **Description**

14     The **PMIx\_Register\_attributes** function is used by the host environment to register with  
15     its PMIx server library the attributes it supports for each **pmix\_server\_module\_t** function.  
16     The *function* is the string name of the server module function (e.g., "register\_events",  
17     "validate\_credential", or "allocate") whose attributes are being registered. See the  
18     **pmix\_regattr\_t** entry for a description of the *attrs* array elements.

19     Note that the host environment can also query the library (using the **PMIx\_Query\_info\_nb**  
20     API) for its attribute support both at the server, client, and tool levels once the host has executed  
21     **PMIx\_server\_init** since the server will internally register those values.

22           **Advice to PMIx server hosts**

23     Host environments are strongly encouraged to register all supported attributes immediately after  
initializing the library to ensure that user requests are correctly serviced.

## Advice to PMIx library implementers

1 PMIx implementations are *required* to register all internally supported attributes for each API  
2 during initialization of the library (i.e., when the process calls their respective PMIx init function).  
3 Specifically, the implementation *must not* register supported attributes upon first call to a given API  
4 as this would prevent users from discovering supported attributes prior to first use of an API.

5 It is the implementation's responsibility to associate registered attributes for a given  
6 `pmix_server_module_t` function with their corresponding user-facing API. Supported  
7 attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by  
8 the level (see Section 5.4.6) at which the attribute is supported.

9 Note that attributes can/will be registered on an API for each level. It is *required* that the  
10 implementation support user queries for supported attributes on a per-level basis. Duplicate  
11 registrations at the *same* level for a function *shall* return an error - however, duplicate registrations  
12 at *different* levels *shall* be independently tracked.

### 16.2.12.1 Attribute registration constants

14 Constants supporting attribute registration.

15 **PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION** The attributes for an identical function have  
16 already been registered at the specified level (host, server, or client).

### 16.2.12.2 Attribute registration structure

18 The `pmix_regattr_t` structure is used to register attribute support for a PMIx function.

PMIx v4.0

C

```
19 typedef struct pmix_regattr {
20 char *name;
21 pmix_key_t *string;
22 pmix_data_type_t type;
23 pmix_info_t *info;
24 size_t ninfo;
25 char **description;
26 } pmix_regattr_t;
```

C

27 Note that in this structure:

- 28 • the *name* is the actual name of the attribute - e.g., "PMIX\_MAX\_PROCS"  
29 • the *string* is the literal string value of the attribute - e.g., "pmix.max.size" for the  
30 **PMIX\_MAX\_PROCS** attribute  
31 • *type* must be a PMIx data type identifying the type of data associated with this attribute.

- the *info* array contains machine-useable information regarding the range of accepted values. This may include entries for **PMIX\_MIN\_VALUE**, **PMIX\_MAX\_VALUE**, **PMIX\_ENUM\_VALUE**, or a combination of them. For example, an attribute that supports all positive integers might delineate it by including a **pmix\_info\_t** with a key of **PMIX\_MIN\_VALUE**, type of **PMIX\_INT**, and value of zero. The lack of an entry for **PMIX\_MAX\_VALUE** indicates that there is no ceiling to the range of accepted values.
- ninfo* indicates the number of elements in the *info* array
- The *description* field consists of a **NULL**-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values - e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the *description* and the number of elements in the *info* array is implied or required.

The attribute *name* and *string* fields must be **NULL**-terminated strings composed of standard alphanumeric values supported by common utilities such as *strcmp*.

Although not strictly required, both PMIx library implementers and host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them.

### 16.2.12.3 Attribute registration structure descriptive attributes

The following attributes relate to the nature of the values being reported in the **pmix\_regattr\_t** structures.

**PMIX\_MAX\_VALUE** "pmix.descr.maxval" (**varies**)  
 Used in **pmix\_regattr\_t** to describe the maximum valid value for the associated attribute.

**PMIX\_MIN\_VALUE** "pmix.descr.minval" (**varies**)  
 Used in **pmix\_regattr\_t** to describe the minimum valid value for the associated attribute.

**PMIX\_ENUM\_VALUE** "pmix.descr.enum" (**char\***)  
 Used in **pmix\_regattr\_t** to describe accepted values for the associated attribute.  
 Numerical values shall be presented in a form convertible to the attribute's declared data type. Named values (i.e., values defined by constant names via a typical C-language enum declaration) must be provided as their numerical equivalent.

### 16.2.12.4 Attribute registration structure support macros

The following macros are provided to support the **pmix\_regattr\_t** structure.

#### Initialize the regattr structure

Initialize the **pmix\_regattr\_t** fields

```
1 PMIX_REGATTR_CONSTRUCT(m) C
2 IN m C
3 Pointer to the structure to be initialized (pointer to pmix_regattr_t)
4 Destruct the regattr structure
5 Destruct the pmix_regattr_t fields, releasing all strings.
PMIx v4.0 C
6 PMIX_REGATTR_DESTRUCT(m) C
7 IN m C
8 Pointer to the structure to be destructed (pointer to pmix_regattr_t)
9 Create a regattr array
10 Allocate and initialize an array of pmix_regattr_t structures.
PMIx v4.0 C
11 PMIX_REGATTR_CREATE(m, n) C
12 INOUT m C
13 Address where the pointer to the array of pmix_regattr_t structures shall be stored
14 (handle)
15 IN n C
16 Number of structures to be allocated (size_t)
17 Free a regattr array
18 Release an array of pmix_regattr_t structures.
PMIx v4.0 C
19 PMIX_REGATTR_FREE(m, n) C
20 INOUT m C
21 Pointer to the array of pmix_regattr_t structures (handle)
22 IN n C
23 Number of structures in the array (size_t)
```

## 1 Load a regattr structure

2 Load values into a `pmix_regattr_t` structure. The macro can be called multiple times to add as  
3 many strings as desired to the same structure by passing the same address and a `NULL` key to the  
4 macro. Note that the *t* type value must be given each time.

5 `PMIX_REGATTR_LOAD(a, n, k, t, ni, v)`

6   **IN** `a`  
7       Pointer to the structure to be loaded (pointer to `pmix_proc_t`)  
8   **IN** `n`  
9       String name of the attribute (string)  
10   **IN** `k`  
11       Key value to be loaded (`pmix_key_t`)  
12   **IN** `t`  
13       Type of data associated with the provided key (`pmix_data_type_t`)  
14   **IN** `ni`  
15       Number of `pmix_info_t` elements to be allocated in *info* (`size_t`)  
16   **IN** `v`  
17       One-line description to be loaded (more can be added separately) (string)

## 18 Transfer a regattr to another regattr

19 Non-destructively transfer the contents of a `pmix_regattr_t` structure to another one.

PMIx v4.0

20 `PMIX_REGATTR_XFER(m, n)`

21   **INOUT** `m`  
22       Pointer to the destination `pmix_regattr_t` structure (handle)  
23   **IN** `m`  
24       Pointer to the source `pmix_regattr_t` structure (handle)

## 25 16.2.13 PMIx\_server\_setup\_local\_support

### 26 Summary

27 Provide a function by which the local PMIx server can perform any application-specific operations  
28 prior to spawning local clients of a given application.

## Format

8

```
pmix_status_t
PMIx_server_setup_local_support(const pmix_nspace_t nspace,
 pmix_info_t info[], size_t ninfo,
 pmix_op_cfunc_t cfunc,
 void *cbdata);
```

## 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_cfunc_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)

Returns one of the following:

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

## Description

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a fabric library might need to setup the local driver for “instant on” addressing. The data provided in the *info* array is the data returned to the host RM by the callback function executed as a result of a call to

## **PMIx\_server\_setup\_application**

## Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting any local application processes from the specified namespace if information was obtained from a call to [PMIx\\_server\\_setup\\_application](#).

1 Host environments must register the *nspace* 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 16.2.14 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

```
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 Returns one of the following:

- 32 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
33 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
34 function prior to returning from the API.

- **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

## Description

Provide a function by which the 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.

## 16.2.15 PMIx server collect inventory

## Summary

Collect inventory of resources on a node.

## Format

```
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)

Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. In the event the function returns an error, the *cbfunc* will not be called.

1           **Description**  
2         Provide a function by which the host environment can request its PMIx server library collect an  
3         inventory of local resources. Supported resources depends upon the PMIx implementation, but may  
4         include the local node topology and fabric interfaces.

5           **Advice to PMIx server hosts**

6         This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested  
7         information. Inventory collection is expected to be a rare event – at system startup and upon  
8         command from a system administrator. Inventory updates are expected to initiate a smaller  
9         operation involving only the changed information. For example, replacement of a node would  
10        generate an event to notify the scheduler with an inventory update without invoking a global  
       inventory operation.

11       **16.2.16 PMIx\_server\_deliver\_inventory**

12       **Summary**

13       Pass collected inventory to the PMIx server library for storage.

14       **Format**

15           `pmix_status_t`  
16           `PMIx_server_deliver_inventory(const pmix_info_t info[],`  
17                            `size_t ninfo,`  
18                            `const pmix_info_t directives[],`  
19                            `size_t ndirs,`  
20                            `pmix_op_cbfunc_t cbfunc,`  
21                            `void *cbdata);`

- 22       **IN info**  
23        Array of `pmix_info_t` containing the inventory (array of handles)  
24       **IN ninfo**  
25        Number of elements in the *info* array (`size_t`)  
26       **IN directives**  
27        Array of `pmix_info_t` directing the request (array of handles)  
28       **IN ndirs**  
29        Number of elements in the *directives* array (`size_t`)  
30       **IN cbfunc**  
31        Callback function `pmix_op_cbfunc_t`. A `NULL` function reference indicates that the  
32        function is to be executed as a blocking operation (function reference)  
33       **IN cbdata**  
34        Data to be passed to the callback function (memory reference)

Returns one of the following:

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

## Description

Provide a function by which the host environment can pass inventory information obtained from a node (as a result of a call to [PMIx\\_server\\_collect\\_inventory](#)) to the PMIx server library for storage. Inventory data is subsequently used by the PMIx server library for allocations in response to [PMIx\\_server\\_setup\\_application](#), and may be available to the library's host via the [PMIx\\_Get](#) API (depending upon PMIx implementation). The *cfunc* callback function will be called once the PMIx server library no longer requires access to the provided data.

### 16.2.17 PMIx\_server\_generate\_locality\_string

## Summary

Generate a PMIx locality string from a given cpuset.

## **Format**

```
pmix_status_t
PMIx_server_generate_locality_string(const pmix_cpuset_t *cpuset,
 char **locality);
```

IN cpuset

Pointer to a `pmix_cpuset_t` containing the bitmap of assigned PUs (handle)

## OUT locality

String representation of the PMIx locality corresponding to the input bitmap (**char\***)

Returns either **PMIX\_SUCCESS** appropriate PMIx error constant.

## Description

Provide a function by which the host environment can generate a PMIx locality string for inclusion in the call to `PMIx_server_register_nspace`. 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.

### 16.2.18 PMIx\_server\_generate\_cpuset\_string

## Summary

Generate a PMIx string representation of the provided cpuset.

## Format

IN cpuset

Pointer to a `pmix_cpuset_t` containing the bitmap of assigned PUs (handle)

**OUT** cpuset\_string

String representation of the input bitmap (**char\***)

Returns either **PMIX\_SUCCESS** indicating that the returned string contains the representation, or an appropriate PMIx error constant.

## Description

Provide a function by which the host environment can generate a string representation of the cpuset bitmap for inclusion in the call to `PMIx_server_register_nspace`. This function shall only be called for local client processes, with the returned string included in the job-level information (via the `PMIX_CPUSET` attribute) provided to local clients. Local clients can use these strings as input to obtain their PU bindings via the `PMIx 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.

## 16.2.18.1 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`).

```
5 typedef struct pmix_cpuset {
6 char *source;
7 void *bitmap;
8 } pmix_cpuset_t;
```

## 16.2.18.2 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

```
13 PMIX_CPUSET_CONSTRUCT(m)
```

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

```
18 PMIX_CPUSET_DESTRUCT(m)
```

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

```
23 PMIX_CPUSET_CREATE(m, n)
```

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)

## Release a cpuset array

Deconstruct and free a `pmix_cpuset_t` array.

PMIx v4.0

### PMIX CPuset FREE(m, n)

Q

**INOUT** *m*

Address the array of `pmix_cpuset_t` structures to be released (handle)

IN n

Number of structures in the array (size\_t)

### **16.2.19 PMIx\_server\_define\_process\_set**

## Summary

Define a PMIx process set.

## Format

```
pmix_status_t
PMIx_server_define_process_set(const pmix_proc_t members[],
 size_t nmembers,
 char *pset name);
```

IN members

Pointer to an array of `pmix_proc_t` containing the identifiers of the processes in the process set (handle).

#### **IN numbers**

Number of elements in *members* (integer)

**IN**    *iset\_name*

**process\_name**  
String name of the process set being defined (**char\***)

Returns either **PMIX\_SUCCESS** or an appropriate PMIx error constant.

## Description

Provide a function by which the host environment can create a process set. The PMIx server shall alert all local clients of the new process set (including process set name and membership) via the [PMIX\\_PROCESS\\_SET\\_DEFINE](#) event.

## 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

## 16.2.20 PMIx\_server\_delete\_process\_set

### Summary

Delete a PMIx process set name

### Format

```
pmix_status_t
PMIx_server_delete_process_set(char *pset_name);
```

#### IN pset\_name

String name of the process set being deleted (`char*`)

Returns either `PMIX_SUCCESS` or an appropriate PMIx error constant.

### Description

Provide a function by which the host environment can delete a process set name. The PMIx server shall alert all local clients of the process set name being deleted via the `PMIX_PROCESS_SET_DELETE` event. Deletion of the name has no impact on the member processes.

### Advice to PMIx server hosts

The host environment is responsible for ensuring consistent knowledge of process set membership across all involved PMIx servers.

## 16.3 Server Function Pointers

PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the protocol. This method allows RMs to implement the server without being burdened with PMIx internal details. When a request is received from the client, the corresponding server function will be called with the information.

Any functions not supported by the RM can be indicated by a `NULL` for the function pointer. PMIx implementations are required to return a `PMIX_ERR_NOT_SUPPORTED` status to all calls to functions that require host environment support and are not backed by a corresponding server module entry. Host environments may, if they choose, include a function pointer for operations they have not yet implemented and simply return `PMIX_ERR_NOT_SUPPORTED`.

Functions that accept directives (i.e., arrays of `pmix_info_t` structures) must check any provided directives for those marked as *required* via the `PMIX_INFO_REQD` flag. PMIx client and server libraries are required to mark any such directives with the `PMIX_INFO_REQD_PROCESSED` flag should they have handled the request. Any required directive that has not been marked therefore 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

## 16.3.1 pmix\_server\_module\_t Module

### Summary

List of function pointers that a PMIx server passes to **PMIx\_server\_init** during startup.

### Format

C

```
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_fencenb_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

Note that some PMIx implementations *require* the use of C99-style designated initializers to clearly correlate each provided function pointer with the correct member of the `pmix_server_module_t` structure as the location/ordering of struct members may change over time.

## 16.3.2 `pmix_server_client_connected_fn_t`

### Summary

Notify the host server that a client connected to this server. This function module entry has been **DEPRECATED** in favor of `pmix_server_client_connected2_fn_t`.

1      **Format**

2      **C**

```
3 typedef pmix_status_t (*pmix_server_client_connected_fn_t) (
4 const pmix_proc_t *proc,
5 void* server_object,
6 pmix_op_cbfunc_t cbfunc,
7 void *cbdata);
```

8      **C**

9      **IN proc**

10     **pmix\_proc\_t** structure (handle)

11     **IN server\_object**

12     object reference (memory reference)

13     **IN cbfunc**

14     Callback function **pmix\_op\_cbfunc\_t** (function reference)

15     **IN cbdata**

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

17     Returns one of the following:

- 18       • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
19       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
20       prior to returning from the API.
- 21       • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
22       returned *success* - the *cbfunc* will not be called
- 23       • a PMIx error constant indicating either an error in the input or that the request was immediately  
24       processed and failed - the *cbfunc* will not be called

25      **Description**

26      This function module entry has been DEPRECATED in favor of

27      **pmix\_server\_client\_connected2\_fn\_t**. If both functions are provided, the PMIx  
28      library will ignore this function module entry in favor of its replacement.

29      **16.3.3 pmix\_server\_client\_connected2\_fn\_t**

30      **Summary**

31      Notify the host server that a client connected to this server - this version of the original function  
32      definition has been extended to include an array of **pmix\_info\_t**, thereby allowing the PMIx  
33      server library to pass additional information identifying the client to the host environment.

## Format

```
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)
```

|           |                      |                                                       |
|-----------|----------------------|-------------------------------------------------------|
| <b>IN</b> | <b>proc</b>          | pmix_proc_t structure (handle)                        |
| <b>IN</b> | <b>server_object</b> | object reference (memory reference)                   |
| <b>IN</b> | <b>info</b>          | Array of info structures (array of handles)           |
| <b>IN</b> | <b>ninfo</b>         | Number of elements in the <i>info</i> array (integer) |
| <b>IN</b> | <b>cbfunc</b>        | Callback function pmix_op_cbfunc_t (fun               |
| <b>IN</b> | <b>cbdta</b>         | Data to be passed to the callback function (mem       |

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` 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 **16.3.4 `pmix_server_client_finalized_fn_t`**

6 **Summary**

7 Notify the host environment that a client called `PMIx_Finalize`.

8 **Format**

```
9 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);
```

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)

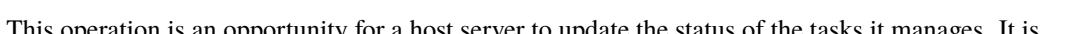
22 Returns one of the following:

- `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

1           **Description**

2       Notify the host environment that a client called **PMIx\_Finalize**. Note that the client will be in a  
3       blocked state until the host server executes the callback function, thus allowing the PMIx server  
4       support library to release the client. The **server\_object** parameter will be the value of the  
5       **server\_object** parameter passed to **PMIx\_server\_register\_client** by the host server when  
6       registering the connecting client. If provided, an implementation of  
7       **pmix\_server\_client\_finalized\_fn\_t** is only required to call the callback function  
8       designated. A host server can choose to not be notified when clients finalize by setting  
9       **pmix\_server\_client\_finalized\_fn\_t** to **NULL**.

10      Note that the host server is only being informed that the client has called **PMIx\_Finalize**. The  
11     client might not have exited. If a client exits without calling **PMIx\_Finalize**, the server support  
12     library will not call the **pmix\_server\_client\_finalized\_fn\_t** implementation.

13            **Advice to PMIx server hosts** 

14      This operation is an opportunity for a host server to update the status of the tasks it manages. It is  
also a convenient and well defined time to release resources used to support that client.  


15      **16.3.5 pmix\_server\_abort\_fn\_t**

16           **Summary**

17      Notify the host environment that a local client called **PMIx\_Abort**.

18      **Format**  C 

19      

```
typedef pmix_status_t (*pmix_server_abort_fn_t) (
20 const pmix_proc_t *proc,
21 void *server_object,
22 int status,
23 const char msg[],
24 pmix_proc_t procs[],
25 size_t nprocs,
26 pmix_op_cbfunc_t cbfunc,
27 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
19 Returns one of the following:
20
21 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
22 will be returned in the provided cbfunc. Note that the host must not invoke the callback function
23 prior to returning from the API.
24
25 • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
26 returned success - the cbfunc will not be called
27
28 • PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED indicating that the host environment supports
29 this API, but the request includes processes that the host environment cannot abort - e.g., if the
30 request is to abort subsets of processes from a namespace, or processes outside of the caller's
31 own namespace, and the host environment does not permit such operations. In this case, none of
32 the specified processes will be terminated - the cbfunc will not be called
33
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
38 • a PMIx error constant indicating either an error in the input or that the request was immediately
39 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**.

### **16.3.6 pmix\_server\_fencenb\_fn\_t**

## Summary

At least one client called either **PMIx\_Fence** or **PMIx\_Fence\_nb**.

## Format

```
typedef pmix_status_t (*pmix_server_fencenb_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);
```

**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)

- 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

9                                  **Required Attributes**

10      PMIx libraries are required to pass any provided attributes to the host environment for processing.

11      The following attributes are required to be supported by all host environments:

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

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

18                                  **Optional Attributes**

19      The following attributes are optional for host environments:

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

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

24                                  **Advice to PMIx server hosts**

25      Host environment are required to return **PMIX\_ERR\_NOT\_SUPPORTED** if passed an attributed  
26       marked as **PMIX\_INFO\_REQD** that they do not support, even if support for that attribute is  
27       optional.

## 1      Description

2      All local clients in the provided array of *procs* called either **PMIx\_Fence** or **PMIx\_Fence\_nb**.  
3      In either case, the host server will be called via a non-blocking function to execute the specified  
4      operation once all participating local processes have contributed. All processes in the specified  
5      *procs* array are required to participate in the **PMIx\_Fence/PMIx\_Fence\_nb** operation. The  
6      callback is to be executed once every daemon hosting at least one participant has called the host  
7      server's **pmix\_server\_fencenb\_fn\_t** function.

8      The provided data is to be collectively shared with all PMIx servers involved in the fence operation,  
9      and returned in the modeX *cbfunc*. A **NULL** data value indicates that the local processes had no data  
10     to contribute.

11     The array of *info* structs is used to pass user-requested options to the server. This can include  
12     directives as to the algorithm to be used to execute the fence operation. The directives are optional  
13     unless the **PMIX\_INFO REQD** flag has been set - in such cases, the host RM is required to return  
14     an error if the directive cannot be met.

### 15     ────────── Advice to PMIx library implementers ──────────

16     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.

### 17     ────────── Advice to PMIx server hosts ──────────

18     The host will receive a single call for each collective operation. It is the responsibility of the host to  
19     identify the nodes containing participating processes, execute the collective across all participating  
20     nodes, and notify the local PMIx server library upon completion of the global collective. Data  
21     received from each node must be simply concatenated to form an aggregated unit, as shown in the  
following example:

22     **C**

```
23 uint8_t *blob1, *blob2, *total;
24 size_t sz_blob1, sz_blob2, sz_total;
25
26 sz_total = sz_blob1 + sz_blob2;
27 total = (uint8_t*)malloc(sz_total);
28 memcpy(total, blob1, sz_blob1);
29 memcpy(&total[sz_blob1], blob2, sz_blob2);
```

30     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.

## 16.3.6.1 Modex Callback Function

### Summary

The `pmix_modex_cbfunc_t` is used by the `pmix_server_fencenb_fn_t` and `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex Business Card Exchange (BCX) data.

PMIx v1.0

C

```
6 typedef void (*pmix_modex_cbfunc_t)
7 (pmix_status_t status,
8 const char *data, size_t ndata,
9 void *cbdata,
10 pmix_release_cbfunc_t release_fn,
11 void *release_cbdata);
```

C

#### IN `status`

Status associated with the operation (handle)

#### IN `data`

Data to be passed (pointer)

#### IN `ndata`

size of the data (`size_t`)

#### IN `cbdata`

Callback data passed to original API call (memory reference)

#### IN `release_fn`

Callback for releasing *data* (function pointer)

#### IN `release_cbdata`

Pointer to be passed to *release\_fn* (memory reference)

### Description

A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data in response to “fence” and “get” operations. The returned blob contains the data collected from each server participating in the operation.

## 16.3.7 `pmix_server_dmodex_req_fn_t`

### Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified process to obtain and return a direct modex blob for that process.

1 Format

C

```
2 typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
3 const pmix_proc_t *proc,
4 const pmix_info_t info[],
5 size_t ninfo,
6 pmix_modex_cfunc_t cfunc,
7 void *cbdata);
```

C

8 **IN proc**  
9     **pmix\_proc\_t** structure identifying the process whose data is being requested (handle)  
10 **IN info**  
11     Array of info structures (array of handles)  
12 **IN ninfo**  
13     Number of elements in the *info* array (integer)  
14 **IN cfunc**  
15     Callback function **pmix\_modex\_cfunc\_t** (function reference)  
16 **IN cbdata**  
17     Data to be passed to the callback function (memory reference)

18 Returns one of the following:

- 19
  - **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
20         will be returned in the provided *cfunc*. Note that the host must not invoke the callback function  
21         prior to returning from the API.
  - **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
22         request, even though the function entry was provided in the server module - the *cfunc* will not  
23         be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately  
24         processed and failed - the *cfunc* will not be called

25 **Required Attributes**

27 PMIx libraries are required to pass any provided attributes to the host environment for processing.

28 All host environments are required to support the following attributes:

29 **PMIX\_REQUIRED\_KEY "pmix.req.key" (char\*)**

30     Identifies a key that must be included in the requested information. If the specified key is not  
31         already available, then the PMIx servers are required to delay response to the dmodex  
32         request until either the key becomes available or the request times out.

## Optional Attributes

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

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

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

### 6 **Description**

7 Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
8 hosts the specified proc to obtain and return any information that process posted via calls to  
9 **PMIx\_Put** and **PMIx\_Commit**.

10 The array of *info* structs is used to pass user-requested options to the server. This can include a  
11 timeout to preclude an indefinite wait for data that may never become available. The directives are  
12 optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an  
13 error if the directive cannot be met.

#### 14 **16.3.7.1 Dmodex attributes**

15 **PMIX\_REQUIRED\_KEY** "pmix.req.key" (char\*)

16 Identifies a key that must be included in the requested information. If the specified key is not  
17 already available, then the PMIx servers are required to delay response to the dmodex  
18 request until either the key becomes available or the request times out.

#### 19 **16.3.8 pmix\_server\_publish\_fn\_t**

##### 20 **Summary**

21 Publish data per the PMIx API specification.

##### 22 **Format**

C

```
23 typedef pmix_status_t (*pmix_server_publish_fn_t)(
24 const pmix_proc_t *proc,
25 const pmix_info_t info[],
26 size_t ninfo,
27 pmix_op_cbfunc_t cbfunc,
28 void *cbdata);
```

```

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
12 Returns one of the following:
13
14 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
15 will be returned in the provided cbfunc. Note that the host must not invoke the callback function
16 prior to returning from the API.
17
18 • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
19 returned success - the cbfunc will not be called
20
21 • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
22 request, even though the function entry was provided in the server module - the cbfunc will not
23 be called
24
25 • a PMIx error constant indicating either an error in the input or that the request was immediately
26 processed and failed - the cbfunc will not be called

```

### Required Attributes

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:

```

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

```

---

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

9      Publish data per the **PMIx\_Publish** specification. The callback is to be executed upon  
10     completion of the operation. The default data range is left to the host environment, but expected to  
11     be **PMIX\_RANGE\_SESSION**, and the default persistence **PMIX\_PERSIST\_SESSION** or their  
12     equivalent. These values can be specified by including the respective attributed in the *info* array.

13     The persistence indicates how long the server should retain the data.

## Advice to PMIx server hosts

14     The host environment is not required to guarantee support for any specific range - i.e., the  
15     environment does not need to return an error if the data store doesn't support a specified range so  
16     long as it is covered by some internally defined range. However, the server must return an error (a)  
17     if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of  
18     published info by the original publisher - it is left to the discretion of the host environment to allow  
19     info-key-based flags to modify this behavior.

20     The **PMIX\_USERID** and **PMIX\_GRPID** of the publishing process will be provided to support  
21     authorization-based access to published information and must be returned on any subsequent  
22     lookup request.

### 16.3.9 pmix\_server\_lookup\_fn\_t

#### Summary

Lookup published data.

## 1 Format

C

```
2 typedef pmix_status_t (*pmix_server_lookup_fn_t) (
3 const pmix_proc_t *proc,
4 char **keys,
5 const pmix_info_t info[],
6 size_t ninfo,
7 pmix_lookup_cbfunc_t cbfunc,
8 void *cbdata) ;
```

9     **IN proc**  
10    **pmix\_proc\_t** structure of the process seeking the data (handle)

11     **IN keys**  
12    (array of strings)

13     **IN info**  
14    Array of info structures (array of handles)

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

17     **IN cbfunc**  
18    Callback function **pmix\_lookup\_cbfunc\_t** (function reference)

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

21 Returns one of the following:

- 22     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24       prior to returning from the API.
- 25     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26       returned *success* - the *cbfunc* will not be called
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28       request, even though the function entry was provided in the server module - the *cbfunc* will not  
29       be called
- 30     • a PMIx error constant indicating either an error in the input or that the request was immediately  
31       processed and failed - the *cbfunc* will not be called

32                      **Required Attributes**

33 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
In addition, the following attributes are required to be included in the passed *info* array:

34     **PMIX\_USERID "pmix.euid" (uint32\_t)**  
35       Effective user ID of the connecting process.

```
1 PMIX_GRP_ID "pmix.egid" (uint32_t)
2 Effective group ID of the connecting process.
```

3 Host environments that implement this entry point are required to support the following attributes:

```
4 PMIX_RANGE "pmix.range" (pmix_data_range_t)
5 Define constraints on the processes that can access the provided data. Only processes that
6 meet the constraints are allowed to access it.
```

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

## Optional Attributes

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

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

## Description

17 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys  
18 identifying the data being requested.

19 The array of *info* structs is used to pass user-requested options to the server. The default data range  
20 is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**. This can include a  
21 wait flag to indicate that the server should wait for all data to become available before executing the  
22 callback function, or should immediately callback with whatever data is available. In addition, a  
23 timeout can be specified on the wait to preclude an indefinite wait for data that may never be  
24 published.

## Advice to PMIx server hosts

25 The **PMIX\_USERID** and **PMIX\_GRP\_ID** of the requesting process will be provided to support  
26 authorization-based access to published information. The host environment is not required to  
27 guarantee support for any specific range - i.e., the environment does not need to return an error if  
28 the data store doesn't support a specified range so long as it is covered by some internally defined  
29 range.

1 **16.3.10 pmix\_server\_unpublish\_fn\_t**

2     **Summary**

3     Delete data from the data store.

4     **Format**

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

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
2 In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (`uint32_t`)

4     Effective user ID of the connecting process.

5 **PMIX\_GRPID** "pmix.egid" (`uint32_t`)

6     Effective group ID of the connecting process.

---

7 Host environments that implement this entry point are required to support the following attributes:  
8

9 **PMIX\_RANGE** "pmix.range" (`pmix_data_range_t`)

10     Define constraints on the processes that can access the provided data. Only processes that  
11     meet the constraints are allowed to access it.

## Optional Attributes

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

13 **PMIX\_TIMEOUT** "pmix.timeout" (`int`)

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

## 17 **Description**

18 Delete data from the data store. The host server will be passed a **NULL**-terminated array of string  
19 keys, plus potential directives such as the data range within which the keys should be deleted. The  
20 default data range is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**.  
21 The callback is to be executed upon completion of the delete procedure.

## Advice to PMIx server hosts

22 The **PMIX\_USERID** and **PMIX\_GRPID** of the requesting process will be provided to support  
23 authorization-based access to published information. The host environment is not required to  
24 guarantee support for any specific range - i.e., the environment does not need to return an error if  
25 the data store doesn't support a specified range so long as it is covered by some internally defined  
26 range.

## 16.3.11 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 • [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
29 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
30 prior to returning from the API.
- 31 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
32 returned *success* - the *cbfunc* will not be called
- 33 • [PMIX\\_ERR\\_NOT\\_SUPPORTED](#), indicating that the host environment does not support the  
34 request, even though the function entry was provided in the server module - the *cbfunc* will not  
35 be called
- 36 • a PMIx error constant indicating either an error in the input or that the request was immediately  
37 processed and failed - the *cbfunc* will not be called

## Required Attributes

1 PMIx server libraries are required to pass any provided attributes to the host environment for  
2 processing. In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
4     Effective user ID of the connecting process.

5 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
6     Effective group ID of the connecting process.

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

10 **PMIX\_PARENT\_ID** "pmix.parent" (**pmix\_proc\_t**)  
11     Process identifier of the parent process of the specified process - typically used to identify  
12     the application process that caused the job containing the specified process to be spawned  
13     (e.g., the process that called **PMIx\_Spawn**).

14 **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (**bool**)  
15     The requesting process is a PMIx tool.

16 **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (**bool**)  
17     The requesting process is a PMIx client.

---

18

19 Host environments that provide this module entry point are required to pass the **PMIX\_SPAWNED**  
20 and **PMIX\_PARENT\_ID** attributes to all PMIx servers launching new child processes so those  
21 values can be returned to clients upon connection to the PMIx server. In addition, they are required  
22 to support the following attributes when present in either the *job\_info* or the *info* array of an  
23 element of the *apps* array:

24 **PMIX\_WDIR** "pmix.wdir" (**char\***)  
25     Working directory for spawned processes.

26 **PMIX\_SET\_SESSION\_CWD** "pmix.ssncwd" (**bool**)  
27     Set the current working directory to the session working directory assigned by the RM - can  
28     be assigned to the entire job (by including attribute in the *job\_info* array) or on a  
29     per-application basis in the *info* array for each **pmix\_app\_t**.

30 **PMIX\_PREFIX** "pmix.prefix" (**char\***)  
31     Prefix to use for starting spawned processes - i.e., the directory where the executables can be  
32     found.

33 **PMIX\_HOST** "pmix.host" (**char\***)  
34     Comma-delimited list of hosts to use for spawned processes.

35 **PMIX\_HOSTFILE** "pmix.hostfile" (**char\***)

1 Hostfile to use for spawned processes.

## Optional Attributes

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

3 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (**char\***)  
4 Hostfile containing hosts to add to existing allocation.

5 **PMIX\_ADD\_HOST** "pmix.addhost" (**char\***)  
6 Comma-delimited list of hosts to add to the allocation.

7 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (**bool**)  
8 Preload executables onto nodes prior to executing launch procedure.

9 **PMIX\_PRELOAD\_FILES** "pmix.loadfiles" (**char\***)  
10 Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

11 **PMIX\_PERSONALITY** "pmix.pers" (**char\***)  
12 Name of personality corresponding to programming model used by application - supported  
13 values depend upon PMIx implementation.

14 **PMIX\_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 Spawns 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_CPU_PER_PROC "pmix.cpuperproc" (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.repbind" (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.

### 16.3.11.1 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.
```

### 16.3.12 pmix\_server\_connect\_fn\_t

#### Summary

Record the specified processes as *connected*.

#### Format

```
9 PMIx v1.0
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);
```

#### 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

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.

### 16.3.13 pmix server disconnect fn t

## Summary

Disconnect a previously connected set of processes.

## 1 Format

C

```
2 typedef pmix_status_t (*pmix_server_disconnect_fn_t)(
3 const pmix_proc_t procs[],
4 size_t nprocs,
5 const pmix_info_t info[],
6 size_t ninfo,
7 pmix_op_cbfunc_t cbfunc,
8 void *cbdata);
```

9 **IN procs**10 Array of **pmix\_proc\_t** structures identifying participants (array of handles)11 **IN nprocs**12 Number of elements in the *procs* array (integer)13 **IN info**

14 Array of info structures (array of handles)

15 **IN ninfo**16 Number of elements in the *info* array (integer)17 **IN cbfunc**18 Callback function **pmix\_op\_cbfunc\_t** (function reference)19 **IN cbdata**

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

21 Returns one of the following:

- **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**

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.

## Optional Attributes

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

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

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

### 6   **Description**

7   Disconnect a previously connected set of processes. The callback is to be executed once every  
8   daemon hosting at least one participant has called the host server's has called the  
9   **pmix\_server\_disconnect\_fn\_t** function, and the host environment has completed any  
10   required supporting operations.

### 7   **Advice to PMIx library implementers**

11   The PMIx server library is required to aggregate participation by local clients, passing the request  
12   to the host environment once all local participants have executed the API.

### 8   **Advice to PMIx server hosts**

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

16   A **PMIX\_ERR\_INVALID\_OPERATION** error must be returned if the specified set of *procs* was  
17   not previously *connected* via a call to the **pmix\_server\_connect\_fn\_t** function.

## 18   **16.3.14 pmix\_server\_register\_events\_fn\_t**

### 19   **Summary**

20   Register to receive notifications for the specified events.

## 1 Format

C

```

2 typedef pmix_status_t (*pmix_server_register_events_fn_t)(
3 pmix_status_t *codes,
4 size_t ncodes,
5 const pmix_info_t info[],
6 size_t ninfo,
7 pmix_op_cbfunc_t cbfunc,
8 void *cbdata);
```

9 **IN codes**10 Array of **pmix\_status\_t** values (array of handles)11 **IN ncodes**12 Number of elements in the *codes* array (integer)13 **IN info**

14 Array of info structures (array of handles)

15 **IN ninfo**16 Number of elements in the *info* array (integer)17 **IN cbfunc**18 Callback function **pmix\_op\_cbfunc\_t** (function reference)19 **IN cbdata**

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

21 Returns one of the following:

- 22 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24 prior to returning from the API.
- 25 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26 returned *success* - the *cbfunc* will not be called
- 27 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28 request, even though the function entry was provided in the server module - the *cbfunc* will not  
29 be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the *cbfunc* will not be called

**Required Attributes**

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
33 In addition, the following attributes are required to be included in the passed *info* array:

34 **PMIX\_USERID "pmix.euid" (uint32\_t)**

35       Effective user ID of the connecting process.

1       **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2              Effective group ID of the connecting process.

3       **Description**

4       Register to receive notifications for the specified status codes. The *info* array included in this API is  
5       reserved for possible future directives to further steer notification.

6               **Advice to PMIx library implementers**

7       The PMIx server library must track all client registrations for subsequent notification. This module  
8       function shall only be called when:

- 9
  - the client has requested notification of an environmental code (i.e., a PMIx codes in the range  
10          between **PMIX\_EVENT\_SYS\_BASE** and **PMIX\_EVENT\_SYS\_OTHER**, inclusive) or codes that  
          lies outside the defined PMIx range of constants; and
  - the PMIx server library has not previously requested notification of that code - i.e., the host  
          environment is to be contacted only once a given unique code value

11               **Advice to PMIx server hosts**

12       The host environment is required to pass to its PMIx server library all non-environmental events  
13       that directly relate to a registered namespace without the PMIx server library explicitly requesting  
14       them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in  
15       the range between **PMIX\_EVENT\_SYS\_BASE** and **PMIX\_EVENT\_SYS\_OTHER** (inclusive).

17     **16.3.15 pmix\_server\_deregister\_events\_fn\_t**

18       **Summary**

19       Deregister to receive notifications for the specified events.

1 Format

C

```
2 typedef pmix_status_t (*pmix_server_deregister_events_fn_t) (
3 pmix_status_t *codes,
4 size_t ncodes,
5 pmix_op_cbfunc_t cbfunc,
6 void *cbdata);
```

C

7 IN codes

8 Array of **pmix\_status\_t** values (array of handles)

9 IN ncodes

10 Number of elements in the *codes* array (integer)

11 IN cbfunc

12 Callback function **pmix\_op\_cbfunc\_t** (function reference)

13 IN cbdata

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

15 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

26 **Description**

27 Deregister to receive notifications for the specified events to which the PMIx server has previously  
28 registered.

29 **Advice to PMIx library implementers**

30 The PMIx server library must track all client registrations. This module function shall only be 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

- 1      • no client (including the server library itself) remains registered for notifications on any included  
2      code - i.e., a code should be included in this call only when no registered notifications against it  
3      remain.
- 

4      **16.3.16 pmix\_server\_notify\_event\_fn\_t**

5      **Summary**

6      Notify the specified processes of an event.

7      **Format**



8      

```
typedef pmix_status_t (*pmix_server_notify_event_fn_t) (
```

9                pmix\_status\_t code,

10               const pmix\_proc\_t \*source,

11               pmix\_data\_range\_t range,

12               pmix\_info\_t info[],

13               size\_t ninfo,

14               pmix\_op\_cbfunc\_t cbfunc,

15               void \*cbdata);



16     **IN code**

17       The `pmix_status_t` event code being referenced structure (handle)

18     **IN source**

19       `pmix_proc_t` of process that generated the event (handle)

20     **IN range**

21       `pmix_data_range_t` range over which the event is to be distributed (handle)

22     **IN info**

23       Optional array of `pmix_info_t` structures containing additional information on the event  
24       (array of handles)

25     **IN ninfo**

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

27     **IN cbfunc**

28       Callback function `pmix_op_cbfunc_t` (function reference)

29     **IN cbdata**

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

31       Returns one of the following:

- 32      • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
33      will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
34      prior to returning from the API.

- 1     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
2        returned *success* - the *cbfunc* will not be called  
  
3     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
4        request, even though the function entry was provided in the server module - the *cbfunc* will not  
5        be called  
  
6     • a PMIx error constant indicating either an error in the input or that the request was immediately  
7        processed and failed - the *cbfunc* will not be called

### Required Attributes

8     PMIx libraries are required to pass any provided attributes to the host environment for processing.

9     Host environments that provide this module entry point are required to support the following  
10      attributes:

#### **PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t)**

12       Define constraints on the processes that can access the provided data. Only processes that  
13        meet the constraints are allowed to access it.

### Description

15     Notify the specified processes (described through a combination of *range* and attributes provided in  
16        the *info* array) of an event generated either by the PMIx server itself or by one of its local clients.  
17     The process generating the event is provided in the *source* parameter, and any further descriptive  
18        information is included in the *info* array.

19     Note that the PMIx server library is not allowed to echo any event given to it by its host via the  
20        **PMIx\_Notify\_event** API back to the host through the  
21        **pmix\_server\_notify\_event\_fn\_t** server module function.

### Advice to PMIx server hosts

22     The callback function is to be executed once the host environment no longer requires that the PMIx  
23        server library maintain the provided data structures. It does not necessarily indicate that the event  
24        has been delivered to any process, nor that the event has been distributed for delivery

## 25    16.3.17 **pmix\_server\_listener\_fn\_t**

### 26    Summary

27     Register a socket the host server can monitor for connection requests.

1   **Format**

C

```
2 typedef pmix_status_t (*pmix_server_listener_fn_t) (
3 int listening_sd,
4 pmix_connection_cbfunc_t cbfunc,
5 void *cbdata);
```

C

6   **IN incoming\_sd**

(integer)

7   **IN cbfunc**

8         Callback function **pmix\_connection\_cbfunc\_t** (function reference)

9   **IN cbdata**

10      (memory reference)

11     Returns **PMIX\_SUCCESS** indicating that the request is accepted, or a negative value corresponding  
12     to a PMIx error constant indicating that the request has been rejected.

13   **Description**

14     Register a socket the host environment can monitor for connection requests, harvest them, and then  
15     call the PMIx server library's internal callback function for further processing. A listener thread is  
16     essential to efficiently harvesting connection requests from large numbers of local clients such as  
17     occur when running on large SMPs. The host server listener is required to call accept on the  
18     incoming connection request, and then pass the resulting socket to the provided cbfunc. A **NULL**  
19     for this function will cause the internal PMIx server to spawn its own listener thread.

20   **16.3.17.1 PMIx Client Connection Callback Function**

21   **Summary**

22     Callback function for incoming connection request from a local client.

23   **Format**

C

```
24 PMIx v1.0 typedef void (*pmix_connection_cbfunc_t) (
25 int incoming_sd, void *cbdata);
```

C

26   **IN incoming\_sd**

(integer)

27   **IN cbdata**

28      (memory reference)

29   **Description**

30     Callback function for incoming connection requests from local clients - only used by host  
31     environments that wish to directly handle socket connection requests.

## 16.3.18 pmix\_server\_query\_fn\_t

### 2 Summary

3 Query information from the resource manager.

### 4 Format

```
5 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);
```

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

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
2 In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
4     Effective user ID of the connecting process.  
5 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
6     Effective group ID of the connecting process.

## Optional Attributes

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

8 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (**char\***)  
9     Request a comma-delimited list of active namespaces. NO QUALIFIERS.  
10 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (**pmix\_status\_t**)  
11     Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX\_NSPACE**  
12       indicating the namespace whose status is being queried.  
13 **PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (**char\***)  
14     Request a comma-delimited list of scheduler queues. NO QUALIFIERS.  
15 **PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (**char\***)  
16     Returns status of a specified scheduler queue, expressed as a string. OPTIONAL  
17       QUALIFIERS: **PMIX\_ALLOC\_QUEUE** naming specific queue whose status is being  
18       requested.  
19 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.ptable" (**char\***)  
20     Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
21       process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
22       **PMIX\_NSPACE** indicating the namespace whose process table is being queried.  
23 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lptable" (**char\***)  
24     Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
25       process in the specified namespace executing on the same node as the requester, ordered by  
26       process job rank. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the namespace  
27       whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX\_HOSTNAME**  
28       indicating the host whose local process table is being queried. By default, the query assumes  
29       that the host upon which the request was made is to be used.  
30 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmix.qry.spawn" (**bool**)  
31     Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.  
32 **PMIX\_QUERY\_DEBUG\_SUPPORT** "pmix.qry.debug" (**bool**)  
33     Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

```
1 PMIX_QUERY_MEMORY_USAGE "pmixqry.mem" (bool)
2 Return information on memory usage for the processes indicated in the qualifiers.
3 OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
4 specific process(es) whose memory usage is being requested.
5 PMIX_QUERY_LOCAL_ONLY "pmixqry.local" (bool)
6 Constrain the query to local information only. NO QUALIFIERS.
7 PMIX_QUERY_REPORT_AVG "pmixqry.avg" (bool)
8 Report only average values for sampled information. NO QUALIFIERS.
9 PMIX_QUERY_REPORT_MINMAX "pmixqry.minmax" (bool)
10 Report minimum and maximum values. NO QUALIFIERS.
11 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
12 String identifier of the allocation whose status is being requested. NO QUALIFIERS.
13 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
14 Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
15 OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
16 requested (defaults to allocation containing the caller).
```



## 17 **Description**

18 Query information from the host environment. The query will include the namespace/rank of the  
19 process that is requesting the info, an array of **pmix\_query\_t** describing the request, and a  
20 callback function/data for the return.

### Advice to PMIx library implementers

21 The PMIx server library should not block in this function as the host environment may, depending  
22 upon the information being requested, require significant time to respond.  


## 23 **16.3.19 pmix\_server\_tool\_connection\_fn\_t**

### 24 **Summary**

25 Register that a tool has connected to the server.

## Format

```
typedef void (*pmix_server_tool_connection_fn_t)(
 pmix_info_t info[], size_t ninfo,
 pmix_tool_connection_cbfunc_t cbfunc,
 void *cbdata);
```

|           |               |                                                                    |
|-----------|---------------|--------------------------------------------------------------------|
| <b>IN</b> | <b>info</b>   | Array of <code>pmix_info_t</code> structures (array of handles)    |
| <b>IN</b> | <b>ninfo</b>  | Number of elements in the <i>info</i> array (integer)              |
| <b>IN</b> | <b>cbfunc</b> | Callback function <code>pmix_tool_connection_cbfunc_t</code> (func |
| <b>IN</b> | <b>cbdata</b> | 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\_NSPACE** "pmix.tool.nspace" (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

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

2   **PMIX\_FWD\_STDOUT "pmix.fwd.stdout" (bool)**

3   Requests that the ability to forward the **stdout** of the spawned processes be maintained.  
4   The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
5   other options for delivery of the forwarded output.

6   **PMIX\_FWD\_STDERR "pmix.fwd.stderr" (bool)**

7   Requests that the ability to forward the **stderr** of the spawned processes be maintained.  
8   The requester will issue a call to **PMIx\_IOF\_pull** to specify the callback function and  
9   other options for delivery of the forwarded output.

10   **PMIX\_FWD\_STDIN "pmix.fwd.stdin" (pmix\_rank\_t)**

11   The requester intends to push information from its **stdin** to the indicated process. The  
12   local spawn agent should, therefore, ensure that the **stdin** channel to that process remains  
13   available. A rank of **PMIX\_RANK\_WILDCARD** indicates that all processes in the spawned  
14   job are potential recipients. The requester will issue a call to **PMIx\_IOF\_push** to initiate  
15   the actual forwarding of information to specified targets - this attribute simply requests that  
16   the IL retain the ability to forward the information to the designated targets.

17   **PMIX\_VERSION\_INFO "pmix.version" (char\*)**

18   PMIx version of the library being used by the connecting process.

## 19   Description

20   Register that a tool has connected to the server, possibly requesting that the tool be assigned a  
21   namespace/rank identifier for further interactions. The **pmix\_info\_t** array is used to pass  
22   qualifiers for the connection request, including the effective uid and gid of the calling tool for  
23   authentication purposes.

24   If the tool already has an assigned process identifier, then this must be indicated in the *info* array.  
25   The host is responsible for checking that the provided namespace does not conflict with any  
26   currently known assignments, returning an appropriate error in the callback function if a conflict is  
27   found.

28   The host environment is solely responsible for authenticating and authorizing the connection using  
29   whatever means it deems appropriate. If certificates or other authentication information are  
30   required, then the tool must provide them. The conclusion of those operations shall be  
31   communicated back to the PMIx server library via the callback function.

32   Approval or rejection of the connection request shall be returned in the *status* parameter of the  
33   **pmix\_tool\_connection\_cbfunc\_t**. If the connection is refused, the PMIx server library  
34   must terminate the connection attempt. The host must not execute the callback function prior to  
35   returning from the API.

### 16.3.19.1 Tool connection attributes

Attributes associated with tool connections.

```
PMIX_USERID "pmix.euid" (uint32_t)
 Effective user ID of the connecting process.
PMIX_GRPID "pmix.egid" (uint32_t)
 Effective group ID of the connecting process.
PMIX_VERSION_INFO "pmix.version" (char*)
 PMIx version of the library being used by the connecting process.
```

### 16.3.19.2 PMIx Tool Connection Callback Function

#### Summary

Callback function for incoming tool connections.

#### Format

```
typedef void (*pmix_tool_connection_cbfunc_t) (
 pmix_status_t status,
 pmix_proc_t *proc, void *cbdata);
```

IN status

pmix\_status\_t value (handle)

IN proc

pmix\_proc\_t structure containing the identifier assigned to the tool (handle)

IN cbdata

Data to be passed (memory reference)

#### Description

Callback function for incoming tool connections. The host environment shall provide a namespace/rank identifier for the connecting tool.

#### Advice to PMIx server hosts

It is assumed that **rank=0** will be the normal assignment, but allow for the future possibility of a parallel set of tools connecting, and thus each process requiring a unique rank.

### 16.3.20 pmix\_server\_log\_fn\_t

#### Summary

Log data on behalf of a client.

1 Format

C

```
2 typedef void (*pmix_server_log_fn_t)(
3 const pmix_proc_t *client,
4 const pmix_info_t data[], size_t ndata,
5 const pmix_info_t directives[], size_t ndirs,
6 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

7 **IN client**  
8      **pmix\_proc\_t** structure (handle)  
9 **IN data**  
10     Array of info structures (array of handles)  
11 **IN ndata**  
12     Number of elements in the *data* 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 **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)

▼----- Required Attributes -----▼

21 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
22 In addition, the following attributes are required to be included in the passed *info* array:

23 **PMIX\_USERID "pmix.euid" (uint32\_t)**  
24      Effective user ID of the connecting process.  
25 **PMIX\_GRPID "pmix.egid" (uint32\_t)**  
26      Effective group ID of the connecting process.

27  
28 Host environments that provide this module entry point are required to support the following  
29 attributes:

30 **PMIX\_LOG\_STDERR "pmix.log.stderr" (char\*)**  
31      Log string to **stderr**.  
32 **PMIX\_LOG\_STDOUT "pmix.log.stdout" (char\*)**  
33      Log string to **stdout**.  
34 **PMIX\_LOG\_SYSLOG "pmix.log.syslog" (char\*)**

1 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
2 otherwise to local syslog.

### Optional Attributes

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

4 **PMIX\_LOG\_MSG** "pmix.log.msg" (**pmix\_byte\_object\_t**)

5 Message blob to be sent somewhere.

6 **PMIX\_LOG\_EMAIL** "pmix.log.email" (**pmix\_data\_array\_t**)

7 Log via email based on **pmix\_info\_t** containing directives.

8 **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (**char\***)

9 Comma-delimited list of email addresses that are to receive the message.

10 **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (**char\***)

11 Subject line for email.

12 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (**char\***)

13 Message to be included in email.

### Description

15 Log data on behalf of a client. This function is not intended for output of computational results, but  
16 rather for reporting status and error messages. The host must not execute the callback function prior  
17 to returning from the API.

## 16.3.21 **pmix\_server\_alloc\_fn\_t**

### Summary

Request allocation operations on behalf of a client.

## 1 Format

C

```

2 typedef pmix_status_t (*pmix_server_alloc_fn_t)(
3 const pmix_proc_t *client,
4 pmix_alloc_directive_t directive,
5 const pmix_info_t data[],
6 size_t ndata,
7 pmix_info_cbfunc_t cbfunc,
8 void *cbdata);
```

**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)

21 Returns one of the following:

- 22 • PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result  
 23 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
 24 prior to returning from the API.
- 25 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and  
 26 returned *success* - the *cbfunc* will not be called
- 27 • PMIX\_ERR\_NOT\_SUPPORTED, indicating that the host environment does not support the  
 28 request, even though the function entry was provided in the server module - the *cbfunc* will not  
 29 be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
 31 processed and failed - the *cbfunc* will not be called


 Required Attributes

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
 33 In addition, the following attributes are required to be included in the passed *info* array:

34 **PMIX\_USERID "pmix.euid" (uint32\_t)**  
 35       Effective user ID of the connecting process.

```
1 PMIX_GRP_ID "pmix.egid" (uint32_t)
2 Effective group ID of the connecting process.
3
4 Host environments that provide this module entry point are required to support the following
5 attributes:
6 PMIX_ALLOC_ID "pmix.alloc.id" (char*)
7 A string identifier (provided by the host environment) for the resulting allocation which can
8 later be used to reference the allocated resources in, for example, a call to PMIX_Spawn.
9 PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
10 The number of nodes being requested in an allocation request.
11 PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
12 Number of PUs being requested in an allocation request.
13 PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
14 Total session time (in seconds) being requested in an allocation request.
```

## Optional Attributes

```
15 The following attributes are optional for host environments that support this operation:
16 PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
17 Regular expression of the specific nodes being requested in an allocation request.
18 PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
19 Regular expression of the number of PUs for each node being requested in an allocation
20 request.
21 PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
22 Regular expression of the specific PUs being requested in an allocation request.
23 PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
24 Number of Megabytes[base2] of memory (per process) being requested in an allocation
25 request.
26 PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
27 Array of pmix_info_t describing requested fabric resources. This must include at least:
28 PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
29 PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
30 PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
```

1       The key to be used when accessing this requested fabric allocation. The fabric allocation  
2       will be returned/stored as a `pmix_data_array_t` of `pmix_info_t` whose first  
3       element is composed of this key and the allocated resource description. The type of the  
4       included value depends upon the fabric support. For example, a TCP allocation might  
5       consist of a comma-delimited string of socket ranges such as `"32000-32100,`  
6       `33005, 38123-38146"`. Additional array entries will consist of any provided resource  
7       request directives, along with their assigned values. Examples include:  
8       `PMIX_ALLOC_FABRIC_TYPE` - the type of resources provided;  
9       `PMIX_ALLOC_FABRIC_PLANE` - if applicable, what plane the resources were assigned  
10      from; `PMIX_ALLOC_FABRIC_QOS` - the assigned QoS; `PMIX_ALLOC_BANDWIDTH` -  
11      the allocated bandwidth; `PMIX_ALLOC_FABRIC_SEC_KEY` - a security key for the  
12      requested fabric allocation. NOTE: the array contents may differ from those requested,  
13      especially if `PMIX_INFO_REQD` was not set in the request.

14      `PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)`  
15      Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
16      request.  
17      `PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)`  
18      Fabric quality of service level for the job being requested in an allocation request.



## 19     **Description**

20     Request new allocation or modifications to an existing allocation on behalf of a client. Several  
21     broad categories are envisioned, including the ability to:

- 22
  - Request allocation of additional resources, including memory, bandwidth, and compute for an  
23       existing allocation. Any additional allocated resources will be considered as part of the current  
24       allocation, and thus will be released at the same time.
  - 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.
  - Extend the reservation on currently allocated resources, subject to scheduling availability and  
29       priorities.
  - Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to  
31       the scheduler with a promise to return them upon subsequent request.

32     The callback function provides a *status* to indicate whether or not the request was granted, and to  
33     provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of  
34     `pmix_info_t` structures.

### 35    **16.3.22 pmix\_server\_job\_control\_fn\_t**

#### 36    **Summary**

37    Execute a job control action on behalf of a client.

1 Format

C

```
2 typedef pmix_status_t (*pmix_server_job_control_fn_t)(
3 const pmix_proc_t *requestor,
4 const pmix_proc_t targets[],
5 size_t ntargs,
6 const pmix_info_t directives[],
7 size_t ndirs,
8 pmix_info_cbfunc_t cbfunc,
9 void *cbdata);
```

C

10 **IN requestor**  
11     **pmix\_proc\_t** structure of requesting process (handle)  
12 **IN targets**  
13     Array of proc structures (array of handles)  
14 **IN ntargs**  
15     Number of elements in the *targets* array (integer)  
16 **IN directives**  
17     Array of info structures (array of handles)  
18 **IN ndirs**  
19     Number of elements in the *info* array (integer)  
20 **IN cbfunc**  
21     Callback function **pmix\_info\_cbfunc\_t** (function reference)  
22 **IN cbdata**  
23     Data to be passed to the callback function (memory reference)

24 Returns one of the following:

- 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.
  - **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
28       returned *success* - the *cbfunc* will not be called
  - **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
29       request, even though the function entry was provided in the server module - the *cbfunc* will not  
30       be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately  
31       processed and failed - the *cbfunc* will not be called

## Required Attributes

1 PMIx libraries are required to pass any attributes provided by the client to the host environment for  
2 processing. In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
4       Effective user ID of the connecting process.  
5 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
6       Effective group ID of the connecting process.

---

7  
8 Host environments that provide this module entry point are required to support the following  
9 attributes:

10 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (**char\***)  
11      Provide a string identifier for this request. The user can provide an identifier for the  
12      requested operation, thus allowing them to later request status of the operation or to  
13      terminate it. The host, therefore, shall track it with the request for future reference.  
14 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (**bool**)  
15      Pause the specified processes.  
16 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (**bool**)  
17      Resume ("un-pause") the specified processes.  
18 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (**bool**)  
19      Forcibly terminate the specified processes and cleanup.  
20 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (**int**)  
21      Send given signal to specified processes.  
22 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (**bool**)  
23      Politely terminate the specified processes.

## Optional Attributes

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

25 **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (**char\***)  
26      Cancel the specified request - the provided request ID must match the  
27      **PMIX\_JOB\_CTRL\_ID** provided to a previous call to **PMIx\_Job\_control**. An ID of  
28      **NULL** implies cancel all requests from this requestor.  
29 **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (**char\***)  
30      Restart the specified processes using the given checkpoint ID.  
31 **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (**char\***)  
32      Checkpoint the specified processes and assign the given ID to it.

```
1 PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
2 Use event notification to trigger a process checkpoint.
3 PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
4 Use the given signal to trigger a process checkpoint.
5 PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
6 Time in seconds to wait for a checkpoint to complete.
7 PMIX_JOB_CTRL_CHECKPOINT_METHOD
8 "pmix.jctrl.ckmethod" (pmix_data_array_t)
9 Array of pmix_info_t declaring each method and value supported by this application.
10 PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
11 Regular expression identifying nodes that are to be provisioned.
12 PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
13 Name of the image that is to be provisioned.
14 PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
15 Indicate that the job can be pre-empted.
```

## 16 Description

17 Execute a job control action on behalf of a client. The *targets* array identifies the processes to  
18 which the requested job control action is to be applied. A **NULL** value can be used to indicate all  
19 processes in the caller's namespace. The use of **PMIX\_RANK\_WILDCARD** can also be used to  
20 indicate that all processes in the given namespace are to be included.

21 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The callback  
22 function provides a *status* to indicate whether or not the request was granted, and to provide some  
23 information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
24 **pmix\_info\_t** structures.

## 25 16.3.23 pmix\_server\_monitor\_fn\_t

### 26 Summary

27 Request that a client be monitored for activity.

## 1 Format

C

```
2 typedef pmix_status_t (*pmix_server_monitor_fn_t)(
3 const pmix_proc_t *requestor,
4 const pmix_info_t *monitor,
5 pmix_status_t error,
6 const pmix_info_t directives[],
7 size_t ndirs,
8 pmix_info_cbfunc_t cbfunc,
9 void *cbdata);
```

C

10 **IN requestor**11 **pmix\_proc\_t** structure of requesting process (handle)12 **IN monitor**13 **pmix\_info\_t** identifying the type of monitor being requested (handle)14 **IN error**

15 Status code to use in generating event if alarm triggers (integer)

16 **IN directives**

17 Array of info structures (array of handles)

18 **IN ndirs**19 Number of elements in the *info* array (integer)20 **IN cbfunc**21 Callback function **pmix\_info\_cbfunc\_t** (function reference)22 **IN cbdata**

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

24 Returns one of the following:

- 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

35 This entry point is only called for monitoring requests that are not directly supported by the PMIx  
36 server library itself.

## Required Attributes

If supported by the PMIx server library, then the library must not pass any supported attributes to the host environment. Any attributes provided by the client that are not directly supported by the server library must be passed to the host environment if it provides this module entry. In addition, the following attributes are required to be included in the passed *info* array:

5     **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
6         Effective user ID of the connecting process.  
  
7     **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
8         Effective group ID of the connecting process.

9     Host environments are not required to support any specific monitoring attributes.

## Optional Attributes

10   The following attributes may be implemented by a host environment.

11   **PMIX\_MONITOR\_ID** "pmix.monitor.id" (**char\***)  
12      Provide a string identifier for this request.  
  
13   **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (**char\***)  
14      Identifier to be canceled (**NULL** means cancel all monitoring for this process).  
  
15   **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (**bool**)  
16      The application desires to control the response to a monitoring event - i.e., the application is  
17      requesting that the host environment not take immediate action in response to the event (e.g.,  
18      terminating the job).  
  
19   **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (**void**)  
20      Register to have the PMIx server monitor the requestor for heartbeats.  
  
21   **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (**uint32\_t**)  
22      Time in seconds before declaring heartbeat missed.  
  
23   **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (**uint32\_t**)  
24      Number of heartbeats that can be missed before generating the event.  
  
25   **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (**char\***)  
26      Register to monitor file for signs of life.  
  
27   **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (**bool**)  
28      Monitor size of given file is growing to determine if the application is running.  
  
29   **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (**char\***)  
30      Monitor time since last access of given file to determine if the application is running.  
  
31   **PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (**char\***)  
32      Monitor time since last modified of given file to determine if the application is running.

```
1 PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
2 Time in seconds between checking the file.
3 PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
4 Number of file checks that can be missed before generating the event.
```

## 5 Description

6 Request that a client be monitored for activity.

### 7 16.3.24 pmix\_server\_get\_cred\_fn\_t

#### 8 Summary

9 Request a credential from the host environment.

#### 10 Format

```
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:

```
1 PMIX_USERID "pmix.euid" (uint32_t)
2 Effective user ID of the connecting process.
3
4 PMIX_GRP_ID "pmix.egid" (uint32_t)
5 Effective group ID of the connecting process.
```

## Optional Attributes

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

```
6 PMIX_CRED_TYPE "pmix.sec.ctype" (char*)
7 When passed in PMIx_Get_credential, a prioritized, comma-delimited list of desired
8 credential types for use in environments where multiple authentication mechanisms may be
9 available. When returned in a callback function, a string identifier of the credential type.
10
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.
```

### 17 **Description**

18 Request a credential from the host environment.

## 19 **16.3.24.1 Credential callback function**

### 20 **Summary**

21 Callback function to return a requested security credential

## Format

```
typedef void (*pmix_credential_cbfunc_t)(
 pmix_status_t status,
 pmix_byte_object_t *credential,
 pmix_info_t info[], size_t ninfo,
 void *cbdata);
```

|           |                           |                                                                                                                                             |
|-----------|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IN</b> | <b>status</b>             |                                                                                                                                             |
|           | <b>pmix_status_t</b>      | value (handle)                                                                                                                              |
| <b>IN</b> | <b>credential</b>         |                                                                                                                                             |
|           | <b>pmix_byte_object_t</b> | structure containing the security credential (handle)                                                                                       |
| <b>IN</b> | <b>info</b>               | Array of provided by the system to pass any additional information about the credential - e.g., the identity of the issuing agent. (handle) |
| <b>IN</b> | <b>ninfo</b>              | Number of elements in <i>info</i> ( <b>size_t</b> )                                                                                         |
| <b>IN</b> | <b>cbdata</b>             | 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.

**16.3.25 pmix server validate cred fn t**

## Summary

Request validation of a credential.

## 1 Format

C

```
2 typedef pmix_status_t (*pmix_server_validate_cred_fn_t)(
3 const pmix_proc_t *proc,
4 const pmix_byte_object_t *cred,
5 const pmix_info_t directives[],
6 size_t ndirs,
7 pmix_validation_cbfunc_t cbfunc,
8 void *cbdata);
```

9     **IN proc**  
10    **pmix\_proc\_t** structure of requesting process (handle)  
11     **IN cred**  
12    Pointer to **pmix\_byte\_object\_t** containing the credential (handle)  
13     **IN directives**  
14    Array of info structures (array of handles)  
15     **IN ndirs**  
16    Number of elements in the *info* array (integer)  
17     **IN cbfunc**  
18    Callback function to return the result (**pmix\_validation\_cbfunc\_t** function reference)  
19     **IN cbdata**  
20    Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*
- 24     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
25       returned *success* - the *cbfunc* will not be called
- 26     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
27       request, even though the function entry was provided in the server module - the *cbfunc* will not  
28       be called
- 29     • a PMIx error constant indicating either an error in the input or that the request was immediately  
30       processed and failed - the *cbfunc* will not be called

31                      Required Attributes

32 If the PMIx library does not itself validate the credential, then it is required to pass any attributes  
33 provided by the client to the host environment for processing. In addition, it must include the  
following attributes in the passed *info* array:

34     **PMIX\_USERID "pmix.euid" (uint32\_t)**  
35       Effective user ID of the connecting process.

1       **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2              Effective group ID of the connecting process.

---

3  
4       Host environments are not required to support any specific attributes.

▲-----▼ **Optional Attributes** ▼-----▲

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

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

10      **Description**

11      Request validation of a credential obtained from the host environment via a prior call to the  
12      **pmix\_server\_get\_cred\_fn\_t** module entry.

13     **16.3.26 Credential validation callback function**

14      **Summary**

15      Callback function for security credential validation.

1      **Format**

2      `typedef void (*pmix_validation_cbfunc_t) (`  
3                `pmix_status_t status,`  
4                `pmix_info_t info[], size_t ninfo,`  
5                `void *cbdata);`

6      **IN    status**

7                `pmix_status_t` value (handle)

8      **IN    info**

9                Array of `pmix_info_t` provided by the system to pass any additional information about the  
10          authentication - e.g., the effective userid and group id of the certificate holder, and any related  
11          authorizations (handle)

12     **IN    ninfo**

13          Number of elements in *info* (`size_t`)

14     **IN    cbdata**

15          Object passed in original request (memory reference)

16          The returned status shall be one of the following:

- 17        • `PMIX_SUCCESS`, indicating that the request was processed and returned *success* (i.e., the  
18          credential was both valid and any information it contained was successfully processed). Details  
19          of the result will be returned in the *info* array
- 20        • a PMIx error constant indicating either an error in the parsing of the credential or that the request  
21          was refused

22      **Description**

23      Define a validation callback function to indicate if a provided credential is valid, and any  
24      corresponding information regarding authorizations and other security matters.

25      **Advice to users**

26      The precise contents of the array will depend on the host environment and its associated security  
27      system. At the minimum, it is expected (but not required) that the array will contain entries for the  
28      `PMIX_USERID` and `PMIX_GRPID` of the client described in the credential. The *info* array is  
29      owned by the PMIx library and is not to be released or altered by the receiving party.

29      **16.3.27 pmix\_server\_iof\_fn\_t**

30      **Summary**

31      Request the specified IO channels be forwarded from the given array of processes.

## 1 Format

C

```

2 typedef pmix_status_t (*pmix_server_iof_fn_t)(const pmix_proc_t procs[],
3 size_t nprocs,
4 const pmix_info_t directives[],
5 size_t ndirs,
6 pmix_iof_channel_t channels,
7 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

9 IN **procs**

10 Array **pmix\_proc\_t** identifiers whose IO is being requested (handle)

11 IN **nprocs**

12 Number of elements in *procs* (**size\_t**)

13 IN **directives**

14 Array of **pmix\_info\_t** structures further defining the request (array of handles)

15 IN **ndirs**

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

17 IN **channels**

18 Bitmask identifying the channels to be forwarded (**pmix\_iof\_channel\_t**)

19 IN **cbfunc**

20 Callback function **pmix\_op\_cbfunc\_t** (function reference)

21 IN **cbdata**

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

23 Returns one of the following:

- 24 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
25 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
26 function prior to returning from the API.
- 27 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
28 returned *success* - the *cbfunc* will not be called
- 29 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
30 request, even though the function entry was provided in the server module - the *cbfunc* will not  
31 be called
- 32 • a PMIx error constant indicating either an error in the input or that the request was immediately  
33 processed and failed - the *cbfunc* will not be called


**Required Attributes**


34 The following attributes are required to be included in the passed *info* array:

35 **PMIX\_USERID "pmix.euid" (uint32\_t)**

1                   Effective user ID of the connecting process.  
2   **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
3                   Effective group ID of the connecting process.

---

5 Host environments that provide this module entry point are required to support the following  
6 attributes:

7   **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.cszie" (**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.

## 8 16.3.27.1 IOF delivery function

## Summary

Callback function for delivering forwarded IO to a process.

## Format

C

```
typedef void (*pmix_ifc_cfunc_t) (
 size_t iofhdlr, pmix_ifc_channel_t channel,
 pmix_proc_t *source, char *payload,
 pmix_info_t info[], size_t ninfo);
```

IN jofhd1r

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_IOC_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.

7 16.3.28 pmix server stdin fn t

## Summary

Pass standard input data to the host environment for transmission to specified recipients.

$^{10} \text{BMJ v3.0}$

```
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);
```

**IN** **source**  
**pmix proc t** structure of source process (handle)

**IN** **targets**  
Array of **pmix\_proc\_t** target identifiers (handle)

**IN** **ntargets**  
Number of targets to be tested.

**IN** Number of elements in the *targets* array (in **directives**)

Array of info structures (array of handles)

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

IN bo Number of elements in the *arr* array (integer)

**IN** Pointer to `pmix_byte_object_t` containing the payload (`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.

### 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.

## 16.3.29 pmix\_server\_grp\_fn\_t

### Summary

23 Request group operations (construct, destruct, etc.) on behalf of a set of processes.

## 1 Format

C

```
2 typedef pmix_status_t (*pmix_server_grp_fn_t)(
3 pmix_group_operation_t op,
4 char grp[],
5 const pmix_proc_t procs[],
6 size_t nprocs,
7 const pmix_info_t directives[],
8 size_t ndirs,
9 pmix_info_cbfunc_t cbfunc,
10 void *cbdata);
```

C

11 **IN op**12 **pmix\_group\_operation\_t** value indicating operation the host is requested to perform  
13 (integer)14 **IN grp**

15 Character string identifying the group (string)

16 **IN procs**17 Array of **pmix\_proc\_t** identifiers of participants (handle)18 **IN nprocs**19 Number of elements in the *procs* array (integer)20 **IN directives**

21 Array of info structures (array of handles)

22 **IN ndirs**23 Number of elements in the *info* array (integer)24 **IN cbfunc**25 Callback function **pmix\_info\_cbfunc\_t** (function reference)26 **IN cbdata**

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

28 Returns one of the following:

- 29 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
30 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
31 function prior to returning from the API.
- 32 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
33 returned *success* - the *cbfunc* will not be called
- 34 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
35 request, even though the function entry was provided in the server module - the *cbfunc* will not  
36 be called
- 37 • a PMIx error constant indicating either an error in the input or that the request was immediately  
38 processed and failed - the *cbfunc* will not be called

## Optional Attributes

1 The following attributes may be supported by a host environment.

2 **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

3 Requests that the RM assign a new context identifier to the newly created group. The  
4 identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
5 specified in the request. Thus, the value serves as a means of identifying the group within  
6 that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

7 **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

8 Group operation only involves local processes. PMIx implementations are *required* to  
9 automatically scan an array of group members for local vs remote processes - if only local  
10 processes are detected, the implementation need not execute a global collective for the  
11 operation unless a context ID has been requested from the host environment. This can result  
12 in significant time savings. This attribute can be used to optimize the operation by indicating  
13 whether or not only local processes are represented, thus allowing the implementation to  
14 bypass the scan.

15 **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

16 Data collected during group construction to ensure communication between group members  
17 is supported upon completion of the operation.

18 **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

19 Participation is optional - do not return an error if any of the specified processes terminate  
20 without having joined. The default is **false**.

21 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

22 Define constraints on the processes that can access the provided data. Only processes that  
23 meet the constraints are allowed to access it.

24 The following attributes may be included in the host's response:

25 **PMIX\_GROUP\_ID** "pmix.grp.id" (char\*)

26 User-provided group identifier - as the group identifier may be used in PMIx operations, the  
27 user is required to ensure that the provided ID is unique within the scope of the host  
28 environment (e.g., by including some user-specific or application-specific prefix or suffix to  
29 the string).

30 **PMIX\_GROUP\_MEMBERSHIP** "pmix.grp.mbrs" (pmix\_data\_array\_t\*)

31 Array **pmix\_proc\_t** identifiers identifying the members of the specified group.

32 **PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (size\_t)

33 Context identifier assigned to the group by the host RM.

34 **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

35 Data collected during group construction to ensure communication between group members  
36 is supported upon completion of the operation.

## 1 Description

2 Perform the specified operation across the identified processes, plus any special actions included in  
3 the directives. Return the result of any special action requests in the callback function when the  
4 operation is completed. Actions may include a request (**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID**)  
5 that the host assign a unique numerical (size\_t) ID to this group - if given, the **PMIX\_RANGE**  
6 attribute will specify the range across which the ID must be unique (default to  
7 **PMIX\_RANGE\_SESSION**).

### 8 16.3.29.1 Group Operation Constants

9 *PMIx v4.0*

10 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.

11 **PMIX\_GROUP\_CONSTRUCT** Construct a group composed of the specified processes - used by  
12 a PMIx server library to direct host operation.

13 **PMIX\_GROUP\_DESTRUCT** Destruct the specified group - used by a PMIx server library to  
14 direct host operation.

### 15 16.3.30 pmix\_server\_fabric\_fn\_t

#### 16 Summary

17 Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

#### 18 Format

```
19 typedef pmix_status_t (*pmix_server_fabric_fn_t)(
20 const pmix_proc_t *requestor,
21 pmix_fabric_operation_t op,
22 const pmix_info_t directives[],
23 size_t ndirs,
24 pmix_info_cfunc_t cfunc,
25 void *cbdata);
```

26 **IN requestor**  
27     **pmix\_proc\_t** identifying the requestor (handle)  
28 **IN op**  
29     **pmix\_fabric\_operation\_t** value indicating operation the host is requested to perform  
30     (integer)  
31 **IN directives**  
32     Array of info structures (array of handles)  
33 **IN ndirs**  
34     Number of elements in the *info* array (integer)

1   **IN cbfunc**  
2    Callback function `pmix_info_cbfunc_t` (function reference)  
3   **IN cldata**  
4    Data to be passed to the callback function (memory reference)  
5   Returns one of the following:  
6    

- `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result  
7      will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
8      function prior to returning from the API.
- `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and  
9      returned *success* - the *cbfunc* will not be called
- `PMIX_ERR_NOT_SUPPORTED`, indicating that the host environment does not support the  
10     request, even though the function entry was provided in the server module - the *cbfunc* will not  
11     be called
- a PMIx error constant indicating either an error in the input or that the request was immediately  
12     processed and failed - the *cbfunc* will not be called

### Required Attributes

16   The following directives are required to be supported by all hosts to aid users in identifying the  
17    fabric and (if applicable) the device to whom the operation references:

18   **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (`string`)  
19     Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.  
20   **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (`string`)  
21     An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).  
22   **PMIX\_FABRIC\_PLANE** "pmix.fab.plane" (`string`)  
23     ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
24     for information, specifies the plane whose information is to be returned. When used directly  
25     as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric  
26     planes in the overall system.  
27   **PMIX\_FABRIC\_DEVICE\_INDEX** "pmix.fabdev.idx" (`uint32_t`)  
28     Index of the device within an associated communication cost matrix.

### Description

29   Perform the specified operation. Return the result of any requests in the callback function when the  
30    operation is completed. Operations may, for example, include a request for fabric information. See  
31    `pmix_fabric_t` for a list of expected information to be included in the response. Note that  
32    requests for device index are to be returned in the callback function's array of `pmix_info_t`  
33    using the `PMIX_FABRIC_DEVICE_INDEX` attribute.  
34

## CHAPTER 17

# Tools and Debuggers

---

The term *tool* widely refers to programs executed by the user or system administrator on a command line. Tools frequently interact with either the SMS, user applications, or both to perform administrative and support functions. For example, a debugger tool might be used to remotely control the processes of a parallel application, monitoring their behavior on a step-by-step basis. Historically, such tools were custom-written for each specific host environment due to the customized and/or proprietary nature of the environment's interfaces.

The advent of PMIx offers the possibility for creating portable tools capable of interacting with multiple RMs without modification. Possible use-cases include:

- querying the status of scheduling queues and estimated allocation time for various resource options
- job submission and allocation requests
- querying job status for executing applications
- launching, monitoring, and debugging applications

Enabling these capabilities requires some extensions to the PMIx Standard (both in terms of APIs and attributes), and utilization of client-side APIs for more tool-oriented purposes.

This chapter defines specific APIs related to tools, provides tool developers with an overview of the support provided by PMIx, and serves to guide RM vendors regarding roles and responsibilities of RMs to support tools. As the number of tool-specific APIs and attributes is fairly small, the bulk of the chapter serves to provide a "theory of operation" for tools and debuggers. Description of the APIs themselves is therefore deferred to the Section 17.5 later in the chapter.

## 17.1 Connection Mechanisms

The key to supporting tools lies in providing mechanisms by which a tool can connect to a PMIx server. Application processes are able to connect because their local RM daemon provides them with the necessary contact information upon execution. A command-line tool, however, isn't spawned by an RM daemon, and therefore lacks the information required for rendezvous with a PMIx server.

Once a tool has started, it initializes PMIx as a tool (via `PMIx_tool_init`) if its access is restricted to PMIx-based informational services such as `PMIx_Query_info`. However, if the

1 tool intends to start jobs, then it must include the **PMIX\_LAUNCHER** attribute to inform the library  
2 of that intent so that the library can initialize and provide access to the corresponding support.

3 Support for tools requires that the PMIx server be initialized with an appropriate attribute  
4 indicating that tool connections are to be allowed. Separate attributes are provided to "fine-tune"  
5 this permission by allowing the environment to independently enable (or disable) connections from  
6 tools executing on nodes other than the one hosting the server itself. The PMIx server library shall  
7 provide an opportunity for the host environment to authenticate and approve each connection  
8 request from a specific tool by calling the **pmix\_server\_tool\_connection\_fn\_t** "hook"  
9 provided in the server module for that purpose. Servers in environments that do not provide this  
10 "hook" shall automatically reject all tool connection requests.

11 Tools can connect to any local or remote PMIx server provided they are either explicitly given the required connection information, or are able to discover it via one of several defined rendezvous  
12 protocols. Connection discovery centers around the existence of *rendezvous files* containing the  
13 necessary connection information, as illustrated in Fig. 17.1.  
14

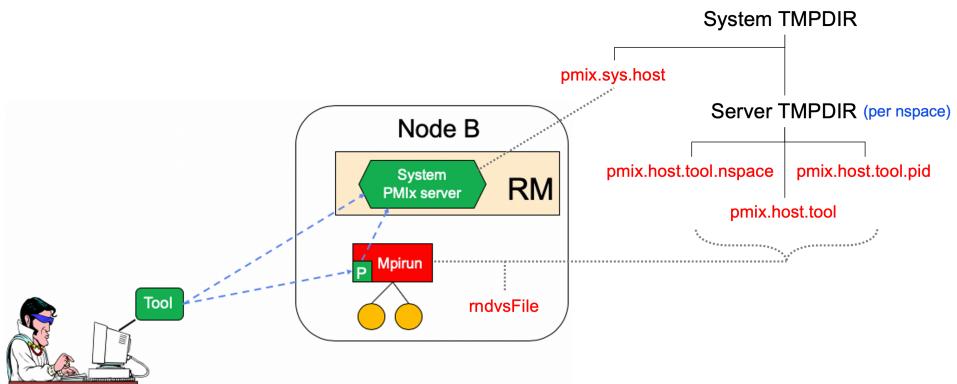


Figure 17.1.: Tool rendezvous files

15 The contents of each rendezvous file are specific to a given PMIx implementation, but should at  
16 least contain the namespace and rank of the server along with its connection URI. Note that tools  
17 linked to one PMIx implementation are therefore unlikely to successfully connect to PMIx server  
18 libraries from another implementation.

19 The top of the directory tree is defined by either the **PMIX\_SYSTEM\_TMPDIR** attribute (if given)  
20 or the **TMPDIR** environmental variable. PMIx servers that are designated as *system servers* by  
21 including the **PMIX\_SERVER\_SYSTEM\_SUPPORT** attribute when calling  
22 **PMIx\_server\_init** will create a rendezvous file in this top-level directory. The filename will  
23 be of the form *pmix.sys.hostname*, where *hostname* is the string returned by the **gethostname**  
24 system call. Note that only one PMIx server on a node can be designated as the system server.

25 Non-system PMIx servers will create a set of three rendezvous files in the directory defined by  
26 either the **PMIX\_SERVER\_TMPDIR** attribute or the **TMPDIR** environmental variable:

- *pmix.host.tool.nspace* where *host* is the string returned by the **gethostname** system call and *nspace* is the namespace of the server.
- *pmix.host.tool.pid* where *host* is the string returned by the **gethostname** system call and *pid* is the PID of the server.
- *pmix.host.tool* where *host* is the string returned by the **gethostname** system call. Note that servers which are not given a namespace-specific **PMIX\_SERVER\_TMPDIR** attribute may not generate this file due to conflicts should multiple servers be present on the node.

The files are identical and may be implemented as symlinks to a single instance. The individual file names are composed so as to aid the search process should a tool wish to connect to a server identified by its namespace or PID.

Servers will additionally provide a rendezvous file in any given location if the path (either absolute or relative) and filename is specified either during **PMIx\_server\_init** using the **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** attribute, or by the **PMIX\_LAUNCHER\_RNDZ\_FILE** environmental variable prior to executing the process containing the server. This latter mechanism may be the preferred mechanism for tools such as debuggers that need to fork/exec a launcher (e.g., "mpiexec") and then rendezvous with it. This is described in more detail in Section 17.2.2.

Rendezvous file ownerships are set to the UID and GID of the server that created them, with permissions set according to the desires of the implementation and/or system administrator policy. All connection attempts are first governed by read access privileges to the target rendezvous file - thus, the combination of permissions, UID, and GID of the rendezvous files act as a first-level of security for tool access.

A tool may connect to as many servers at one time as the implementation supports, but is limited to designating only one such connection as its *primary* server. This is done to avoid confusion when the tool calls an API as to which server should service the request. The first server the tool connects to is automatically designated as the *primary* server.

Tools are allowed to change their primary server at any time via the **PMIx\_tool\_set\_server** API, and to connect/disconnect from a server as many times as desired. Note that standing requests (e.g., event registrations) with the current primary server may be lost and/or may not be transferred when transitioning to another primary server - PMIx implementors are not required to maintain or transfer state across tool-server connections.

Tool process identifiers are assigned by one of the following methods:

- If **PMIX\_TOOL\_NSPACE** is given, then the namespace of the tool will be assigned that value.
  - If **PMIX\_TOOL\_RANK** is also given, then the rank of the tool will be assigned that value.
  - If **PMIX\_TOOL\_RANK** is not given, then the rank will be set to a default value of zero.
- If a process ID is not provided and the tool connects to a server, then one will be assigned by the host environment upon connection to that server.

- 1     • If a process ID is not provided and the tool does not connect to a server (e.g., if  
2       **PMIX\_TOOL\_DO\_NOT\_CONNECT** is given), then the tool shall self-assign a unique identifier.  
3       This is often done using some combination involving hostname and PID.

4     Tool process identifiers remain constant across servers. Thus, it is critical that a system-wide unique  
5       namespace be provided if the tool itself sets the identifier, and that host environments provide a  
6       system-wide unique identifier in the case where the identifier is set by the server upon connection.  
7       The host environment is required to reject any connection request that fails to meet this criterion.

8     For simplicity, the following descriptions will refer to the:

- 9     • **PMIX\_SYSTEM\_TMPDIR** as the directory specified by either the **PMIX\_SYSTEM\_TMPDIR**  
10       attribute (if given) or the **TMPDIR** environmental variable.  
11     • **PMIX\_SERVER\_TMPDIR** as the directory specified by either the **PMIX\_SERVER\_TMPDIR**  
12       attribute or the **TMPDIR** environmental variable.

13     The rendezvous methods are automatically employed for the initial tool connection during  
14       **PMIx\_tool\_init** unless the **PMIX\_TOOL\_DO\_NOT\_CONNECT** attribute is specified, and on  
15       all subsequent calls to **PMIx\_tool\_attach\_to\_server**.

### 16    17.1.1 Rendezvousing with a local server

17     Connection to a local PMIx server is pursued according to the following precedence chain based on  
18       attributes contained in the call to the **PMIx\_tool\_init** or  
19       **PMIx\_tool\_attach\_to\_server** APIs. Servers to which the tool already holds a connection  
20       will be ignored. Except where noted, the PMIx library will return an error if the specified file  
21       cannot be found, the caller lacks permissions to read it, or the server specified within the file does  
22       not respond to or accept the connection — the library will not proceed to check for other  
23       connection options as the user specified a particular one to use.

24     Note that the PMIx implementation may choose to introduce a "delayed connection" protocol  
25       between steps in the precedence chain - i.e., the library may cycle several times, checking for  
26       creation of the rendezvous file each time after a delay of some period of time, thereby allowing the  
27       tool to wait for the server to create the rendezvous file before either returning an error or continuing  
28       to the next step in the chain.

- 29     • If **PMIX\_TOOL\_ATTACHMENT\_FILE** is given, then the tool will attempt to read the specified  
30       file and connect to the server based on the information contained within it. The format of the  
31       attachment file is identical to the rendezvous files described in earlier in this section. An error  
32       will be returned if the specified file cannot be found.
- 33     • If **PMIX\_SERVER\_URI** or **PMIX\_TCP\_URI** is given, then connection will be attempted to the  
34       server at the specified URI. Note that it is an error for both of these attributes to be specified.  
35       **PMIX\_SERVER\_URI** is the preferred method as it is more generalized — **PMIX\_TCP\_URI** is  
36       provided for those cases where the user specifically wants to use a TCP transport for the  
37       connection and wants to error out if one isn't available or cannot be used.

- If **PMIX\_SERVER\_PIDINFO** was provided, then the tool will search for a rendezvous file created by a PMIx server of the given PID in the **PMIX\_SERVER\_TMPDIR** directory. An error will be returned if a matching rendezvous file cannot be found.
- If **PMIX\_SERVER\_NSPACE** is given, then the tool will search for a rendezvous file created by a PMIx server of the given namespace in the **PMIX\_SERVER\_TMPDIR** directory. An error will be returned if a matching rendezvous file cannot be found.
- If **PMIX\_CONNECT\_TO\_SYSTEM** is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the **PMIX\_SYSTEM\_TMPDIR** directory. An error will be returned if a matching rendezvous file cannot be found.
- If **PMIX\_CONNECT\_SYSTEM\_FIRST** is given, then the tool will look for a system-level rendezvous file created by a PMIx server in the **PMIX\_SYSTEM\_TMPDIR** directory. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused — the PMIx library will silently continue to the next option.
- By default, the tool will search the directory tree under the **PMIX\_SERVER\_TMPDIR** directory for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error. No "delayed connection" protocols may be utilized at this point.

Note that there can be multiple local servers - one from the system plus others from launchers and active jobs. The PMIx tool connection search method is not guaranteed to pick a particular server unless directed to do so. Tools can obtain a list of servers available on their local node using the **PMIx\_Query\_info** APIs with the **PMIX\_QUERY\_AVAIL\_SERVERS** key.

### 17.1.2 Connecting to a remote server

Connecting to remote servers is complicated due to the lack of access to the previously-described rendezvous files. Two methods are required to be supported, both based on the caller having explicit knowledge of either connection information or a path to a local file that contains such information:

- If **PMIX\_TOOL\_ATTACHMENT\_FILE** is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it. The format of the attachment file is identical to the rendezvous files described in earlier in this section.
- If **PMIX\_SERVER\_URI** or **PMIX\_TCP\_URI** is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. **PMIX\_SERVER\_URI** is the preferred method as it is more generalized — **PMIX\_TCP\_URI** is provided for those cases where the user specifically wants to use the TCP transport for the connection and wants to error out if it isn't available or cannot be used.

Additional methods may be provided by particular PMIx implementations. For example, the tool may use *ssh* to launch a *probe* process onto the remote node so that the probe can search the **PMIX\_SYSTEM\_TMPDIR** and **PMIX\_SERVER\_TMPDIR** directories for rendezvous files,

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 17.1.3 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 17.1.4 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_NSPACE "pmix.tool.nspace" (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 17.1.5 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_KEEPALIVE_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 17.1.6 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).

```

## 17.2 Launching Applications with Tools

Tool-directed launches require that the tool include the **PMIX\_LAUNCHER** attribute when calling **PMIx\_tool\_init**. Two launch modes are supported:

- *Direct launch* where the tool itself is directly responsible for launching all processes, including debugger daemons, using either the RM or daemons launched by the tool – i.e., there is no *intermediate launcher* (IL) such as *mpiexec*. The case where the tool is self-contained (i.e., uses its own daemons without interacting with an external entity such as the RM) lies outside the scope of this Standard; and
- *Indirect launch* where all processes are started via an IL such as *mpiexec* and the tool itself is not directly involved in launching application processes or debugger daemons. Note that the IL may utilize the RM to launch processes and/or daemons under the tool’s direction.

Either of these methods can be executed interactively or by a batch script. Note that not all host environments may support the direct launch method.

### 17.2.1 Direct launch

In the direct-launch use-case (Fig. 17.2), the tool itself performs the role of the launcher. Once invoked, the tool connects to an appropriate PMIx server - e.g., a system-level server hosted by the

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.

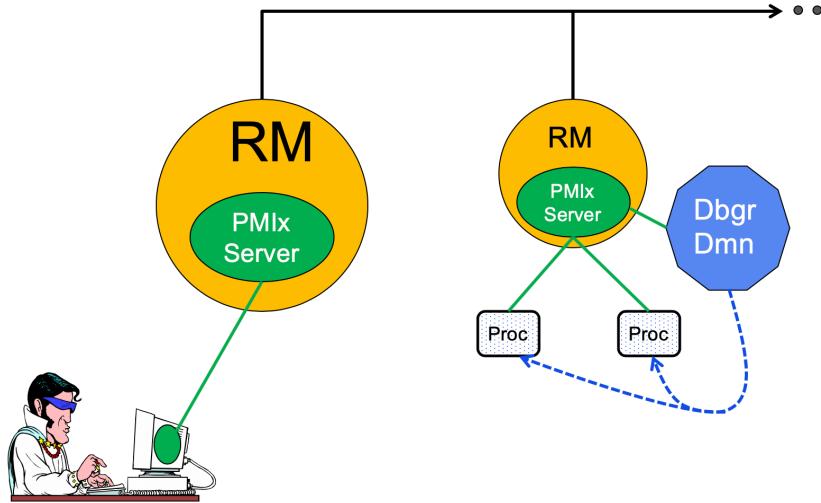


Figure 17.2.: Direct Launch

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 17.4):

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.cszie" (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 12.4.3.
- 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 12.4.3. 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

1 spawned processes until either released by debugger modification of an appropriate  
2 variable or receipt of the **PMIX\_DEBUGGER\_RELEASE** event. The launcher (RM or IL)  
3 is responsible for generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event when  
4 all processes have reached the pause point.

- 5 • **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** "pmix.dbg.notify" (bool)

6 Included in either the **pmix\_info\_t** array in a **pmix\_app\_t** description (if the  
7 directive applies only to that application) or in the **job\_info** array if it applies to all  
8 applications in the given spawn request. Indicates that the specified application is being  
9 spawned under a debugger. The resulting application processes are to notify their server  
10 (by generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event) when they reach  
11 some application-determined location and pause at that point until either released by  
12 debugger modification of an appropriate variable or receipt of the  
13 **PMIX\_DEBUGGER\_RELEASE** event. The launcher (RM or IL) is responsible for  
14 generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event when all processes have  
15 indicated they are at the pause point.

16 The tool then calls the **PMIx\_Spawn** API so that the PMIx library can communicate the spawn  
17 request to the server.

18 Upon receipt, the PMIx server library passes the spawn request to its host RM daemon for  
19 processing via the **pmix\_server\_spawn\_fn\_t** server module function. If this callback was not  
20 provided, then the PMIx server library will return the **PMIX\_ERR\_NOT\_SUPPORTED** error status.

21 If an allocation must be made, then the host environment is responsible for communicating the  
22 request to its associated scheduler. Once resources are available, the host environment initiates the  
23 launch process to start the job. The host environment must parse the spawn request for relevant  
24 directives, returning an error if any required directive cannot be supported. Optional directives may  
25 be ignored if they cannot be supported.

26 Any error while executing the spawn request must be returned by **PMIx\_Spawn** to the requester.  
27 Once the spawn request has succeeded in starting the specified processes, the request will return  
28 **PMIX\_SUCCESS** back to the requester along with the namespace of the started job. Upon  
29 termination of the spawned job, the host environment must generate a **PMIX\_EVENT\_JOB\_END**  
30 event for normal or abnormal termination if requested to do so. The event shall include:

- 31 • the returned status code (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job;  
32 • the identity (**PMIX\_PROCID**) and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if  
33 applicable;  
34 • a **PMIX\_EVENT\_TIMESTAMP** indicating the time the termination occurred; plus  
35 • any other info provided by the host environment.

## 36 17.2.2 Indirect launch

37 In the indirect launch use-case, the application processes are started via an intermediate launcher  
38 (e.g., *mpirexec*) that is itself started by the tool (see Fig 17.3). Thus, at a high level, this is a

1 two-stage launch procedure to start the application: the tool (henceforth referred to as the *initiator*)  
2 starts the IL, which then starts the applications. In practice, additional steps may be involved if, for  
3 example, the IL starts its own daemons to shepherd the application processes.

4 A key aspect of this operational mode is the avoidance of any requirement that the initiator parse  
5 and/or understand the command line of the IL. Instead, the indirect launch procedure supports  
6 either of two methods: one where the initiator assumes responsibility for parsing its command line  
7 to obtain the application as well as the IL and its options, and another where the initiator defers the  
8 command line parsing to the IL. Both of these methods are described in the following sections.

### 9 17.2.2.1 Initiator-based command line parsing

10 This method utilizes a first call to the **PMIx\_Spawn** API to start the IL itself, and then uses a  
11 second call to **PMIx\_Spawn** to request that the IL spawn the actual job. The burden of analyzing  
12 the initial command line to separately identify the IL's command line from the application itself  
13 falls upon the initiator. An example is provided below:

14 \$ initiator --launcher "mpiexec --verbose" -n 3 ./app <appoptions>

15 The initiator spawns the IL using the same procedure for launching an application - it begins by  
16 assembling the description of the IL into a spawn request containing an array of **pmix\_app\_t** and  
17 **pmix\_info\_t** job-level information. Note that this step does not include any information  
18 regarding the application itself - only the launcher is included. In addition, the initiator must  
19 include the rendezvous URI in the environment so the IL knows how to connect back to it.

20 An allocation of resources for the IL itself may or may not be required – if it is, then the allocation  
21 must be made in advance or the spawn request must include allocation request information.

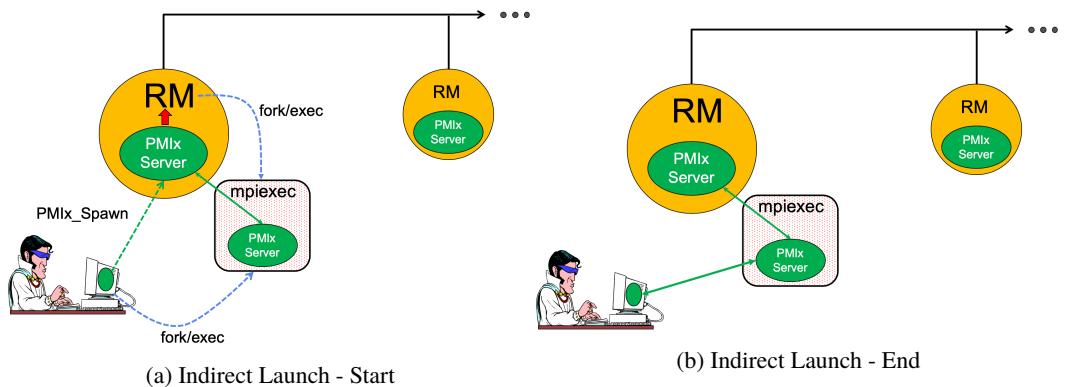


Figure 17.3.: Indirect launch procedure

22 The initiator may optionally wish to include the following tool-specific attributes in the *job\_info*  
23 argument to **PMIx\_Spawn** - note that these attributes refer only to the behavior of the IL itself and  
24 not the eventual job to be launched:

- 25 • **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (**pmix\_rank\_t**)

The requester intends to push information from its `stdin` to the indicated process. The local spawn agent should, therefore, ensure that the `stdin` channel to that process remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the spawned job are potential recipients. The requester will issue a call to `PMIx_IOF_push` to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

- **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
Requests that the ability to forward the `stdout` of the spawned processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and other options for delivery of the forwarded output.
- **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
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\_STDDIAG** "pmix.fwd.stddiag" (bool)  
Requests that the ability to forward the diagnostic channel (if it exists) of the spawned processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and other options for delivery of the forwarded output.
- **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csizes" (uint32\_t)  
The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.
- **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)  
In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.
- **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)  
In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).
- **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)  
Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be *flushed* to the callback upon a call to deregister the respective channel.
- **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)  
Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
- **PMIX\_IOF\_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 17.4.3 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 17.4.3 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:

- 1     ● **PMIX\_KEEPALIVE\_PIPE** - an integer **read**-end of a POSIX pipe that the IL should monitor  
2                  for closure, thereby indicating that the initiator has terminated.

3       The initiator then calls the **PMIx\_Spawn** API so that the PMIx library can either communicate the  
4                  spawn request to a server (if connected to one), or locally spawn the IL itself if not connected to a  
5                  server and the PMIx implementation includes self-spawn support. **PMIx\_Spawn** shall return an  
6                  error if neither of these conditions is met.

7       When initialized by the IL, the **PMIx\_tool\_init** function must perform two operations:

- 8       ● check for the presence of the **PMIX\_KEEPALIVE\_PIPE** environmental variable - if provided,  
9                  then the library shall monitor the pipe for closure, providing a **PMIX\_EVENT\_JOB\_END** event  
10                 when the pipe closes (thereby indicating the termination of the initiator). The IL should register  
11                 for this event after completing **PMIx\_tool\_init** - the initiator's namespace can be obtained  
12                 via a call to **PMIx\_Get** with the **PMIX\_PARENT\_ID** key. Note that this feature will only be  
13                 available if the spawned IL is local to the initiator.
- 14       ● check for the **PMIX\_LAUNCHER\_RNDZ\_URI** environmental parameter - if found, the library  
15                 shall connect back to the initiator using the **PMIx\_tool\_attach\_to\_server** API,  
16                 retaining its current server as its primary server.

17       Once the IL completes **PMIx\_tool\_init**, it must register for the **PMIX\_EVENT\_JOB\_END**  
18                  termination event and then idle until receiving that event - either directly from the initiator, or from  
19                  the PMIx library upon detecting closure of the keepalive pipe. The IL idles in the intervening time  
20                  as it is solely acting as a relay (if connected to a server that is performing the actual application  
21                  launch) or as a PMIx server responding to spawn requests.

22       Upon return from the **PMIx\_Spawn** API, the initiator should set the spawned IL as its primary  
23                  server using the **PMIx\_tool\_set\_server** API with the nspace returned by **PMIx\_Spawn** and  
24                  any valid rank (a rank of zero would ordinarily be used as only one IL process is typically started).  
25                  It is advisable to set a connection timeout value when calling this function. The initiator can then  
26                  proceed to spawn the actual application according to the procedure described in Section 17.2.1.

### 27     17.2.2.2 IL-based command line parsing

28       In the case where the initiator cannot parse its command line, it must defer that parsing to the IL. A  
29                  common example is provided below:

30       \$ initiator mpiexec --verbose -n 3 ./app <appoptions>

31       For this situation, the initiator proceeds as above with only one notable exception: instead of calling  
32                  **PMIx\_Spawn** twice (once to start the IL and again to start the actual application), the initiator only  
33                  calls that API one time:

- 34       ● The *app* parameter passed to the spawn request contains only one **pmix\_app\_t** that contains  
35                  the entire command line, including both launcher and application(s).
- 36       ● The launcher executable must be in the *app.cmd* field and in *app.argv[0]*, with the rest of the  
37                  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\_KEEPALIVE\_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.

### 17.2.3 Tool spawn-related attributes

Tools are free to utilize the spawn attributes available to applications (see 11.2.4) 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 17.4.3 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 17.4.3 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.

```

### 17.2.4 Tool rendezvous-related events

The following constants refer to events relating to rendezvous of a tool and launcher during spawn of the IL.

**PMIX\_LAUNCHER\_READY** An application launcher (e.g., *mpiexec*) shall generate this event to signal a tool that started it that the launcher is ready to receive directives/commands (e.g., **PMIx\_Spawn**). This is only used when the initiator is able to parse the command line itself, or the launcher is started as a persistent Distributed Virtual Machine (DVM).

## 17.3 IO Forwarding

Underlying the operation of many tools is a common need to forward **stdin** from the tool to 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 17.3.5 that are supported by the  
23 implementation.

### 24 17.3.1 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 *mpexec* - 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 17.4.

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.

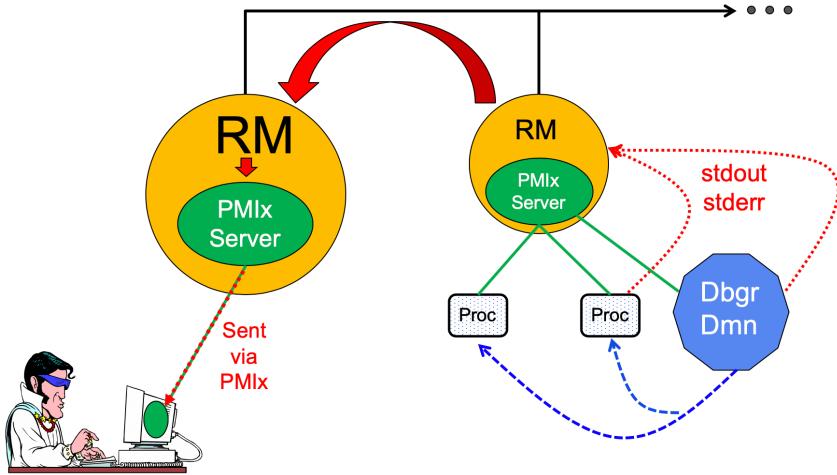


Figure 17.4.: Forwarding stdout/stderr

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\_TOF\_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\_TOF\_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 17.3.2 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:

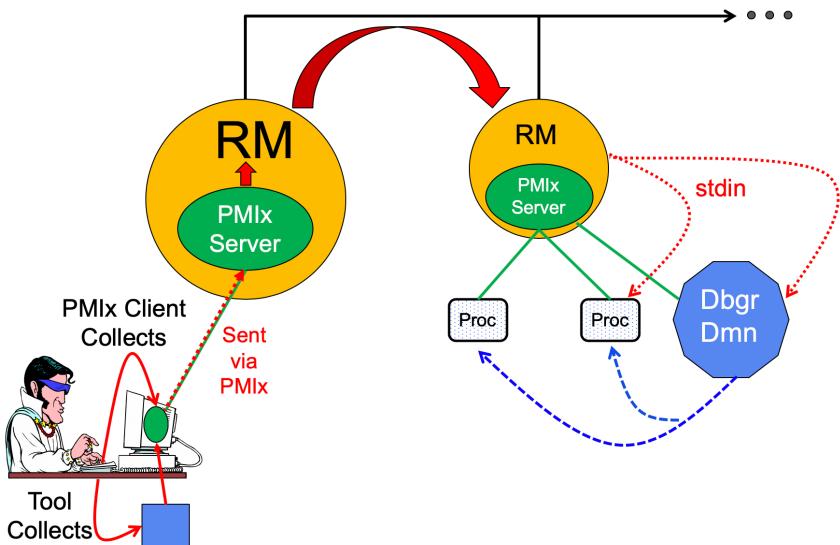


Figure 17.5.: Forwarding **stdin**

- 15 • internal collection by the PMIx tool library itself. This is requested via the  
16 **PMIX\_TOF\_PUSH\_STDIN** attribute in the **PMIx\_TOF\_push** call. When this mode is  
17 selected, the tool library begins collecting all **stdin** data and internally passing it to the local

1 server for distribution to the specified target processes. All collected data is sent to the same  
2 targets until **stdin** is closed, or a subsequent call to **PMIx\_IOF\_push** is made that includes  
3 the **PMIX\_IOF\_COMPLETE** attribute indicating that forwarding of **stdin** is to be terminated.

- 4 • external collection directly by the tool. It is assumed that the tool will provide its own  
5 code/mechanism for collecting its **stdin** as the tool developers may choose to insert some  
6 filtering and/or editing of the stream prior to forwarding it. In addition, the tool can directly  
7 control the targets for the data on a per-call basis – i.e., each call to **PMIx\_IOF\_push** can  
8 specify its own set of target recipients for that particular *blob* of data. Thus, this method provides  
9 maximum flexibility, but requires that the tool developer provide their own code to capture  
10 **stdin**.

11 Note that it is the responsibility of the RM to forward data to the host where the target process(es)  
12 are executing, and for the host daemon on that node to deliver the data to the **stdin** of target  
13 process(es). The PMIx server on the remote node is not involved in this process. Systems that do  
14 not support forwarding of **stdin** shall return **PMIX\_ERR\_NOT\_SUPPORTED** in response to a  
15 forwarding request.

---

### Advice to users

---

16 Scalable forwarding of **stdin** represents a significant challenge. Most environments will at least  
17 handle a *send-to-1* model whereby **stdin** is forwarded to a single identified process, and  
18 occasionally an additional *send-to-all* model where **stdin** is forwarded to all processes in the  
19 application. Users are advised to check their host environment for available support as the  
20 distribution method lies outside the scope of PMIx.

21 **Stdin** buffering by the RM and/or PMIx library can be problematic. If any targeted recipient is  
22 slow reading data (or decides never to read data), then the data must be buffered in some  
23 intermediate daemon or the PMIx tool library itself. Thus, piping a large amount of data into  
24 **stdin** can result in a very large memory footprint in the system management stack or the tool.  
25 Best practices, therefore, typically focus on reading of input files by application processes as  
26 opposed to forwarding of **stdin**.

---

### 27 17.3.3 IO Forwarding Channels

28 *PMIx v3.0*

29 The **pmix\_ifc\_channel\_t** structure is a **uint16\_t** type that defines a set of bit-mask flags  
30 for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple  
channels.

|    |                                 |                                        |
|----|---------------------------------|----------------------------------------|
| 31 | <b>PMIX_FWD_NO_CHANNELS</b>     | Forward no channels.                   |
| 32 | <b>PMIX_FWD_STDIN_CHANNEL</b>   | Forward <b>stdin</b> .                 |
| 33 | <b>PMIX_FWD_STDOUT_CHANNEL</b>  | Forward <b>stdout</b> .                |
| 34 | <b>PMIX_FWD_STDERR_CHANNEL</b>  | Forward <b>stderr</b> .                |
| 35 | <b>PMIX_FWD_STDDIAG_CHANNEL</b> | Forward <b>stddiag</b> , if available. |
| 36 | <b>PMIX_FWD_ALL_CHANNELS</b>    | Forward all available channels.        |

## 17.3.4 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.

## 17.3.5 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.cszie" (**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 17.4 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.
  - 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 11.2.4 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)`**

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the application is being spawned under a debugger, and that the local launch agent is to pause the resulting application processes on first instruction for debugger attach. The launcher (RM or IL) is to generate the `PMIX_LAUNCH_COMPLETE` event when all processes are stopped at the exec point. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

- `PMIX_DEBUG_STOP_IN_INIT "pmix.dbg.init" (bool)`

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The PMIx client library in each resulting application process shall notify its PMIx server that it is pausing and then pause during `PMIx_Init` of the spawned processes until 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 implementations that do not support this operation shall return an error from `PMIx_Init` if this behavior is requested. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

- `PMIX_DEBUG_WAIT_FOR_NOTIFY "pmix.dbg.notify" (bool)`

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The resulting application processes are to notify their server (by generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some application-determined location and pause at that point until 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. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

Note that there is no mechanism by which the PMIx library or the launcher can verify that an application will recognize and support the `PMIX_DEBUG_WAIT_FOR_NOTIFY` request. Debuggers utilizing this attachment method must, therefore, be prepared to deal with the case where the application fails to recognize and/or honor the request.

If the PMIx implementation and/or the host environment support it, debuggers can utilize the `PMIx_Query_info` API to determine which features are available via the `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute.

- `PMIX_DEBUG_STOP_IN_INIT` by checking `PMIX_CLIENT_ATTRIBUTES` for the

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 17.6) - 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\_NSPACE** 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\_NSPACE** 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      17.4.1 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:

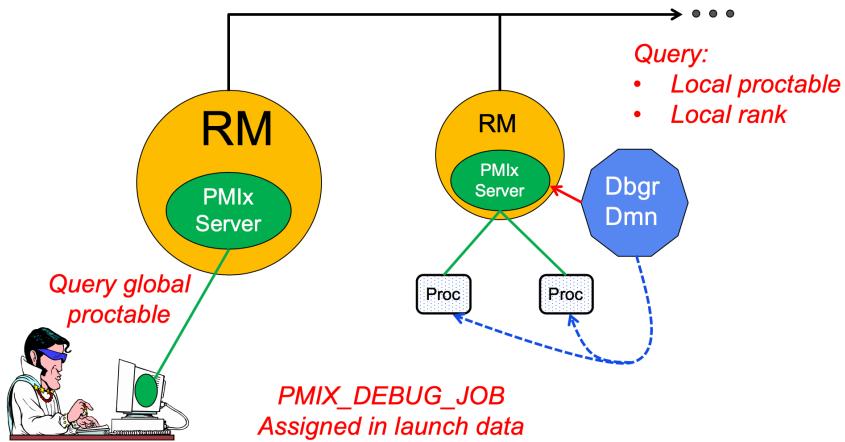


Figure 17.6.: Obtaining proctables

- 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 **17.4.2 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.

### 17.4.3 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. 17.7a), 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 to  
19 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 17.4.2.

24 In contrast, the **PMIX\_EXEC\_AGENT** attribute (Fig. 17.7b) 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.

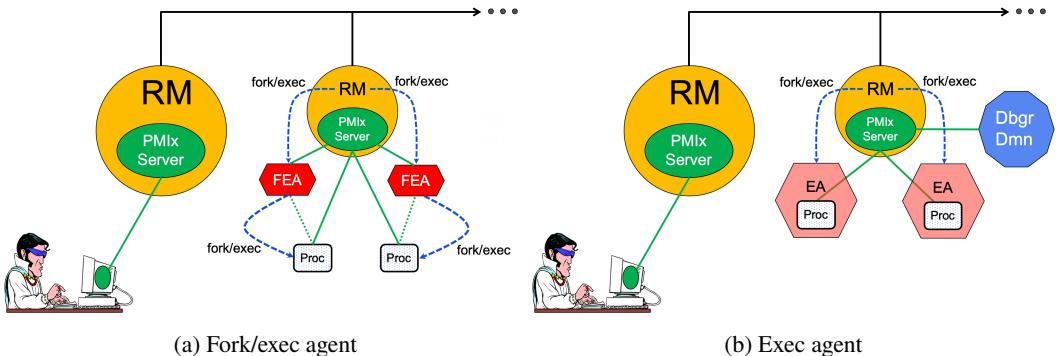


Figure 17.7.: Intermediate agents

## 17.4.4 Tracking the job lifecycle

There are a wide range of events a debugger can register to receive, but three are specifically defined for tracking a job's progress:

- **PMIX\_EVENT\_JOB\_START** indicates when the first process in the job has been spawned.
  - **PMIX\_LAUNCH\_COMPLETE** indicates when the last process in the job has been spawned.
  - **PMIX\_EVENT\_JOB\_END** indicates that all processes have terminated.

Each event is required to contain at least the namespace of the corresponding job and a **PMIX\_EVENT\_TIMESTAMP** indicating the time the event occurred. In addition, the **PMIX\_EVENT\_JOB\_END** event shall contain the returned status code (**PMIX\_JOB\_TERM\_STATUS**) for the corresponding job, plus the identity (**PMIX\_PROCID**) and exit status (**PMIX\_EXIT\_CODE**) of the first failed process, if applicable. Generation of these events by the launcher can be requested by including the **PMIX\_NOTIFY\_JOB\_EVENTS** attributes in the spawn request. Note that these events can be logged via the **PMIX\_Log** API by including the **PMIX\_LOG\_JOB\_EVENTS** attribute - this can be done either in conjunction with generated events, or in place of them.

Alternatively, if the debugger or tool solely wants to be alerted to job termination, then including the `PMIX_NOTIFY_COMPLETION` attribute in the spawn request would suffice. This attribute directs the launcher to provide just the `PMIX_EVENT_JOB_END` event. Note that this event can be logged via the `PMIx_Log` API by including the `PMIX_LOG_COMPLETION` attribute - this can be done either in conjunction with the generated event, or in place of it.

## Advice to users

The PMIx server is required to cache events in order to avoid race conditions - e.g., when a tool is trying to register for the [PMIX\\_EVENT\\_JOB\\_END](#) event from a very short-lived job. Accordingly, registering for job-related events can result in receiving events relating to jobs other than the one of interest.

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 17.4.4.1 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\_NSPACE** 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\_NSPACE** 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\_NSPACE** 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 17.4.4.2 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 17.4.5 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 17.4.6 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

```

```

1 used to cospawn debugger daemons alongside an application.
2 PMIX_DEBUG_DAEMONS_PER_PROC "pmix.dbg.dpproc" (uint16_t)
3 Number of debugger daemons to be spawned per application process. The launcher is to pass
4 the identifier of the namespace to be debugged by including the PMIX_DEBUG_TARGET
5 attribute in the daemon's job-level information. The debugger daemons spawned on a given
6 node are responsible for self-determining their specific target process(es) - e.g., by
7 referencing their own PMIX_LOCAL_RANK in the daemon debugger job versus the
8 corresponding PMIX_LOCAL_RANK of the target processes on the node.
9 PMIX_DEBUG_DAEMONS_PER_NODE "pmix.dbg.dpnd" (uint16_t)
10 Number of debugger daemons to be spawned on each node where the target job is executing.
11 The launcher is to pass the identifier of the namespace to be debugged by including the
12 PMIX_DEBUG_TARGET attribute in the daemon's job-level information. The debugger
13 daemons spawned on a given node are responsible for self-determining their specific target
14 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.
16 PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
17 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
18 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
19 PMIX_NSPACE indicating the namespace whose process table is being queried.
20 PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
21 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
22 process in the specified namespace executing on the same node as the requester, ordered by
23 process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace
24 whose local process table is being queried. OPTIONAL QUALIFIER: PMIX_HOSTNAME
25 indicating the host whose local process table is being queried. By default, the query assumes
26 that the host upon which the request was made is to be used.

```

## 27 17.5 Tool-Specific APIs

28 PMIx-based tools automatically have access to all PMIx client functions. Tools designated as a  
29 *launcher* or a *server* will also have access to all PMIx server functions. There are, however, an  
30 additional set of functions (described in this section) that are specific to a PMIx tool. Access to  
31 those functions require use of the tool initialization routine.

### 32 17.5.1 **PMIx\_tool\_init**

#### 33 Summary

34 Initialize the PMIx library for operating as a tool, optionally connecting to a specified PMIx server.

#### 35 Format

*PMIx v2.0*

```
1 pmix_status_t
2 PMIx_tool_init(pmix_proc_t *proc,
3 pmix_info_t info[], size_t ninfo);
4
5 INPUT proc
6 pmix_proc_t structure (handle)
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
12 Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
13
14 Required Attributes
15
16 The following attributes are required to be supported by all PMIx libraries:
17
18 PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
19 Name of the namespace to use for this tool.
20
21 PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
22 Rank of this tool.
23
24 PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
25 The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.
26
27 PMIX_TOOL_ATTACHMENT_FILE "pmix.tool.attach" (char*)
28 Pathname of file containing connection information to be used for attaching to a specific
29 server.
30
31 PMIX_SERVER_URI "pmix.srvr.uri" (char*)
32 URI of the PMIx server to be contacted.
33
34 PMIX_TCP_URI "pmix.tcp.uri" (char*)
35 The URI of the PMIx server to connect to, or a file name containing it in the form of
36 file:<name of file containing it>.
37
38 PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
39 PID of the target PMIx server for a tool.
40
41 PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
42 Name of the namespace to use for this PMIx server.
43
44 PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)
45 The requester requires that a connection be made only to a local, system-level PMIx server.
46
47 PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool)
```

1 Preferentially, look for a system-level PMIx server first.

## Optional Attributes

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

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\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)

7 Maximum number of times to try to connect to PMIx server - the default value is  
8 implementation specific.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
1 PMIX_EXTERNAL_PROGRESS "pmix.evext" (bool)
2 The host shall progress the PMIx library via calls to PMIx_Progress
3
4 PMIX_EVENT_BASE "pmix.evbase" (void*)
5 Pointer to an event_base to use in place of the internal progress thread. All PMIx library
6 events are to be assigned to the provided event base. The event base must be compatible with
7 the event library used by the PMIx implementation - e.g., either both the host and PMIx
8 library must use libevent, or both must use libev. Cross-matches are unlikely to work and
9 should be avoided - it is the responsibility of the host to ensure that the PMIx
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\_NSPACE** 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 17.1. 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 17.5.2 **PMIx\_tool\_finalize**

### 31 Summary

32 Finalize the PMIx tool library.

1   **Format**

2    `pmix_status_t`  
3    `PMIx_tool_finalize(void);`

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

5   **Description**

6   Finalize the PMIx tool library, closing all existing connections to servers. An error code will be  
7   returned if, for some reason, a connection cannot be cleanly terminated — in such cases, the  
8   connection is dropped. Upon detecting loss of the connection, the PMIx server shall cleanup all  
9   associated records of the tool.

10 **17.5.3 PMIx\_tool\_disconnect**

11 **Summary**

12   Disconnect the PMIx tool from the specified server connection while leaving the tool library  
13   initialized.

14 *PMIx v4.0*   **Format**

15    `pmix_status_t`  
16    `PMIx_tool_disconnect(const pmix_proc_t *server);`

17   **IN**   `server`  
18       `pmix_proc_t` structure (handle)

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

20 **Description**

21   Close the current connection to the specified server, if one has been made, while leaving the PMIx  
22   library initialized. An error code will be returned if, for some reason, the connection cannot be  
23   cleanly terminated - in this case, the connection is dropped. In either case, the library will remain  
24   initialized. Upon detecting loss of the connection, the PMIx server shall cleanup all associated  
25   records of the tool.

26   Note that if the server being disconnected is the current *primary* server, then all operations  
27   requiring support from a server will return the `PMIX_ERR_UNREACH` error until the tool either  
28   designates an existing connection to be the *primary* server or, if no other connections exist, the tool  
29   establishes a connection to a PMIx server.

## 17.5.4 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 corresponding to a PMIx error constant.

### Required Attributes

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

20 **PMIX\_TOOL\_ATTACHMENT\_FILE** "pmix.tool.attach" (`char*`)

21 Pathname of file containing connection information to be used for attaching to a specific  
22 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\_NSPACE** "pmix.srv.nspace" (`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.srvr" (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 17.1, 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.

## Advice to PMIx library implementers

21 When a tool connects to a server that is under a different namespace manager (e.g., host RM) from  
22 the prior server, the namespace in the identifier of the tool must remain unique in the new universe.  
23 If the namespace of the tool fails to meet this criteria in the new universe, then the new namespace  
24 manager is required to return an error and the connection attempt must fail.

## Advice to users

25 Some PMIx implementations may not support connecting to a server that is not under the same  
26 namespace manager (e.g., host RM) as the server to which the tool is currently connected.

## 17.5.5 **PMIx\_tool\_get\_servers**

### Summary

28 Get an array containing the **pmix\_proc\_t** process identifiers of all servers to which the tool is  
29 currently connected.

**Format**

`pmix_status_t  
PMIx_tool_get_servers(pmix_proc_t *servers[], size_t *nservers);`

**OUT servers**  
Address where the pointer to an array of `pmix_proc_t` structures shall be returned (handle)

**INOUT nservers**  
Address where the number of elements in *servers* shall be returned (handle)

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

**Description**  
Return an array containing the `pmix_proc_t` process identifiers of all servers to which the tool is currently connected. The process identifier of the current primary server shall be the first entry in the array, with the remaining entries in order of attachment from earliest to most recent.

## 13 17.5.6 PMIx tool set server

## **Summary**

**Format** C

```
pmix_status_t
PMIx_tool_set_server(const pmix_proc_t *server,
 pmix_info_t info[], size_t ninfo);
```

IN server  
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 corresponding to a PMIx error constant.

## Required Attributes

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

2 **PMIX\_WAIT\_FOR\_CONNECTION** "pmix.wait.conn" (bool)

3 Wait until the specified process has connected to the requesting tool or server, or the  
4 operation times out (if the **PMIX\_TIMEOUT** directive is included in the request).

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 Designate the specified server to be the tool's *primary* server for all subsequent API calls.

## 11 17.5.7 **PMIx\_IOF\_pull**

### 12 **Summary**

13 Register to receive output forwarded from a set of remote processes.

### 14 **Format**

15       **pmix\_status\_t**  
16       **PMIx\_IOF\_pull**(**const pmix\_proc\_t procs[], size\_t nprocs,**  
17                   **const pmix\_info\_t directives[], size\_t ndirs,**  
18                   **pmix\_iof\_channel\_t channel,**  
19                   **pmix\_iof\_cbfunc\_t cbfunc,**  
20                   **pmix\_hdlr\_reg\_cbfunc\_t regcbfunc,**  
21                   **void \*regcbdata);**

C

C

### 22 **IN procs**

23       Array of proc structures identifying desired source processes (array of handles)

### 24 **IN nprocs**

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

### 26 **IN directives**

27       Array of **pmix\_info\_t** structures (array of handles)

### 28 **IN ndirs**

29       Number of elements in the *directives* array (integer)

### 30 **IN channel**

31       Bitmask of IO channels included in the request (**pmix\_iof\_channel\_t**)

### 32 **IN cbfunc**

33       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 corresponding to a PMIx error constant. In the event  
7   the function returns 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.csizes`" (`uint32_t`)  
10   The requested size of the PMIx server cache in bytes for each specified channel. By default,  
11   the server is allowed (but not required) to drop all bytes received beyond the max size.  
12   **`PMIX_IOF_DROP_OLDEST`** "`pmix.iof.old`" (`bool`)  
13   In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
14   cache.  
15   **`PMIX_IOF_DROP_NEWEST`** "`pmix.iof.new`" (`bool`)  
16   In an overflow situation, the PMIx server is to drop any new bytes received until room  
17   becomes available in the cache (default).

### Optional Attributes

18   The following attributes 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 nspace,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.

#### 5 Advice to users

6 Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated  
7 channels to be written to their corresponding **stdout/stderr** file descriptors. Use of  
8 **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but should be  
used carefully due to bandwidth and memory footprint considerations.

## 9 17.5.8 PMIx\_IOF\_deregister

### 10 Summary

11 Deregister from output forwarded from a set of remote processes.

### 12 Format

```
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 Returns one of the following:

- 29 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
30 will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback  
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     • a PMIx error constant indicating either an error in the input or that the request was immediately  
4        processed and failed - the *cbfunc* will *not* be called

5     **Description**  
6     Deregister from output forwarded from a set of remote processes.

7                   **Advice to PMIx library implementers**

8     Any currently buffered IO should be flushed upon receipt of a deregistration request. All received  
IO after receipt of the request shall be discarded.

9     **17.5.9 PMIx\_IOF\_push**

10    **Summary**

11    Push data collected locally (typically from **stdin** or a file) to **stdin** of the target recipients.

12    **Format**

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*);

- 18    **IN    targets**  
19      Array of proc structures identifying desired target processes (array of handles)  
20    **IN    ntargets**  
21      Number of elements in the *targets* array (integer)  
22    **IN    bo**  
23      Pointer to **pmix\_byte\_object\_t** containing the payload to be delivered (handle)  
24    **IN    directives**  
25      Array of **pmix\_info\_t** structures (array of handles)  
26    **IN    ndirs**  
27      Number of elements in the *directives* array (integer)  
28    **IN    cbfunc**  
29      Callback function to be called when operation has been completed. (**pmix\_op\_cbfunc\_t**  
30        function reference)

IN cbdata

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

Returns one of the following:

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

## Required Attributes

The following attributes are required for PMIx libraries that support IO forwarding:

**PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csizes" (uint32\_t)

The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

**PMIX\_IOF\_DROP\_OLEDEST** "pmix.iof.old" (bool)

In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.

**PMIX IOF DROP NEWEST "pmix.iof.new" (bool)**

In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).

## Optional Attributes

The following attributes are optional for PMIx libraries that support IO forwarding:

**PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)

Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be *flushed* to the callback upon a call to deregister the respective channel.

**PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)

Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.

**PMIX\_IOF\_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
- 9
- 10
- 11
- 12
- push data collected by the caller themselves (typically from **stdin** or a file) to **stdin** of the target recipients;
  - request that the PMIx library automatically collect and push the **stdin** of the caller to the target recipients; or
  - indicate that automatic collection and transmittal of **stdin** is to stop
- 

13 **Advice to users**

14 Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires  
15 the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets.  
16 Use of **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but should be used carefully due to bandwidth and memory footprint considerations.



## CHAPTER 18

# Storage Support Definitions

---

1      **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.

9      PMIx enables users to better understand storage hierarchies by defining attributes that formalize  
10     storage system characteristics, state, and other parameters. These attributes can be queried by  
11     applications, I/O libraries and middleware, and workflow systems to discover available storage  
12     resources and to inform on which resources are most suitable for different I/O workload  
13     requirements.

## 14     18.1 Storage support constants

15      **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.

18      **Provisional**

`PMIX_STORAGE_MEDIUM_UNKNOWN`      The storage medium type is unknown.

19      **Provisional**

`PMIX_STORAGE_MEDIUM_TAPE`      The storage system uses tape media.

20      **Provisional**

`PMIX_STORAGE_MEDIUM_HDD`  
21          interfaces.      The storage system uses HDDs with traditional SAS, SATA

22      **Provisional**

`PMIX_STORAGE_MEDIUM_SSD`  
23          interfaces.      The storage system uses SSDs with traditional SAS, SATA

24      **Provisional**

`PMIX_STORAGE_MEDIUM_NVME`      The storage system uses SSDs with NVMe interface.

25      **Provisional**

`PMIX_STORAGE_MEDIUM_PMEM`      The storage system uses persistent memory.

26      **Provisional**

`PMIX_STORAGE_MEDIUM_RAM`      The storage system is volatile (e.g., tmpfs).

## Advice to PMIx library implementers

1 PMIx implementations should maintain the same ordering for bit-mask values for  
2 `pmix_storage_medium_t` struct as provided in this standard, since these constants are ordered  
3 to provide semantic information that may be of use to PMIx users. Namely,  
4 `pmix_storage_medium_t` constants are ordered in terms of increasing medium bandwidth.

5 It is further recommended that implementations should try to allocate empty bits in the mask so  
6 that they can be extended to account for new constant definitions corresponding to new storage  
7 mediums.

8 *Provisional* The `pmix_storage_accessibility_t` is a `uint64_t` type that defines a set of bit-mask  
9 flags for specifying different levels of storage accessibility (i.e., from where a storage system may  
10 be accessed). These can be bitwise OR'd together to accommodate storage systems that are  
11 accessible in multiple ways.

12 *Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_NODE** The storage system resources are accessible  
13 within the same node.

14 *Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_SESSION** The storage system resources are accessible  
15 within the same session.

16 *Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_JOB** The storage system resources are accessible  
17 within the same job.

18 *Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_RACK** The storage system resources are accessible  
19 within the same rack.

20 *Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_CLUSTER** The storage system resources are accessible  
21 within the same cluster.

22 *Provisional* **PMIX\_STORAGE\_ACCESSIBILITY\_REMOTE** The storage system resources are remote.

23 *Provisional* The `pmix_storage_persistence_t` type specifies different levels of persistence for a  
24 particular storage system.

25 *Provisional* **PMIX\_STORAGE\_PERSISTENCE\_TEMPORARY** Data on the storage system is persisted only  
26 temporarily (i.e., it does not survive across sessions or node reboots).

27 *Provisional* **PMIX\_STORAGE\_PERSISTENCE\_NODE** Data on the storage system is persisted on the node.

28 *Provisional* **PMIX\_STORAGE\_PERSISTENCE\_SESSION** Data on the storage system is persisted for the  
29 duration of the session.

30 *Provisional* **PMIX\_STORAGE\_PERSISTENCE\_JOB** Data on the storage system is persisted for the  
31 duration of the job.

32 *Provisional* **PMIX\_STORAGE\_PERSISTENCE\_SCRATCH** Data on the storage system is persisted  
33 according to scratch storage policies (short-term storage, typically persisted for days to weeks).

34 *Provisional* **PMIX\_STORAGE\_PERSISTENCE\_PROJECT** Data on the storage system is persisted  
35 according to project storage policies (long-term storage, typically persisted for the duration of  
36 a project).

|                                                                                             |                                                                                                                                      |
|---------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <b>PMIX_STORAGE_PERSISTENCE_ARCHIVE</b>                                                     | Data on the storage system is persisted according to archive storage policies (long-term storage, typically persisted indefinitely). |
| The <b>pmix_storage_access_type_t</b> type specifies different storage system access types. |                                                                                                                                      |
| <b>PMIX_STORAGE_ACCESS_RD</b>                                                               | Provide information on storage system read operations.                                                                               |
| <b>PMIX_STORAGE_ACCESS_WR</b>                                                               | Provide information on storage system write operations.                                                                              |
| <b>PMIX_STORAGE_ACCESS_RDWR</b>                                                             | Provide information on storage system read and write operations.                                                                     |

## **18.2 Storage support attributes**

The following attributes may be returned in response to queries (e.g., [PMIx\\_Get](#) or [PMIx\\_Query\\_info](#)) made by processes or tools.

|                                       |                                                   |                                                                                                                |
|---------------------------------------|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>PMIX_STORAGE_ID</b>                | "pmix.strg.id" (char*)                            | An identifier for the storage system (e.g., lustre-fs1, daos-oss1, home-fs)                                    |
| <b>PMIX_STORAGE_PATH</b>              | "pmix.strg.path" (char*)                          | Mount point path for the storage system (valid only for file-based storage systems)                            |
| <b>PMIX_STORAGE_TYPE</b>              | "pmix.strg.type" (char*)                          | Type of storage system (i.e., "lustre", "gpfs", "daos", "ext4")                                                |
| <b>PMIX_STORAGE_VERSION</b>           | "pmix.strg.ver" (char*)                           | Version string for the storage system                                                                          |
| <b>PMIX_STORAGE_MEDIUM</b>            | "pmix.strg.medium" (pmix_storage_medium_t)        | Types of storage mediums utilized by the storage system (e.g., SSDs, HDDs, tape)                               |
| <b>PMIX_STORAGE_ACCESSIBILITY</b>     | "pmix.strg.access" (pmix_storage_accessibility_t) | Accessibility level of the storage system (e.g., within same node, within same session)                        |
| <b>PMIX_STORAGE_PERSISTENCE</b>       | "pmix.strg.persist" (pmix_storage_persistence_t)  | Persistence level of the storage system (e.g., scratch storage or archive storage)                             |
| <b>PMIX_QUERY_STORAGE_LIST</b>        | "pmix.strg.list" (char*)                          | Comma-delimited list of storage identifiers (i.e., <b>PMIX_STORAGE_ID</b> types) for available storage systems |
| <b>PMIX_STORAGE_CAPACITY_LIMIT</b>    | "pmix.strg.caplim" (double)                       | Overall limit on capacity (in bytes) for the storage system                                                    |
| <b>PMIX_STORAGE_CAPACITY_USED</b>     | "pmix.strg.capuse" (double)                       | Overall used capacity (in bytes) for the storage system                                                        |
| <b>PMIX_STORAGE_OBJECT_LIMIT</b>      | "pmix.strg.objlim" (uint64_t)                     | Overall limit on number of objects (e.g., inodes) for the storage system                                       |
| <b>PMIX_STORAGE_OBJECTS_USED</b>      | "pmix.strg.objuse" (uint64_t)                     | Overall used number of objects (e.g., inodes) for the storage system                                           |
| <b>PMIX_STORAGE_MINIMAL_XFER_SIZE</b> | "pmix.strg.minxfer" (double)                      |                                                                                                                |

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.

## A.1 Design Considerations

Several issues arose during design of the Python bindings:

### A.1.1 Error Codes vs Python Exceptions

The C programming language reports errors through the return of the corresponding integer status codes. PMIx has defined a range of negative values for this purpose. However, Python has the option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and return PMIx status codes in lieu of raising exceptions as this method was considered more consistent for those working in both domains.

### A.1.2 Representation of Structured Data

PMIx utilizes a number of C-language structures to efficiently bundle related information. For example, the PMIx process identifier is represented as a struct containing a character array for the namespace and a 32-bit unsigned integer for the process rank. There are several options for translating such objects to Python – e.g., the PMIx process identifier could be represented as a two-element tuple (nspc, rank) or as a dictionary ‘nspc’: name, ‘rank’: 0. Exploration found no discernible benefit to either representation, nor was any clearly identifiable rationale developed that would lead a user to expect one versus the other for a given PMIx data type. Consistency in the translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion. Hence, the decision was made to express all complex datatypes as Python dictionaries.

## 1 A.2 Datatype Definitions

2 PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range  
3 integers (e.g., `uint32_t`), and structures. Each datatype is represented by a named unsigned 16-bit  
4 integer (`uint16_t`) constant. Users are advised to use the named PMIx constants for indicating  
5 datatypes instead of integer values to ensure compatibility with future PMIx versions.

6 With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 12 directly  
7 translate to Python. However, Python lacks the size-specific value definitions of C (e.g., `uint8_t`)  
8 and thus some care must be taken to protect against overflow/underflow situations when moving  
9 between the languages. Python bindings that accept values including PMIx datatypes shall  
10 therefore have the datatype and associated value checked for compatibility with their PMIx-defined  
11 equivalents, returning an error if:

- 12 • datatypes not defined by PMIx are encountered
- 13 • provided values fall outside the range of the C-equivalent definition - e.g., if a value identified as  
14 `PMIX_UINT8` lies outside the `uint8_t` range

15 Note that explicit labeling of PMIx data type, even when Python itself doesn't care, is often  
16 required for the Python bindings to know how to properly interpret and label the provided value  
17 when passing it to the PMIx library.

18 Table A.1 lists the correspondence between data types in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                                       | PMIx Name                                                           | Python Definition                                              | Notes                                                                                                                                                |
|----------------------------------------------------|---------------------------------------------------------------------|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bool</b>                                        | <b>PMIX_BOOL</b>                                                    | boolean                                                        |                                                                                                                                                      |
| <b>byte</b>                                        | <b>PMIX_BYTE</b>                                                    | A single element byte array (i.e., a byte array of length one) |                                                                                                                                                      |
| <b>char*</b>                                       | <b>PMIX_STRING</b>                                                  | string                                                         |                                                                                                                                                      |
| <b>size_t</b>                                      | <b>PMIX_SIZE</b>                                                    | integer                                                        |                                                                                                                                                      |
| <b>pid_t</b>                                       | <b>PMIX_PID</b>                                                     | integer                                                        | value shall be limited to the <b>uint32_t</b> range                                                                                                  |
| <b>int, int8_t, int16_t, int32_t, int64_t</b>      | <b>PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64</b>      | integer                                                        | value shall be limited to its corresponding range                                                                                                    |
| <b>uint, uint8_t, uint16_t, uint32_t, uint64_t</b> | <b>PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64</b> | integer                                                        | value shall be limited to its corresponding range                                                                                                    |
| <b>float, double</b>                               | <b>PMIX_FLOAT, PMIX_DOUBLE</b>                                      | float                                                          | value shall be limited to its corresponding range                                                                                                    |
| <b>struct timeval</b>                              | <b>PMIX_TIMEVAL</b>                                                 | {'sec': sec, 'usec': microsec}                                 | each field is an integer value                                                                                                                       |
| <b>time_t</b>                                      | <b>PMIX_TIME</b>                                                    | integer                                                        | limited to positive values                                                                                                                           |
| <b>pmix_data_type_t</b>                            | <b>PMIX_DATA_TYPE</b>                                               | integer                                                        | value shall be limited to the <b>uint16_t</b> range                                                                                                  |
| <b>pmix_status_t</b>                               | <b>PMIX_STATUS</b>                                                  | integer                                                        |                                                                                                                                                      |
| <b>pmix_key_t</b>                                  | N/A                                                                 | string                                                         | The string's length shall be limited to one less than the size of the <b>pmix_key_t</b> array (to reserve space for the terminating <b>NULL</b> )    |
| <b>pmix_nspace_t</b>                               | N/A                                                                 | string                                                         | The string's length shall be limited to one less than the size of the <b>pmix_nspace_t</b> array (to reserve space for the terminating <b>NULL</b> ) |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                    | PMIx Name                     | Python Definition                                                                                                                            | Notes                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>pmix_rank_t</code>        | <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 17.3.3)                                                                                                                                                                                                                                                             |
| <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': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type} | <i>proc</i> is a Python <code>proc</code> dictionary; <i>key</i> is a Python string <code>key</code> ; <i>type</i> is the PMIx datatype of <i>value</i> ; and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype                                         |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                   | PMIx Name                     | Python Definition                                                                | Notes                                                                                                                                                                                                                                               |
|--------------------------------|-------------------------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>pmix_app_t</code>        | <code>PMIX_APP</code>         | {'cmd': cmd, 'argv': [argv], 'env': [env], 'maxprocs': maxprocs, 'info': [info]} | <i>cmd</i> is a Python string; <i>argv</i> and <i>env</i> are Python lists containing Python strings; <i>maxprocs</i> is an integer; and <i>info</i> is a Python list of <code>info</code> values                                                   |
| <code>pmix_query_t</code>      | <code>PMIX_QUERY</code>       | {'keys': [keys], 'qualifiers': [info]}                                           | <i>keys</i> is a Python list of Python strings, and <i>qualifiers</i> is a Python list of <code>info</code> values                                                                                                                                  |
| <code>pmix_regattr_t</code>    | <code>PMIX_REGATTR</code>     | {'name': name, 'key': key, 'type': type, 'info': [info], 'description': [desc]}  | <i>name</i> and <i>string</i> are Python strings; <i>type</i> is the PMIx datatype for the attribute's value; <i>info</i> is a Python list of <code>info</code> values; and <i>description</i> is a list of Python strings describing the attribute |
| <code>pmix_job_state_t</code>  | <code>PMIX_JOB_STATE</code>   | integer                                                                          | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                            |
| <code>pmix_link_state_t</code> | <code>PMIX_LINK_STATE</code>  | integer                                                                          | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                            |
| <code>pmix_cpuset_t</code>     | <code>PMIX_PROC_CPUSET</code> | {'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>   | <code>PMIX_LOCTYPE</code>     | list                                                                             | list of integer values (defined in Section 11.4.2.3) 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 <code>info</code> describing the fabric                                                                      |
| <code>pmix_endpoint_t</code>   | <code>PMIX_ENDPOINT</code>    | {'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 <code>byteobject</code> 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,<br>'osname': osname,<br>'mindist': mindist,<br>'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':<br>uuid, 'osname':<br>osname, 'coordinates':<br>[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 11.4.8)                                                                                                                                                                                                                                                                       |
| <code>pmix_bind_envelope_t</code>   | N/A                           | integer                                                                           | one of the values defined in Section 11.4.4.1                                                                                                                                                                                                                                                                            |

## 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:

```
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 iofcfunc(iohdlr:integer, channel:bitarray,
7 source:dict, payload:dict, info:list)
```

Python

8 IN **iohdlr**

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\\_iocbfunc\\_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)
3 Python
4
5 IN iofhdlr
6 Registration number of the handler being invoked (integer)
7 IN status
8 Status associated with the operation (integer)
9 IN source
10 Python proc identifying the namespace/rank of the process that generated the event (dict)
11 IN info
12 List of Python info provided by the source containing metadata about the event (list)
13 IN results
14 List of Python info containing the aggregated results of all prior evhandlers (list)
15
16 Returns:
17
18 • rc - Status returned by the event handler's operation (integer)
19
20 • results - List of Python info containing results from this event handler's operation on the event
21 (list)
22
23 See pmix_notification_fn_t for details
```

## A.3.3 Server Module Functions

The following definitions represent functions that may be provided to the PMIx server library at time of initialization for servicing of client requests. Module functions that are not provided default to returning "not supported" to the caller.

### A.3.3.1 Client Connected

#### Summary

Notify the host server that a client connected to this server.

#### Format

*PMIx v4.0*

## Python

```
1 def clientconnected2(proc:dict is not None, info:list)
```

## Python

2   **IN proc**

3       Python **proc** identifying the namespace/rank of the process that connected (dict)

4   **IN info**

5       list of Python **info** containing information about the process (list)

6 Returns:

7   • **rc** - **PMIX\_SUCCESS** or a PMIx error code indicating the connection should be rejected (integer)

8 See [\*\*pmix\\_server\\_client\\_connected2\\_fn\\_t\*\*](#) for details

### A.3.3.2 Client Finalized

#### Summary

Notify the host environment that a client called [\*\*PMIx\\_Finalize\*\*](#).

#### Format

## Python

```
12 PMIx v4.0 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

### A.3.3.3 Client Aborted

#### Summary

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
```

### A.3.3.4 Fence

#### Summary

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_fencenb_fn_t for details
```

### 1 A.3.3.5 Direct Modex

#### 2 Summary

3 Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
4 hosts the specified proc to obtain and return a direct modex blob for that proc.

#### 5 PMIx v4.0 Format

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 PMIx v4.0 Format

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     *PMIx v4.0*   **Format**

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     *PMIx v4.0*   **Format**

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**

5      

```
def spawn(args:dict is not None)
```

Python

Python

6      **IN args**

7      Python dictionary containing:

- 'proc': Python [proc](#) of process making the request (dict)
- 'jobinfo': Optional list of Python [info](#) job-level directives and information (list)
- 'apps': List of Python [app](#) describing applications to be spawned (list)

11     Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- *nspace* - Python string containing namespace of the spawned job (str)

14     See [pmix\\_server\\_spawn\\_fn\\_t](#) for details

15 **A.3.3.10 Connect**

16      **Summary**

17      Record the specified processes as *connected*.

18      **Format**

19      

```
def connect(args:dict is not None)
```

Python

Python

20      **IN args**

21      Python dictionary containing:

- 'procs': List of Python [proc](#) identifying the namespace/rank of the participating processes (list)
- 'directives': Optional list of Python [info](#) containing directives controlling the operation (list)

26     Returns:

- *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

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    PMIx v4.0      Format

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  
9                    (list)
- 10     • 'directives': Optional list of Python **info** containing directives controlling the operation  
11                    (list)

12     Returns:

- 13     • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

14     See [pmix\\_server\\_disconnect\\_fn\\_t](#) for details

### 15 A.3.3.12 Register Events

#### 16      Summary

17      Register to receive notifications for the specified events.

18    PMIx v4.0      Format

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  
24                    (list)

25     Returns:

- 26     • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

27     See [pmix\\_server\\_register\\_events\\_fn\\_t](#) for details

### 1 A.3.3.13 Deregister Events

#### 2 Summary

3 Deregister to receive notifications for the specified events.

#### 4 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.

```
1 Format Python
2 def query(args:dict is not None) Python
3 IN args
4 Python dictionary containing:
5 • 'source': Python proc of requesting process (dict)
6 • 'queries': List of Python query directives (list)
7 Returns:
8 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
9 • info - List of Python info containing the returned results (list)
10 See pmix_server_query_fn_t for details
```

### A.3.3.16 Tool Connected

#### Summary

Register that a tool has connected to the server.

```
14 PMIx v4.0 Format Python
15 def tool_connected(args:dict is not None) Python
16 IN args
17 Python dictionary containing:
18 • 'directives': Optional list of Python info info on the connecting tool (list)
19 Returns:
20 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
21 • proc - Python proc containing the assigned namespace:rank for the tool (dict)
22 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.

### **11 A.3.3.18 Allocate Resources**

## Summary

Request allocation operations on behalf of a client.

```
14 PMIx v4.0 Format Python
15 def allocate(args:dict is not None) Python
16 IN args
17 Python dictionary containing:
18 • 'source': Python proc of requesting process (dict)
19 • 'action': Python allocdir specifying requested action (integer)
20 • 'directives': Optional list of Python info containing directives (list)
21 Returns:
22 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
23 • refarginfo - List of Python info containing results of requested operation (list)
24 See pmix server alloc fn t for details.
```

25 A.3.3.19 Job Control

## Summary

Execute a job control action on behalf of a client.

```
1 Format Python
2 def job_control(args:dict is not None) Python
3 IN args
4 Python dictionary containing:
5 • 'source': Python proc of requesting process (dict)
6 • 'targets': List of Python proc specifying target processes (list)
7 • 'directives': Optional list of Python info containing directives (list)
8 Returns:
9 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
10 See pmix_server_job_control_fn_t for details.
```

### **11 A.3.3.20 Monitor**

## Summary

13 Request that a client be monitored for activity.

```
14 PMIx v4.0 Format Python
15 def monitor(args:dict is not None) Python
16 IN args
17 Python dictionary containing:
18 • 'source': Python proc of requesting process (dict)
19 • 'monitor': Python info attribute indicating the type of monitor being requested (dict)
20 • 'error': Status code to be used when generating an event notification (integer) alerting that
21 the monitor has been triggered.
22 • 'directives': Optional list of Python info containing directives (list)
23 Returns:
24 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
25 See pmix server monitor fn t for details.
```

### 26 A.3.3.21 Get Credential

## Summary

Request a credential from the host environment.

```
1 Format Python
2 def get_credential(args:dict is not None) Python
3 IN args
4 Python dictionary containing:
5 • 'source': Python proc of requesting process (dict)
6 • 'directives': Optional list of Python info containing directives (list)
7 Returns:
8 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
9 • cred - Python byteobject containing returned credential (dict)
10 • info - List of Python info containing any additional info about the credential (list)
11 See pmix_server_get_cred_fn_t for details.
```

### 12 A.3.3.22 Validate Credential

#### 13 Summary

14 Request validation of a credential

```
15 PMIx v4.0 Format Python
16 def validate_credential(args:dict is not None) Python
17 IN args
18 Python dictionary containing:
19 • 'source': Python proc of requesting process (dict)
20 • 'credential': Python byteobject containing credential (dict)
21 • 'directives': Optional list of Python info containing directives (list)
22 Returns:
23 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
24 • info - List of Python info containing any additional info from the credential (list)
25 See pmix_server_validate_cred_fn_t for details.
```

### 26 A.3.3.23 IO Forward

#### 27 Summary

28 Request the specified IO channels be forwarded from the given array of processes.

**Format**

```
def iof_pull(args:dict is not None)
```

**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.

IO Push

## Summary

Pass standard input data to the host environment for transmission to specified recipients.

| Format                                         | Python                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>def iof_push(args:dict is not None)</pre> | Python                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>IN args</b>                                 | Python dictionary containing:                                                                                                                                                                                                                                                                                                                                                                     |
|                                                | <ul style="list-style-type: none"><li>• 'source': Python <b>proc</b> of process whose input is being forwarded (dict)</li><li>• 'payload': Python <b>byteobject</b> containing input bytes (dict)</li><li>• 'targets': List of <b>proc</b> of processes that are to receive the payload (list)</li><li>• 'directives': Optional list of Python <b>info</b> containing directives (list)</li></ul> |
| Returns:                                       |                                                                                                                                                                                                                                                                                                                                                                                                   |
|                                                | <ul style="list-style-type: none"><li>• <i>rc</i> - <b>PMIX_SUCCESS</b> or a PMIx error code indicating the operation failed (integer)</li></ul>                                                                                                                                                                                                                                                  |
| See <b>pmix server stdin fn t</b> for details. |                                                                                                                                                                                                                                                                                                                                                                                                   |

## Group Operations

## Summary

Request group operations (construct, destruct, etc.) on behalf of a set of processes.

```
1 Format Python
2 def group(args:dict is not None) Python
3 IN args
4 Python dictionary containing:
5 • 'op': Operation host is to perform on the specified group (integer)
6 • 'group': String identifier of target group (str)
7 • 'procs': List of Python proc of participating processes (dict)
8 • 'directives': Optional list of Python info containing directives (list)
9 Returns:
10 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
11 • refarginfo - List of Python info containing results of requested operation (list)
12 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.

```
16 PMIx v4.0 Format Python
17 def fabric(args:dict is not None) Python
18 IN args
19 Python dictionary containing:
20 • 'source': Python proc of requesting process (dict)
21 • 'index': Identifier of the fabric being operated upon (integer)
22 • 'op': Operation host is to perform on the specified fabric (integer)
23 • 'directives': Optional list of Python info containing directives (list)
24 Returns:
25 • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
26 • refarginfo - List of Python info containing results of requested operation (list)
27 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 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 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**

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 **PMIx v4.0 Format**

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.

```
1 Format Python
2 rc = myclient.abort(status:integer, msg:str, targets:list)
3 IN status
4 PMIx status to be returned on exit (integer)
5 IN msg
6 String message to be printed (string)
7 IN targets
8 List of Python proc dictionaries (list)
9 Returns:
10 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
11 See PMIx Abort for description of all relevant attributes and behaviors.
```

## **12 A.4.6 Client.store internal**

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

```
15 PMIx v4.0 Format Python
16 rc = myclient.store_internal(proc:dict, key:str, value:dict)
17 IN proc Python proc dictionary of the process being referenced (dict)
18 IN key String key of the data (string)
19 IN value Python value dictionary (dict)
20
21 Returns:
22
23 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24
25 See PMIx Store internal for details.
```

26 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)`

3 **IN scope**  
4 Scope of the data being posted (integer)

5 **IN key**  
6 String key of the data (string)

7 **IN value**  
8 Python **value** dictionary (dict)

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()  
17                                                    Python  
18                                                    Returns:  
19                                                    

- `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**

**Summary**  
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**

**Format**

Python

```
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

**Format** Python  
`rc = myclient.publish(directives:list)` Python  
**IN directives**  
List of Python **info** dictionaries containing data to be published and directives (list)  
Returns:

- `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**

## Summary

Lookup information published by this or another process with [PMIx\\_Publish](#).

**Format** Python  
`rc,info = myclient.lookup(pdata:list, directives:list)`

**IN pdata**  
List of Python `pdata` dictionaries identifying data to be retrieved (list)

**IN directives**  
List of Python `info` dictionaries (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\\_Lookup](#) for description of all relevant attributes and behaviors.

## 21 A.4.13 Client.unpublish

## Summary

Delete data published by this process with [PMIx\\_Publish](#).

```
rc = myclient.unpublish(keys:list, directives:list)
```

**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.

## 10 A.4.14 Client.spawn

## Summary

Spawn a new job.

```
rc, nspace = myclient.spawn(jobinfo:list, apps:list)
```

- 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.

## 23 A.4.15 Client.connect

## Summary

Connect namespaces.

```
1 Format Python
2 rc = myclient.connect(peers:list, directives:list)
3 IN peers
4 List of Python proc dictionaries (list)
5 IN directives
6 List of Python info dictionaries (list)
7 Returns:
8 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9 See PMIx_Connect for description of all relevant attributes and behaviors.
```

## A.4.16 Client.disconnect

### Summary

Disconnect namespaces.

```
13 PMIx v4.0 Format Python
14 rc = myclient.disconnect(peers:list, directives:list)
15 IN peers
16 List of Python proc dictionaries (list)
17 IN directives
18 List of Python info dictionaries (list)
19 Returns:
20 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
21 See PMIx_Disconnect for description of all relevant attributes and behaviors.
```

## A.4.17 Client.resolve\_peers

### Summary

Return list of processes within the specified nspace on the given node.

1 **Format** Python

2 `rc, procs = myclient.resolve_peers(node:str, nspace:str)`

3 **IN node**  
4 Name of node whose processes are being requested (str)

5 **IN nspace**  
6 Python **nspace** whose processes are to be returned (str)

7 Returns:

8 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9 • `procs` - List of Python **proc** dictionaries (list)

10 See **PMIx\_Resolve\_peers** for description of all relevant attributes and behaviors.

#### 11 A.4.18 Client.resolve nodes

## 22 A.4.19 Client.query

**Summary**  
Query information about the system in general.

```
1 Format Python
2 rc,info = myclient.query(queries:list) Python
3 IN queries
4 List of Python query dictionaries (list)
5 Returns:
6 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
7 • info - List of Python info containing results of the query (list)
8 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) Python
14 IN data
15 List of Python info (list)
16 IN directives
17 Optional list of Python info (list)
18 Returns:
19 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
20 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.

```
1 Format Python
2 rc,info = myclient.allocation_request(request:integer, directives:list)
3 IN request
4 Python allocdir specifying requested operation (integer)
5 IN directives
6 List of Python info describing request (list)
7 Returns:
8 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9 • info - List of Python info containing results of the request (list)
10 See PMIx_Allocation_request for description of all relevant attributes and behaviors.
```

## 11 A.4.22 Client.job\_ctrl

## Summary

Request a job control action.

```
14 PMIx v4.0 Format Python
15 rc,info = myclient.job_ctrl(targets:list, directives:list)
16 IN targets
17 List of Python proc specifying targets of requested operation (integer)
18 IN directives
19 List of Python info describing operation to be performed (list)
20 Returns:
21 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
22 • info - List of Python info containing results of the request (list)
23 See PMIx_Job_control for description of all relevant attributes and behaviors.
```

## 24 A.4.23 Client.monitor

## Summary

Request that something be monitored.

**Format**

`rc,info = myclient.monitor(monitor:dict, error_code:integer, directives:list)`

**IN monitor**  
Python `info` 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**

**Format**

```
rc, cred = myclient.get_credential(directives: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)
- `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.

1     **Format**

2     `rc, info = myclient.validate_credential(cred:dict, directives:list)`

Python

3     **IN**    **cred**

4        Python `byteobject` containing credential (dict)

5     **IN**    **directives**

6        Optional list of Python `info` describing request (list)

7     Returns:

8        • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

9        • `info` - List of Python `info` containing additional results of the request (list)

10      See `PMIx_Validate_credential` for description of all relevant attributes and behaviors.

11     **A.4.26 Client.group\_construct**

12     **Summary**

13     Construct a new group composed of the specified processes and identified with the provided group  
14     identifier.

15     **PMIx v4.0 Format**

Python

16     `rc, info = myclient.construct_group(grp:string,  
17                                                          members:list, directives:list)`

Python

18     **IN**    **grp**

19        Python string identifier for the group (str)

20     **IN**    **members**

21        List of Python `proc` dictionaries identifying group members (list)

22     **IN**    **directives**

23        Optional list of Python `info` describing request (list)

24     Returns:

25        • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

26        • `info` - List of Python `info` containing results of the request (list)

27      See `PMIx_Group_construct` for description of all relevant attributes and behaviors.

28     **A.4.27 Client.group\_invite**

29     **Summary**

30     Explicitly invite specified processes to join a group.

1           **Format**                                                                                  Python  
2            rc,info = myclient.group\_invite(grp:string,  
3                                                                                                          members:list, directives:list)  
4                                                                                          Python

5         **IN grp**  
6           Python string identifier for the group (str)  
7         **IN members**  
8           List of Python **proc** dictionaries identifying processes to be invited (list)  
9         **IN directives**  
10           Optional list of Python **info** describing request (list)

11           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)

13           See **PMIx\_Group\_invite** for description of all relevant attributes and behaviors.

## 14 A.4.28 Client.group\_join

### 15           **Summary**

16           Respond to an invitation to join a group that is being asynchronously constructed.

17           **PMIx v4.0 Format**                                                          Python

18           rc,info = myclient.group\_join(grp:string,  
19                                                                                                  leader:dict, opt:integer,  
20                                                                                                          directives:list)  
21                                                                                          Python

22         **IN grp**  
23           Python string identifier for the group (str)  
24         **IN leader**  
25           Python **proc** dictionary identifying process leading the group (dict)  
26         **IN opt**  
27           One of the **pmix\_group\_opt\_t** values indicating decline/accept (integer)  
28         **IN directives**  
29           Optional list of Python **info** describing request (list)

30           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)

32           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      **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      **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 Format Python
2 rc,id = myclient.register_event_handler(codes:list,
3 directives:list, cbfunc)
4 IN codes
5 List of Python integer status codes that should be reported to this handler (list)
6 IN directives
7 Optional list of Python info describing request (list)
8 IN cbfunc
9 Python evhandler to be called when event is received (func)
10 Returns:
11 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
12 • id - PMIx reference identifier for handler (integer)
13 See PMIx_Register_event_handler for description of all relevant attributes and behaviors.
```

#### A.4.32 Client.deregister\_event\_handler

```
15 Summary
16 Deregister an event handler.
17 PMIx v4.0 Format Python
18 myclient.deregister_event_handler(id:integer)
19 IN id
20 PMIx reference identifier for handler (integer)
21 Returns: None
22 See PMIx_Deregister_event_handler for description of all relevant attributes and
23 behaviors.
```

#### A.4.33 Client.notify\_event

```
25 Summary
26 Report an event for notification via any registered handler.
```

```
1 Format Python
2 rc = myclient.notify_event(status:integer, source:dict,
3 range:integer, directives:list)
4 IN status
5 PMIx status code indicating the event being reported (integer)
6 IN source
7 Python proc of the process that generated the event (dict)
8 IN range
9 Python range in which the event is to be reported (integer)
10 IN directives
11 Optional list of Python info dictionaries describing the event (list)
12 Returns:
13 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
14 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.

```
18 PMIx v4.0 Format Python
19 rc, idx, fabricinfo = myclient.fabric_register(directives:list)
20 IN directives
21 Optional list of Python info containing directives (list)
22 Returns:
23 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24 • idx - Index of the registered fabric (integer)
25 • fabricinfo - List of Python info containing fabric info (list)
26 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)` Python

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)` Python

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()` Python

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      **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      **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.

1 Format  
2 rc, cpuset = myclient.parse\_cpuset\_string(cpuset:string)  
3 IN cpuset  
4 String returned by [PMIxServer.generate\\_cpuset\\_string](#)(string)

5 Returns:

- 6 • rc - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- 7 • cpuset - [cpuset](#) containing the source and bitmap of the cpuset (dict)

8 See [PMIx\\_Parse\\_cpuset\\_string](#) for details.

## 9 A.4.41 Client.compute\_distances

### 10 Summary

11 Compute distances from specified process location to local devices.

### 12 PMIx v4.0 Format

13 rc,distances = myclient.compute\_distances(cpuset:dict, info:list)  
14 IN cpuset  
15 cpuset describing the location of the process (dict)

### 16 IN info

17 List of info dictionaries describing the devices whose distance is to be computed (list)

18 Returns:

- 19 • rc - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- 20 • distances - List of [devdist](#) structures containing the distances from the caller to the specified  
21 devices (list)

22 See [PMIx\\_Compute\\_distances](#) for details. Note that distances can only be computed against  
23 the local topology.

## 24 A.4.42 Client.error\_string

### 25 Summary

26 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      **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      **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) 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) 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) 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**

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**

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.

1           **Format**                                                                  Python  
2           rep = myclient.get\_attribute\_string(attribute:str)  
3           **IN attribute**  
4           PMIx attribute name (string)  
5           Returns:  
6           • rep - String representation of the provided attribute (str)  
7           See [PMIx\\_Get\\_attribute\\_string](#) for further details.

## 8 A.4.53 Client.get\_attribute\_name

### 9           **Summary**

10           Pretty-print name of a PMIx attribute corresponding to the provided string.

11 *PMIx v4.0*           **Format**                                                  Python  
12           rep = myclient.get\_attribute\_name(attribute:str)  
13           **IN attributestring**  
14           Attribute string (string)  
15           Returns:  
16           • rep - Attribute name corresponding to the provided string (str)  
17           See [PMIx\\_Get\\_attribute\\_name](#) for further details.

## 18 A.4.54 Client.link\_state\_string

### 19           **Summary**

20           Pretty-print string representation of [pmix\\_link\\_state\\_t](#).

21 *PMIx v4.0*           **Format**                                                  Python  
22           rep = myclient.link\_state\_string(state:integer)  
23           **IN state**  
24           PMIx link state value (integer)  
25           Returns:  
26           • rep - String representation of the provided link state (str)  
27           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      **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      **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)
3 IN directives Python
4 List of Python info dictionaries (list)
5 IN map Python
6 Python dictionary key-function pairs that map server module callback functions to
7 provided implementations (see pmix_server_module_t) (dict)
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

## Summary

Finalize the PMIx server library.

```
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

## Summary

Generate a regular expression representation of the input strings.

```
1 Format Python
2 rc, regex = myserver.generate_regex(input:list)
3 IN input Python
4 List of Python strings (e.g., node names) (list)
5 Returns:
6 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
7 • regex - Python bytearray containing regular expression representation of the input list
8 (bytearray)
9 See PMIx_generate_regex for details.
```

## 10 A.5.4 Server.generate\_ppn

11 **Summary**  
12 Generate a regular expression representation of the input strings.

```
13 PMIx v4.0 Format Python
14 rc, regex = myserver.generate_ppn(input:list)
15 IN input Python
16 List of Python strings, each string consisting of a comma-delimited list of ranks on each node,
17 with the strings being in the same order as the node names provided to "generate_regex" (list)
18 Returns:
19 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
20 • regex - Python bytearray containing regular expression representation of the input list
21 (bytearray)
22 See PMIx_generate_ppn for details.
```

## 23 A.5.5 Server.generate\_locality\_string

24 **Summary**  
25 Generate a PMIx locality string from a given cpuset.

```
1 Format Python
2 rc, locality = myserver.generate_locality_string(cpuset:dict) Python
3 IN cset
4 cpuset containing the bitmap of assigned PUs (dict)
5 Returns:
6 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
7 • locality - String representation of the PMIx locality corresponding to the input bitmap (string)
8 See PMIx_server_generate_locality_string for details.
```

## 9 A.5.6 Server.generate\_cpuset\_string

```
10 Summary
11 Generate a PMIx string representation of the provided cpuset.
12 PMIx v4.0 Format Python
13 rc, cpustr = myserver.generate_cpuset_string(cpuset:dict) Python
14 IN cset
15 cpuset containing the bitmap of assigned PUs (dict)
16 Returns:
17 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
18 • cpustr - String representation of the input bitmap (string)
19 See PMIx_server_generate_cpuset_string for details.
```

## 20 A.5.7 Server.register\_nspace

```
21 Summary
22 Setup the data about a particular namespace.
```

1           **Format**                                                                  Python

2        rc = myserver.register\_nspace(nspace:str,  
3                                                                                  nlocalprocs:integer,  
4                                                                                          directives:list)

5           **IN nspace**  
6            Python string containing the namespace (str)  
7           **IN nlocalprocs**  
8           Number of local processes (integer)  
9           **IN directives**  
10           List of Python `info` dictionaries (list)

11           Returns:  
12           • `rc` - `PMIx_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)  
13           See `PMIx_server_register_nspace` for description of all relevant attributes and behaviors.

## 14 A.5.8 Server.deregister\_nspace

### 15           **Summary**

16           Deregister a namespace.

17        *PMIx v4.0*           **Format**                                                  Python

18        myserver.deregister\_nspace(nspace:str)

19           **IN nspace**  
20           Python string containing the namespace (str)  
21           Returns: None  
22           See `PMIx_server_deregister_nspace` for details.

## 23 A.5.9 Server.register\_resources

### 24           **Summary**

25           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.

## 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.

## 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, envin:dict)`

Python

15      **IN proc**

16      Python `proc` dictionary identifying the client process (dict)

17      **INOUT envin**

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.

1      **Format**

Python

2      `rc, data = myserver.dmodex_request(proc:dict)`

Python

3      **IN proc**

4      Python **proc** dictionary identifying the process whose data is requested (dict)

5      Returns:

- 6      • **rc** - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
7      • **data** - Python **byteobject** containing the returned data (dict)

8      See [PMIx\\_server\\_dmodex\\_request](#) for details.

9      **A.5.15 Server.setup\_application**

10     **Summary**

11     Function by which the resource manager can request application-specific setup data prior to launch  
12    of a *job*.

13     **Format**

Python

14     `rc, info = myserver.setup_application(nspace:str, directives:list)`

Python

15     **IN nspace**

16     Namespace whose setup information is being requested (str)

17     **IN directives**

18     Python list of **info** directives

19      Returns:

- 20      • **rc** - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
21      • **info** - Python list of **info** dictionaries containing the returned data (list)  
22      See [PMIx\\_server\\_setup\\_application](#) for details.

23      **A.5.16 Server.register\_attributes**

24      **Summary**

25      Register host environment attribute support for a function.

```
1 Format Python
2 rc = myserver.register_attributes(function:str, attrs:list) Python
3 IN function
4 Name of the function (str)
5 IN attrs
6 Python list of regattr describing the supported attributes
7 Returns:
8 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9 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) Python
16 IN nspace
17 Namespace whose setup information is being requested (str)
18 IN info
19 Python list of info containing the setup data (list)
20 Returns:
21 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
22 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.

## 15 A.5.19 Server.collect\_inventory

```
16 Summary
17 Collect inventory of resources on a node.
18 PMIx v4.0 Format  Python
19 rc,info = myserver.collect_inventory(directives:list)
20 IN directives 
21 Optional Python list of info containing directives (list)
22 Returns:
23 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24 • info - Python list of info containing the returned data (list)
25 See PMIx server collect inventory for details.
```

## 26 A.5.20 Server.deliver inventory

**Summary**  
Pass collected inventory to the PMIx server library for storage.

```
1 Format Python
2 rc = myserver.deliver_inventory(info:list, directives:list)
3 IN info
4 - Python list of info dictionaries containing the inventory data (list)
5 IN directives
6 Python list of info dictionaries containing directives (list)
7 Returns:
8 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9 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
14 rc = myserver.define_process_set(members:list, name:str)
15 IN members
16 - List of Python proc dictionaries identifying the processes to be added to the process set
17 (list)
18 IN name
19 - Name of the process set (str)
20 Returns:
21 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
22 See PMIx_server_define_process_set for details.
```

## 23 A.5.22 Server.delete\_process\_set

### 24 Summary

25 Delete a PMIx process set.

**Format** Python  
`rc = myserver.delete_process_set(name:str)`

**IN name** Python  
- Name of the process set (str)

Returns:

- `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx\_server\_delete\_process\_set** for details.

## 8 A.5.23 Server.register resources

## Summary

Register non-namespace related information with the local PMIx server library.

**Format** Python  
`rc = myserver.register_resources(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)

See **PMIx\_server\_register\_resources** for details.

## 18 A.5.24 Server.deregister\_resources

## Summary

Deregister non-namespace related information with the local PMIx server library.

**Format** Python  
`rc = myserver.deregister_resources(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)

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:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `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:

- `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.

**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

## Summary

Establish a connection to a PMIx server.

```
11 PMIx v4.0
12 rc, proc, server = mytool.connect_to_server(info:list)
13
14 IN info
15 List of Python info dictionaries (list)
16
17 Returns:
18
19 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
20
21 • proc - a Python proc containing the tool's identifier (dict)
22
23 • server - a Python proc containing the identifier of the server to which the tool attached (dict)
24
25 See PMIx tool attach to server for details.
```

## 20 A.6.5 Tool.get\_servers

## Summary

Get a list containing the **proc** process identifiers of all servers to which the tool is currently connected.

**Format** Python  
`rc, servers = mytool.get_servers()` Python

Returns:

- *rc* - **PMIx\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *servers* - a list of Python **proc** containing the identifiers of the servers to which the tool is currently attached (dict)

See **PMIx tool get servers** for details.

## 8 A.6.6 Tool.set\_server

## Summary

Designate a server as the tool's *primary* server.

**Format** Python  
`rc = mytool.set_server(proc:dict, info:list)` Python  
**IN proc**  
Python **proc** containing the identifier of the servers to which the tool is to attach (list)  
**IN info**  
List of Python **info** dictionaries (list)  
  
Returns:  

- `rc` - **PMIx\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx tool set server** for details.

## 20 A.6.7 Tool.iof\_pull

## Summary

Register to receive output forwarded from a remote process.

## 16 A.6.8 Tool.iof deregister

## Summary

Deregister from output forwarded from a remote process.

```
19 PMIx v4.0 Format Python
20 rc = mytool.iof_deregister(id:integer, directives:list)
21 IN id
22 PMIx reference identifier returned by pull request (list)
23 IN directives
24 List of Python info dictionaries describing request (list)
25 Returns:
26 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
27 See PMIx_IOF_deregister for description of all relevant attributes and behaviors.
```

## 28 A.6.9 Tool.iof\_push

## Summary

Push data collected locally (typically from stdin) to stdin of target recipients.

```

1 Format Python
2 rc = mytool.iof_push(targets:list, data:dict, directives:list)
3 IN sources Python
4 List of Python proc of target processes (list)
5 IN data
6 Python byteobject containing data to be delivered (dict)
7 IN directives
8 Optional list of Python info describing request (list)
9 Returns:
10 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
11 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.

```

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('nspc': "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.

## Python

```
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

# Revision History

---

## 1 B.1 Version 1.0: June 12, 2015

2 The PMIx version 1.0 *ad hoc* standard was defined in a set of header files as part of the v1.0.0  
3 release of the OpenPMIx library prior to the creation of the formal PMIx 2.0 standard. Below are a  
4 summary listing of the interfaces defined in the 1.0 headers.

5 • Client APIs

- 6   – `PMIx_Init`, `PMIx_Initialized`, `PMIx_Abort`, `PMIx_Finalize`  
7   – `PMIx_Put`, `PMIx_Commit`,  
8   – `PMIx_Fence`, `PMIx_Fence_nb`  
9   – `PMIx_Get`, `PMIx_Get_nb`  
10   – `PMIx_Publish`, `PMIx_Publish_nb`  
11   – `PMIx_Lookup`, `PMIx_Lookup_nb`  
12   – `PMIx_Unpublish`, `PMIx_Unpublish_nb`  
13   – `PMIx_Spawn`, `PMIx_Spawn_nb`  
14   – `PMIx_Connect`, `PMIx_Connect_nb`  
15   – `PMIx_Disconnect`, `PMIx_Disconnect_nb`  
16   – `PMIx_Resolve_nodes`, `PMIx_Resolve_peers`

17 • Server APIs

- 18   – `PMIx_server_init`, `PMIx_server_finalize`  
19   – `PMIx_generate_regex`, `PMIx_generate_ppn`  
20   – `PMIx_server_register_nspace`, `PMIx_server_deregister_nspace`  
21   – `PMIx_server_register_client`, `PMIx_server_deregister_client`  
22   – `PMIx_server_setup_fork`, `PMIx_server_dmodex_request`

23 • Common APIs

- 24   – `PMIx_Get_version`, `PMIx_Store_internal`, `PMIx_Error_string`  
25   – `PMIx_Register_errhandler`, `PMIx_Deregister_errhandler`, `PMIx_Notify_error`

26 The `PMIx_Init` API was subsequently modified in the v1.1.0 release of that library.

## 1 B.2 Version 2.0: Sept. 2018

2 The following APIs were introduced in v2.0 of the PMIx Standard:

3 • Client APIs

- 4   – `PMIx_Query_info_nb`, `PMIx_Log_nb`  
5   – `PMIx_Allocation_request_nb`, `PMIx_Job_control_nb`,  
6    `PMIx_Process_monitor_nb`, `PMIx_Heartbeat`

7 • Server APIs

- 8   – `PMIx_server_setup_application`, `PMIx_server_setup_local_support`

9 • Tool APIs

- 10   – `PMIx_tool_init`, `PMIx_tool_finalize`

11 • Common APIs

- 12   – `PMIx_Register_event_handler`, `PMIx_Deregister_event_handler`  
13   – `PMIx_Notify_event`  
14   – `PMIx_Proc_state_string`, `PMIx_Scope_string`  
15   – `PMIx_Persistence_string`, `PMIx_Data_range_string`  
16   – `PMIx_Info_directives_string`, `PMIx_Data_type_string`  
17   – `PMIx_Alloc_directive_string`  
18   – `PMIx_Data_pack`, `PMIx_Data_unpack`, `PMIx_Data_copy`  
19   – `PMIx_Data_print`, `PMIx_Data_copy_payload`

### 20 B.2.1 Removed/Modified APIs

21 The `PMIx_Init` API was modified in v2.0 of the standard from its *ad hoc* v1.0 signature to  
22 include passing of a `pmix_info_t` array for flexibility and “future-proofing” of the API. In  
23 addition, the `PMIx_Notify_error`, `PMIx_Register_errhandler`, and  
24 `PMIx_Deregister_errhandler` APIs were replaced. This pre-dated official adoption of  
25 PMIx as a Standard.

### 26 B.2.2 Deprecated constants

27 The following constants were deprecated in v2.0:

28   `PMIX_MODEX`  
29   `PMIX_INFO_ARRAY`

## 1 B.2.3 Deprecated attributes

2 The following attributes were deprecated in v2.0:

```
3 PMIX_ERROR_NAME "pmix.errname" (pmix_status_t)
4 Specific error to be notified
5 PMIX_ERROR_GROUP_COMM "pmix.errgroup.comm" (bool)
6 Set true to get comm errors notification
7 PMIX_ERROR_GROUP_ABORT "pmix.errgroup.abort" (bool)
8 Set true to get abort errors notification
9 PMIX_ERROR_GROUP_MIGRATE "pmix.errgroup.migrate" (bool)
10 Set true to get migrate errors notification
11 PMIX_ERROR_GROUP_RESOURCE "pmix.errgroup.resource" (bool)
12 Set true to get resource errors notification
13 PMIX_ERROR_GROUP_SPAWN "pmix.errgroup.spawn" (bool)
14 Set true to get spawn errors notification
15 PMIX_ERROR_GROUP_NODE "pmix.errgroup.node" (bool)
16 Set true to get node status notification
17 PMIX_ERROR_GROUP_LOCAL "pmix.errgroup.local" (bool)
18 Set true to get local errors notification
19 PMIX_ERROR_GROUP_GENERAL "pmix.errgroup.gen" (bool)
20 Set true to get notified of generic errors
21 PMIX_ERROR_HANDLER_ID "pmix.errhandler.id" (int)
22 Errhandler reference id of notification being reported
```

## 23 B.3 Version 2.1: Dec. 2018

24 The v2.1 update includes clarifications and corrections from the v2.0 document, plus addition of  
25 examples:

- 26 • Clarify description of **PMIx\_Connect** and **PMIx\_Disconnect** APIs.
- 27 • Explain that values for the **PMIX\_COLLECTIVE\_ALGO** are environment-dependent
- 28 • Identify the namespace/rank values required for retrieving attribute-associated information using  
29 the **PMIx\_Get** API
- 30 • Provide definitions for *session*, *job*, *application*, and other terms used throughout the document
- 31 • Clarify definitions of **PMIX\_UNIV\_SIZE** versus **PMIX\_JOB\_SIZE**
- 32 • Clarify server module function return values
- 33 • Provide examples of the use of **PMIx\_Get** for retrieval of information
- 34 • Clarify the use of **PMIx\_Get** versus **PMIx\_Query\_info\_nb**
- 35 • Clarify return values for non-blocking APIs and emphasize that callback functions must not be  
36 invoked prior to return from the API
- 37 • Provide detailed example for construction of the **PMIx\_server\_register\_nspace** input  
38 information array

- 1     ● Define information levels (e.g., *session* vs *job*) and associated attributes for both storing and  
2        retrieving values  
3     ● Clarify roles of PMIx server library and host environment for collective operations  
4     ● Clarify definition of **PMIX\_UNIV\_SIZE**

5     

## B.4 Version 2.2: Jan 2019

6     The v2.2 update includes the following clarifications and corrections from the v2.1 document:

- 7     ● Direct modex upcall function (**pmix\_server\_dmodex\_req\_fn\_t**) cannot complete  
8        atomically as the API cannot return the requested information except via the provided callback  
9        function  
10    ● Add missing **pmix\_data\_array\_t** definition and support macros  
11    ● Add a rule divider between implementer and host environment required attributes for clarity  
12    ● Add **PMIX\_QUERY\_QUALIFIERS\_CREATE** macro to simplify creation of **pmix\_query\_t**  
13        qualifiers  
14    ● Add **PMIX\_APP\_INFO\_CREATE** macro to simplify creation of **pmix\_app\_t** directives  
15    ● Add flag and **PMIX\_INFO\_IS\_END** macro for marking and detecting the end of a  
16        **pmix\_info\_t** array  
17    ● Clarify the allowed hierarchical nesting of the **PMIX\_SESSION\_INFO\_ARRAY**,  
18        **PMIX\_JOB\_INFO\_ARRAY**, and associated attributes

19    

## B.5 Version 3.0: Dec. 2018

20    The following APIs were introduced in v3.0 of the PMIx Standard:

- 21    ● Client APIs  
22      – **PMIx\_Log**, **PMIx\_Job\_control**  
23      – **PMIx\_Allocation\_request**, **PMIx\_Process\_monitor**  
24      – **PMIx\_Get\_credential**, **PMIx\_Validate\_credential**  
25    ● Server APIs  
26      – **PMIx\_server\_IOF\_deliver**  
27      – **PMIx\_server\_collect\_inventory**, **PMIx\_server\_deliver\_inventory**  
28    ● Tool APIs  
29      – **PMIx\_IOF\_pull**, **PMIx\_IOF\_push**, **PMIx\_IOF\_deregister**  
30      – **PMIx\_tool\_connect\_to\_server**  
31    ● Common APIs  
32      – **PMIx\_IOF\_channel\_string**

1 The document added a chapter on security credentials, a new section for IO forwarding to the  
2 Process Management chapter, and a few blocking forms of previously-existing non-blocking APIs.  
3 Attributes supporting the new APIs were introduced, as well as additional attributes for a few  
4 existing functions.

## 5 **B.5.1 Removed constants**

6 The following constants were removed in v3.0:

7 **PMIX\_MODEX**  
8 **PMIX\_INFO\_ARRAY**

## 9 **B.5.2 Deprecated attributes**

10 The following attributes were deprecated in v3.0:

11 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)  
12 If **true**, indicates that the requested choice of algorithm is mandatory.

## 13 **B.5.3 Removed attributes**

14 The following attributes were removed in v3.0:

15 **PMIX\_ERROR\_NAME** "pmix.errname" (pmix\_status\_t)  
16 Specific error to be notified  
17 **PMIX\_ERROR\_GROUP\_COMM** "pmix.errgroup.comm" (bool)  
18 Set true to get comm errors notification  
19 **PMIX\_ERROR\_GROUP\_ABORT** "pmix.errgroup.abort" (bool)  
20 Set true to get abort errors notification  
21 **PMIX\_ERROR\_GROUP\_MIGRATE** "pmix.errgroup.migrate" (bool)  
22 Set true to get migrate errors notification  
23 **PMIX\_ERROR\_GROUP\_RESOURCE** "pmix.errgroup.resource" (bool)  
24 Set true to get resource errors notification  
25 **PMIX\_ERROR\_GROUP\_SPAWN** "pmix.errgroup.spawn" (bool)  
26 Set true to get spawn errors notification  
27 **PMIX\_ERROR\_GROUP\_NODE** "pmix.errgroup.node" (bool)  
28 Set true to get node status notification  
29 **PMIX\_ERROR\_GROUP\_LOCAL** "pmix.errgroup.local" (bool)  
30 Set true to get local errors notification  
31 **PMIX\_ERROR\_GROUP\_GENERAL** "pmix.errgroup.gen" (bool)  
32 Set true to get notified of generic errors  
33 **PMIX\_ERROR\_HANDLER\_ID** "pmix.errhandler.id" (int)  
34 Errhandler reference id of notification being reported

## 1 B.6 Version 3.1: Jan. 2019

2 The v3.1 update includes clarifications and corrections from the v3.0 document:

- 3 • Direct modex upcall function (`pmix_server_dmodex_req_fn_t`) cannot complete  
4 atomically as the API cannot return the requested information except via the provided callback  
5 function
- 6 • Fix typo in name of `PMIX_FWD_STDDIAG` attribute
- 7 • Correctly identify the information retrieval and storage attributes as “new” to v3 of the standard
- 8 • Add missing `pmix_data_array_t` definition and support macros
- 9 • Add a rule divider between implementer and host environment required attributes for clarity
- 10 • Add `PMIX_QUERY_QUALIFIERS_CREATE` macro to simplify creation of `pmix_query_t`  
11 qualifiers
- 12 • Add `PMIX_APP_INFO_CREATE` macro to simplify creation of `pmix_app_t` directives
- 13 • Add new attributes to specify the level of information being requested where ambiguity may exist  
(see 6.1)
- 14 • Add new attributes to assemble information by its level for storage where ambiguity may exist  
(see 16.2.3.1)
- 15 • Add flag and `PMIX_INFO_IS_END` macro for marking and detecting the end of a  
`pmix_info_t` array
- 16 • Clarify that `PMIX_NUM_SLOTS` is duplicative of (a) `PMIX_UNIV_SIZE` when used at the  
`session` level and (b) `PMIX_MAX_PROCS` when used at the `job` and `application` levels, but leave  
it in for backward compatibility.
- 17 • Clarify difference between `PMIX_JOB_SIZE` and `PMIX_MAX_PROCS`
- 18 • Clarify that `PMIx_server_setup_application` must be called per-`job` instead of  
per-`application` as the name implies. Unfortunately, this is a historical artifact. Note that both  
`PMIX_NODE_MAP` and `PMIX_PROC_MAP` must be included as input in the `info` array provided  
to that function. Further descriptive explanation of the “instant on” procedure will be provided in  
the next version of the PMIx Standard.
- 19 • Clarify how the PMIx server expects data passed to the host by  
`pmix_server_fencenb_fn_t` should be aggregated across nodes, and provide a code  
snippet example

## 31 B.7 Version 3.2: 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.

## B.7.1 Deprecated constants

The following constants were deprecated in v3.2:

```

PMIX_ERR_DATA_VALUE_NOT_FOUND Data value not found
PMIX_ERR_HANDSHAKE_FAILED Connection handshake failed
PMIX_ERR_IN_ERRNO Error defined in errno
PMIX_ERR_INVALID_ARG Invalid argument
PMIX_ERR_INVALID_ARGS Invalid arguments
PMIX_ERR_INVALID_KEY Invalid key
PMIX_ERR_INVALID_KEY_LENGTH Invalid key length
PMIX_ERR_INVALID_KEYVALP Invalid key/value pair
PMIX_ERR_INVALID_LENGTH Invalid argument length
PMIX_ERR_INVALID_NAMESPACE Invalid namespace
PMIX_ERR_INVALID_NUM_ARGS Invalid number of arguments
PMIX_ERR_INVALID_NUM_PARSED Invalid number parsed
PMIX_ERR_INVALID_SIZE Invalid size
PMIX_ERR_INVALID_VAL Invalid value
PMIX_ERR_INVALID_VAL_LENGTH Invalid value length
PMIX_ERR_NOT_IMPLEMENTED Not implemented
PMIX_ERR_PACK_MISMATCH Pack mismatch
PMIX_ERR_PROC_ENTRY_NOT_FOUND Process not found
PMIX_ERR_PROC_REQUESTED_ABORT Process is already requested to abort
PMIX_ERR_READY_FOR_HANDSHAKE Ready for handshake
PMIX_ERR_SERVER_FAILED_REQUEST Failed to connect to the server
PMIX_ERR_SERVER_NOT_AVAIL Server is not available
PMIX_ERR_SILENT Silent error
PMIX_GDS_ACTION_COMPLETE The Global Data Storage (GDS) action has completed
PMIX_NOTIFY_ALLOC_COMPLETE Notify that a requested allocation operation is complete
 - the result of the request will be included in the info array

```

## 1 B.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 B.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  
**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**,  
**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**,  
**PMIx\_Process\_monitor**, **PMIx\_Get\_credential**, and  
**PMIx\_Validate\_credential** to allow return of results
- 34 • Update description to provide an option for blocking behavior of the  
**PMIx\_Register\_event\_handler**, **PMIx\_Deregister\_event\_handler**,  
**PMIx\_Notify\_event**, **PMIx\_IOF\_pull**, **PMIx\_IOF\_deregister**, and  
**PMIx\_IOF\_push** APIs. The need for blocking forms of these functions was not initially

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
4 • Tool APIs
5 - PMIx_tool_disconnect
6 - PMIx_tool_set_server
7 - PMIx_tool_attach_to_server
8 - PMIx_tool_get_servers
9
10 • Data types
11 - pmix_regattr_t
12 - pmix_cpuset_t
13 - pmix_topology_t
14 - pmix_locality_t
15 - pmix_bind_envelope_t
16 - pmix_group_opt_t
17 - pmix_group_operation_t
18 - pmix_fabric_t
19 - pmix_device_distance_t
20 - pmix_coord_t
21 - pmix_coord_view_t
22 - pmix_geometry_t
23 - pmix_link_state_t
24 - pmix_job_state_t
25 - pmix_device_type_t
26
27 • Callback functions
28 - pmix_device_dist_cfunc_t
```

## 26 B.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
16

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
```

## Events

```
15 PMIX_EVENT_SYS_BASE
16 PMIX_EVENT_NODE_DOWN
17 PMIX_EVENT_NODE_OFFLINE
18 PMIX_EVENT_SYS_OTHER
19
20
```

## B.8.2 Added Attributes

### Sync-Access attributes

```
22 PMIX_COLLECT_GENERATED_JOB_INFO "pmix.collect.gen" (bool)
```

23 Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx  
24 servers. Some job-level information (e.g., distance between processes and fabric devices) is  
25 best determined on a distributed basis as it primarily pertains to local processes. Should  
26 remote processes need to access the information, it can either be obtained collectively using  
27 the **PMIx\_Fence** operation with this directive, or can be retrieved one peer at a time using  
28 **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.ptrs" (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\_NSPACE** 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_NSPACE 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 it's 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_nspace` API, there is no requirement that the array  
6      contain either the `PMIX_NSPACE` 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_NSPACE` or `PMIX_JOBID` attributes of the `job` containing the application, plus its  
12     `PMIX_APPNUM` attribute, must be included in the array when the array is *not* included as  
13     part of a call to `PMIx_server_register_nspace` - 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_NSPACE` 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_nspace` - 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
5 PMIX_HOSTNAME_ALIASES "pmix.alias" (char*)
6 Comma-delimited list of names by which the target node is known.
7
8 PMIX_HOSTNAME_KEEP_FQDN "pmix.fqdn" (bool)
9 FQDNs are being retained by the PMIx library.
10
11 PMIX_CPUSET_BITMAP "pmix.bitmap" (pmix_cpuset_t*)
12 Bitmap applied to the process upon launch.
13
14 PMIX_EXTERNAL_PROGRESS "pmix.evext" (bool)
15 The host shall progress the PMIx library via calls to PMIx_Progress
16
17 PMIX_NODE_MAP_RAW "pmix.nmap.raw" (char*)
18 Comma-delimited list of nodes containing procs within the specified realm. Defaults to the
19 job realm.
20
21 PMIX_PROC_MAP_RAW "pmix.pmap.raw" (char*)
22 Semi-colon delimited list of strings, each string containing a comma-delimited list of ranks
23 on the corresponding node within the specified realm. Defaults to the job realm.
24
25 Tool attributes
26 PMIX_TOOL_CONNECT_OPTIONAL "pmix.tool.conopt" (bool)
27 The tool shall connect to a server if available, but otherwise continue to operate
28 unconnected.
29
30 PMIX_TOOL_ATTACHMENT_FILE "pmix.tool.attach" (char*)
31 Pathname of file containing connection information to be used for attaching to a specific
32 server.
33
34 PMIX_LAUNCHER_RENDEZVOUS_FILE "pmix.tool.lncrnd" (char*)
35 Pathname of file where the launcher is to store its connection information so that the
36 spawning tool can connect to it.
37
38 PMIX_PRIMARY_SERVER "pmix.pri.srvr" (bool)
39 The server to which the tool is connecting shall be designated the primary server once
40 connection has been accomplished.
41
42 PMIX_NOHUP "pmix.nohup" (bool)
43 Any processes started on behalf of the calling tool (or the specified namespace, if such
44 specification is included in the list of attributes) should continue after the tool disconnects
45 from its server.
46
47 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 17.4.3 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 17.4.3 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.grp" (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.

11 Device attributes
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 UUID or node-local OS name of a particular device.

20 Sets-Groups attributes
21 PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
22 Return the number of process sets defined in the specified range (defaults to
23 PMIX_RANGE_SESSION).

24 PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)
25 Return a pmix_data_array_t containing an array of strings of the process set names
26 defined in the specified range (defaults to PMIX_RANGE_SESSION).

27 PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t*)
28 Return an array of pmix_proc_t containing the members of the specified process set.

29 PMIX_PSET_NAME "pmix.pset.nm" (char*)
30 The name of the newly defined process set.

31 PMIX_PSET_MEMBERS "pmix.pset.mems" (pmix_data_array_t*)
32 An array of pmix_proc_t containing the members of the newly defined process set.

33 PMIX_PSET_NAMES "pmix.pset.nms" (pmix_data_array_t*)
34 Returns an array of char* string names of the process sets in which the given process is a
35 member.

36 PMIX_QUERY_NUM_GROUPS "pmix.qry.pgrpnum" (size_t)
37 Return the number of process groups defined in the specified range (defaults to session).
38 OPTIONAL QUALIFERS: PMIX_RANGE.

```

```

1 PMIX_QUERY_GROUP_NAMES "pmixqry.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
6 PMIX_QUERY_GROUP_MEMBERSHIP
7 "pmixqry.pgrpmems" (pmix_data_array_t*)
8 Return a pmix_data_array_t of pmix_proc_t containing the members of the
9 specified process group. REQUIRED QUALIFIERS: PMIX_GROUP_ID.
10
11 PMIX_GROUP_ID "pmix.grp.id" (char*)
12 User-provided group identifier - as the group identifier may be used in PMIx operations, the
13 user is required to ensure that the provided ID is unique within the scope of the host
14 environment (e.g., by including some user-specific or application-specific prefix or suffix to
15 the string).
16
17 PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)
18 This process is the leader of the group.
19
20 PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)
21 Participation is optional - do not return an error if any of the specified processes terminate
22 without having joined. The default is false.
23
24 PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)
25 Notify remaining members when another member terminates without first leaving the group.
26
27 PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)
28 Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective
29 operation.
30
31 PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)
32 Requests that the RM assign a new context identifier to the newly created group. The
33 identifier is an unsigned, size_t value that the RM guarantees to be unique across the range
34 specified in the request. Thus, the value serves as a means of identifying the group within
35 that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.
36
37 PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)
38 Group operation only involves local processes. PMIx implementations are required to
39 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.
40
41 PMIX_GROUP_CONTEXT_ID "pmix.grp.ctxid" (size_t)
42 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.

7 Process Mgmt attributes
8 PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
9 Direct output into files of form "<directory>/<jobid>/rank.<rank>/
10 stdout[err]" - can be assigned to the entire job (by including attribute in the job_info
11 array) or on a per-application basis in the info array for each pmix_app_t.
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 12.4.3.
4 PMIX_LOG_COMPLETION "pmix.logcomp" (bool)
5 Requests that the launcher log the PMIX_EVENT_JOB_END event for normal or abnormal
6 termination of the spawned job using PMIx_Log, subject to the logging attributes of
7 Section 12.4.3. The event shall include the returned status code
8 (PMIX_JOB_TERM_STATUS) for the corresponding job; the identity (PMIX_PROCID)
9 and exit status (PMIX_EXIT_CODE) of the first failed process, if applicable; and a
10 PMIX_EVENT_TIMESTAMP indicating the time the termination occurred.
```

```
11 PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
12 Ensure the given value appears first in the specified envar using the separator character,
13 creating the envar if it doesn't already exist
```

#### 14 Event attributes

```
15 PMIX_EVENT_TIMESTAMP "pmix.evtstamp" (time_t)
16 System time when the associated event occurred.
```

### 17 B.8.3 Added Environmental Variables

```
18 Tool environmental variables
19 PMIX_LAUNCHER_RNDZ_URI
20 PMIX_LAUNCHER_RNDZ_FILE
21 PMIX_KEEPALIVE_PIPE
```

### 23 B.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 B.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 B.8.6 Deprecated constants

2 The following constants were deprecated in v4.0:

```
3 PMIX_ERR_DEBUGGER_RELEASE Renamed to PMIX_DEBUGGER_RELEASE
4 PMIX_ERR_JOB_TERMINATED Renamed to PMIX_EVENT_JOB_END
5 PMIX_EXISTS Renamed to PMIX_ERR_EXISTS
6 PMIX_ERR_PROC_ABORTED Consolidated with PMIX_EVENT_PROC_TERMINATED
7 PMIX_ERR_PROC_ABORTING Consolidated with PMIX_EVENT_PROC_TERMINATED
8 PMIX_ERR_LOST_CONNECTION_TO_SERVER Consolidated into
9 PMIX_ERR_LOST_CONNECTION
10 PMIX_ERR_LOST_PEER_CONNECTION Consolidated into
11 PMIX_ERR_LOST_CONNECTION
12 PMIX_ERR_LOST_CONNECTION_TO_CLIENT Consolidated into
13 PMIX_ERR_LOST_CONNECTION
14 PMIX_ERR_INVALID_TERMINATION Renamed to PMIX_ERR_JOB_TERM_WO_SYNC
15 PMIX_PROC_TERMINATED Renamed to PMIX_EVENT_PROC_TERMINATED
16 PMIX_ERR_NODE_DOWN Renamed to PMIX_EVENT_NODE_DOWN
17 PMIX_ERR_NODE_OFFLINE Renamed to PMIX_EVENT_NODE_OFFLINE
18 PMIX_ERR_SYS_OTHER Renamed to PMIX_EVENT_SYS_OTHER
19 PMIX_CONNECT_REQUESTED Connection has been requested by a PMIx-based tool -
20 deprecated as not required.
21 PMIX_PROC_HAS_CONNECTED A tool or client has connected to the PMIx server -
22 deprecated in favor of the new pmix_server_client_connected2_fn_t server
23 module API
```

## 24 B.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 PMIX_ERR_HANDSHAKE_FAILED Connection handshake failed
28 PMIX_ERR_READY_FOR_HANDSHAKE Ready for handshake
29 PMIX_ERR_IN_ERRNO Error defined in errno
30 PMIX_ERR_INVALID_VAL_LENGTH Invalid value length
31 PMIX_ERR_INVALID_LENGTH Invalid argument length
32 PMIX_ERR_INVALID_NUM_ARGS Invalid number of arguments
33 PMIX_ERR_INVALID_ARGS Invalid arguments
34 PMIX_ERR_INVALID_NUM_PARSED Invalid number parsed
35 PMIX_ERR_INVALID_KEYVALP Invalid key/value pair
36 PMIX_ERR_INVALID_SIZE Invalid size
37 PMIX_ERR_PROC_REQUESTED_ABORT Process is already requested to abort
38 PMIX_ERR_SERVER_FAILED_REQUEST Failed to connect to the server
39 PMIX_ERR_PROC_ENTRY_NOT_FOUND Process not found
```

```

1 PMIX_ERR_INVALID_ARG Invalid argument
2 PMIX_ERR_INVALID_KEY Invalid key
3 PMIX_ERR_INVALID_KEY_LENGTH Invalid key length
4 PMIX_ERR_INVALID_VAL Invalid value
5 PMIX_ERR_INVALID_NAMESPACE Invalid namespace
6 PMIX_ERR_SERVER_NOT_AVAIL Server is not available
7 PMIX_ERR_SILENT Silent error
8 PMIX_ERR_PACK_MISMATCH Pack mismatch
9 PMIX_ERR_DATA_VALUE_NOT_FOUND Data value not found
10 PMIX_ERR_NOT_IMPLEMENTED Not implemented
11 PMIX_GDS_ACTION_COMPLETE The GDS action has completed
12 PMIX_NOTIFY_ALLOC_COMPLETE Notify that a requested allocation operation is complete
13 - the result of the request will be included in the info array

```

## 14 B.8.8 Deprecated attributes

15 The following attributes were deprecated in v4.0:

```

16 PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
17 Renamed to PMIX_TOPOLOGY2.
18 PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
19 Renamed to PMIX_DEBUG_TARGET
20 PMIX_RECONNECT_SERVER "pmix.tool.recon" (bool)
21 Renamed to the PMIX_TOOL_CONNECT_TO_SERVER API
22 PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)
23 Renamed to PMIX_ALLOC_FABRIC
24 PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
25 Renamed to PMIX_ALLOC_FABRIC_ID
26 PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
27 Renamed to PMIX_ALLOC_FABRIC_QOS
28 PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)
29 Renamed to PMIX_ALLOC_FABRIC_TYPE
30 PMIX_ALLOC_NETWORK_PLANE "pmix.alloc.netplane" (char*)
31 Renamed to PMIX_ALLOC_FABRIC_PLANE
32 PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)
33 Renamed to PMIX_ALLOC_FABRIC_ENDPTS
34 PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
35 Renamed to PMIX_ALLOC_FABRIC_ENDPTS_NODE
36 PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
37 Renamed to PMIX_ALLOC_FABRIC_SEC_KEY
38 PMIX_PROC_DATA "pmix pdata" (pmix_data_array_t)
39 Renamed to PMIX_PROC_INFO_ARRAY
40 PMIX_LOCALITY "pmix.loc" (pmix_locality_t)

```

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       B.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 Spawning 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 **B.9 Version 4.1: Oct. 2021**

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 [18](#) on page [455](#)

### 19 **B.9.1 Added Functions (Provisional)**

- ```

20 • PMIx_Data_load
21 • PMIx_Data_unload
22 • PMIx_Data_compress
23 • PMIx_Data_decompress

```

24 **B.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 **B.9.3 Added Macros (Provisional)**

- ```

30 • PMIX_NSPACE_INVALID
31 • PMIX_RANK_IS_VALID
32 • PMIX_PROCID_INVALID
33 • PMIX_PROCID_XFER

```

1 B.9.4 Added Constants (Provisional)

- 2 • `PMIX_PROC_NSPACE`
- 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 B.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)`

```

1      Types of storage mediums utilized by the storage system (e.g., SSDs, HDDs, tape)
2      PMIX_STORAGE_ACCESSIBILITY
3      "pmix.strg.access" (pmix_storage_accessibility_t)
4          Accessibility level of the storage system (e.g., within same node, within same session)
5      PMIX_STORAGE_PERSISTENCE
6      "pmix.strg.persist" (pmix_storage_persistence_t)
7          Persistence level of the storage system (e.g., scratch storage or archive storage)
8      PMIX_QUERY_STORAGE_LIST "pmix.strg.list" (char*)
9          Comma-delimited list of storage identifiers (i.e., PMIX_STORAGE_ID types) for available
10         storage systems
11     PMIX_STORAGE_CAPACITY_LIMIT "pmix.strg.caplim" (double)
12         Overall limit on capacity (in bytes) for the storage system
13     PMIX_STORAGE_CAPACITY_USED "pmix.strg.capuse" (double)
14         Overall used capacity (in bytes) for the storage system
15     PMIX_STORAGE_OBJECT_LIMIT "pmix.strg.objlim" (uint64_t)
16         Overall limit on number of objects (e.g., inodes) for the storage system
17     PMIX_STORAGE_OBJECTS_USED "pmix.strg.objuse" (uint64_t)
18         Overall used number of objects (e.g., inodes) for the storage system
19     PMIX_STORAGE_MINIMAL_XFER_SIZE "pmix.strg.minxfer" (double)
20         Minimal transfer size (in bytes) for the storage system - this is the storage system's atomic
21         unit of transfer (e.g., block size)
22     PMIX_STORAGE_SUGGESTED_XFER_SIZE "pmix.strg.sxfer" (double)
23         Suggested transfer size (in bytes) for the storage system
24     PMIX_STORAGE_BW_MAX "pmix.strg.bwmax" (double)
25         Maximum bandwidth (in bytes/sec) for storage system - provided as the theoretical
26         maximum or the maximum observed bandwidth value
27     PMIX_STORAGE_BW_CUR "pmix.strg.bwcur" (double)
28         Observed bandwidth (in bytes/sec) for storage system - provided as a recently observed
29         bandwidth value, with the exact measurement interval depending on the storage system
30         and/or PMIx library implementation
31     PMIX_STORAGE_IOPS_MAX "pmix.strg.iopsmax" (double)
32         Maximum IOPS (in I/O operations per second) for storage system - provided as the
33         theoretical maximum or the maximum observed IOPS value
34     PMIX_STORAGE_IOPS_CUR "pmix.strg.iopscur" (double)
35         Observed IOPS (in I/O operations per second) for storage system - provided as a recently
36         observed IOPS value, with the exact measurement interval depending on the storage system
37         and/or PMIx library implementation

```

```
1 PMIX_STORAGE_ACCESS_TYPE
2 "pmix.strg.atype" (pmix_storage_access_type_t)
3 Qualifier describing the type of storage access to return information for (e.g., for qualifying
4 PMIX_STORAGE_BW_CUR, PMIX_STORAGE_IOPS_CUR, or
5 PMIX_STORAGE_SUGGESTED_XFER_SIZE attributes)
```

APPENDIX C

Acknowledgements

This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.

C.1 Version 4.0

The following list includes some of the active participants in the PMIx v4 standardization process.

- Ralph H. Castain and Danielle Sikich
- Joshua Hursey and David Solt
- Dirk Schubert
- John DelSignore
- Aurelien Bouteiller
- Michael A Raymond
- Howard Pritchard and Nathan Hjelm
- Brice Goglin
- Kathryn Mohror and Stephen Herbein
- Thomas Naughton and Swaroop Pophale
- William E. Allcock and Paul Rich
- Michael Karo
- Artem Polyakov

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- Allinea (ARM)

- Perforce
- University of Tennessee, Knoxville
- The Exascale Computing Project, an initiative of the US Department of Energy
- National Science Foundation
- HPE Co.
- Los Alamos National Laboratory
- INRIA
- Lawrence Livermore National Laboratory
- Oak Ridge National Laboratory
- Argonne National Laboratory
- Altair
- NVIDIA

C.2 Version 3.0

The following list includes some of the active participants in the PMIx v3 standardization process.

- Ralph H. Castain, Andrew Friedley, Brandon Yates
- Joshua Hursey and David Solt
- Aurelien Bouteiller and George Bosilca
- Dirk Schubert
- Kevin Harms
- Artem Polyakov

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- The Exascale Computing Project, an initiative of the US Department of Energy
- National Science Foundation
- Argonne National Laboratory

- 1 • Allinea (ARM)
2 • NVIDIA

3 **C.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 Gouaillardet
- 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.

- Research Organization for Information Science and Technology
- HPE Co.
- Allinea (ARM)
- SchedMD, Inc.
- Fujitsu Limited
- Los Alamos National Laboratory
- Adaptive Solutions, Inc.
- INRIA
- Oak Ridge National Laboratory
- Lawrence Livermore National Laboratory
- Sandia National Laboratory
- Altair

C.4 Version 1.0

The following list includes some of the active participants in the PMIx v1 standardization process.

- Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- Joshua Hursey and David Solt
- Aurelien Bouteiller and George Bosilca
- Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- Gilles Gouaillardet
- Gary Brown
- Moe Jette

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- Mellanox, Inc.
- Research Organization for Information Science and Technology

- 1 ● Adaptive Solutions, Inc.
- 2 ● SchedMD, Inc.

Bibliography

- [1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMIx: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.
- [2] Balaji P. et al. PMI: A scalable parallel process-management interface for extreme-scale systems. In *Recent Advances in the Message Passing Interface*, EuroMPI '10, pages 31–41, Berlin, Heidelberg, 2010. Springer.

Index

General terms and other items not induced in the other indices.

application, [6](#), [95](#), [107](#), [307](#), [309](#), [313](#), [531](#), [534](#)

attribute, [8](#)

client, [7](#), [59](#)

clients, [7](#)

clone, [7](#)

clones, [7](#), [68](#), [71](#), [183](#), [185](#), [186](#), [188](#), [211](#), [214](#), [542](#)

data realm, [98](#), [272](#), [273](#)

data realms, [98](#)

device, [8](#)

devices, [8](#)

Direct Modex, [260](#), [323](#)

fabric, [7](#)

fabric device, [8](#)

fabric devices, [8](#)

fabric plane, [8](#), [168](#), [173](#), [203](#), [206](#), [207](#), [277](#), [326](#)

fabric planes, [8](#)

fabrics, [7](#)

host environment, [7](#)

instant on, [8](#), [112](#), [259](#)

job, [6](#), [7](#), [95](#), [100–104](#), [107](#), [297–301](#), [303](#), [307–309](#), [311](#), [313](#), [324](#), [326](#), [327](#), [514](#), [531](#), [532](#), [534](#),
[545–547](#)

key, [8](#)

namespace, [6](#)

node, [7](#), [95](#), [107](#), [168](#), [173](#), [203](#), [206](#), [207](#), [307](#), [326](#)

package, [7](#), [105](#), [305](#), [546](#)

peer, [7](#), [106](#), [303](#)

peers, [7](#)

process, [7](#), [95](#), [107](#), [168](#), [173](#), [203](#), [206](#), [207](#), [307](#), [326](#)

processing unit, [7](#)
rank, [7](#), [313](#)
realm, [98](#)
realms, [98](#)
resource manager, [7](#)
RM, [7](#)
scheduler, [7](#), [275](#)
session, [6](#), [95](#), [100](#), [107](#), [108](#), [297](#), [298](#), [307](#), [308](#), [531](#), [532](#), [534](#), [545](#)
thread, [7](#)
threads, [7](#)
workflow, [7](#)
workflows, [7](#), [395](#)

Index of APIs

PMIx_Abort, 25, [162](#), 163, 348, 350, 469, 484, 529
 PMIxClient.abort (Python), 483
PMIx_Alloc_directive_string, [56](#), 504, 530
 PMIxClient.alloc_directive_string (Python), [504](#)
PMIx_Allocation_request, [96](#), [200](#), 200, 206, 492, 532, 534, 537
 PMIxClient.allocation_request (Python), 491
PMIx_Allocation_request_nb, [203](#), 206, 208, 530
PMIx_Commit, 68, [70](#), 71, 113, [115](#), 115, 116, 323, 324, 351, 355, 485, 529
 PMIxClient.commit (Python), 485
PMIx_Compute_distances, [194](#), 196, 501, 538
 PMIxClient.compute_distances (Python), 501
PMIx_Compute_distances_nb, [195](#), 538
PMIx_Connect, [182](#), 183, 185, 186, 188, 232–234, 438, 439, 489, 529, 531
 PMIxClient.connect (Python), 488
PMIx_Connect_nb, [184](#), 184, 529
pmix_connection_cbfunc_t, [376](#), 376
pmix_credential_cbfunc_t, [283](#), 393, [394](#)
PMIx_Data_compress, [159](#), 161, 559
PMIx_Data_copy, [156](#), 530
PMIx_Data_copy_payload, [157](#), 530
PMIx_Data_decompress, [160](#), 559
PMIx_Data_load, [158](#), 559
PMIx_Data_pack, [152](#), 153, 295, 530
PMIx_Data_print, [156](#), 530
PMIx_Data_range_string, [56](#), 503, 530
 PMIxClient.data_range_string (Python), 503
PMIx_Data_type_string, [56](#), 504, 530
 PMIxClient.data_type_string (Python), 504
PMIx_Data_unload, [159](#), 559
PMIx_Data_unpack, [154](#), 159, 530
PMIx_Deregister_event_handler, [144](#), 497, 530, 537
 PMIxClient.deregister_event_handler (Python), 497
pmix_device_dist_cbfunc_t, [196](#), 196, 539
PMIx_Device_type_string, [58](#), 507, 538
 PMIxClient.device_type_string (Python), 507
PMIx_Disconnect, [185](#), 186–188, 234, 438, 439, 489, 529, 531
 PMIxClient.disconnect (Python), 489
PMIx_Disconnect_nb, [187](#), 188, 234, 529

pmix_dmodex_response_fn_t, 322, 323
PMIx_Error_string, 55, 502, 529
 PMIxClient.error_string (Python), 501
pmix_event_notification_cbfunc_fn_t, 143, 149, 149
PMIx_Fabric_deregister, 279, 280, 499, 538
 PMIxClient.fabric_deregister (Python), 499
PMIx_Fabric_deregister_nb, 280, 538
PMIx_Fabric_register, 268, 276, 278, 498, 538
 PMIxClient.fabric_register (Python), 498
PMIx_Fabric_register_nb, 277, 538
PMIx_Fabric_update, 277, 278, 279, 499, 538
 PMIxClient.fabric_update (Python), 498
PMIx_Fabric_update_nb, 279, 538
PMIx_Fence, 4, 67, 68, 69, 71, 112, 183, 186, 231, 240, 244, 260, 293, 323, 350, 352, 470, 486, 529, 542
 PMIxClient.fence (Python), 485
PMIx_Fence_nb, 53, 69, 350, 352, 470, 529, 537
PMIx_Finalize, 25, 63, 65, 65, 181, 347, 348, 438, 439, 469, 483, 529
 PMIxClient.finalize (Python), 483
PMIx_generate_ppn, 296, 509, 529, 535
 PMIxServer.generate_ppn (Python), 509
PMIx_generate_regex, 294, 296, 308, 509, 529, 535
 PMIxServer.generate_regex (Python), 508
PMIx_Get, 3, 8, 28, 63, 68, 71, 72, 73, 74, 76–78, 81, 82, 85–87, 90, 95, 96, 98–101, 103–105, 107, 108, 112, 116, 165, 166, 171, 172, 174–176, 190, 192, 193, 206, 215, 221, 228, 230, 232, 236, 240, 260, 262, 272, 273, 275, 300, 304, 308, 338, 364, 366, 414, 423–425, 439, 457, 486, 529, 531, 536, 542, 543, 549, 550, 559
 PMIxClient.get (Python), 486
PMIx_Get_attribute_name, 57, 506, 538
 PMIxClient.get_attribute_name (Python), 506
PMIx_Get_attribute_string, 57, 506, 538
 PMIxClient.get_attribute_string (Python), 505
PMIx_Get_cpuset, 193, 500, 538
 PMIxClient.get_cpuset (Python), 500
PMIx_Get_credential, 282, 284, 394, 493, 532, 537
 PMIxClient.get_credential (Python), 493
PMIx_Get_credential_nb, 283
PMIx_Get_nb, 53, 74, 529
PMIx_Get_relative_locality, 190, 192, 304, 339, 500, 538, 558
 PMIxClient.get_relative_locality (Python), 500
PMIx_Get_version, 10, 60, 483, 529
 PMIxClient.get_version (Python), 483
PMIx_Group_construct, 231, 232, 237, 239, 240, 243, 494, 538

PMIxClient.group_construct (Python), 494
 PMIx_Group_construct_nb, 240, 243, 538
 PMIx_Group_destruct, 234, 243, 244, 246, 256, 496, 538
 PMIxClient.group_destruct (Python), 496
 PMIx_Group_destruct_nb, 244, 246, 538
 PMIx_Group_invite, 233, 246, 248, 249, 251, 495, 538
 PMIxClient.group_invite (Python), 494
 PMIx_Group_invite_nb, 249, 538
 PMIx_Group_join, 233, 248, 249, 251, 252, 253–255, 495, 538
 PMIxClient.group_join (Python), 495
 PMIx_Group_join_nb, 251, 254, 255, 538
 PMIx_Group_leave, 234, 255, 256–258, 496, 538
 PMIxClient.group_leave (Python), 496
 PMIx_Group_leave_nb, 257, 538
pmix_hdlr_reg_cfunc_t, 54, 138, 450, 451, 555
pmix_info_cfunc_t, 52, 54, 54, 85, 195, 204, 212, 214, 219, 220, 241, 250, 254, 336, 377, 385,
 387, 388, 390, 391, 404, 407
PMIx_Info_directives_string, 56, 504, 530
 PMIxClient.info_directives_string (Python), 503
PMIx_Init, 7, 59, 60, 63, 82, 87, 104, 346, 418, 422, 432, 433, 440, 482, 530, 536, 559
 PMIxClient.init (Python), 482
PMIx_Initialized, 59, 482, 529
 PMIxClient.initialized (Python), 482
pmix_ifc_cfunc_t, 401, 449, 467
 iofcfunc (Python), 467
PMIx_IOF_channel_string, 57, 505, 532
 PMIxClient.iof_channel_string (Python), 505
PMIx_IOF_deregister, 451, 522, 532, 537
 PMIxTool.iof_deregister (Python), 522
PMIx_IOF_pull, 365, 381, 416, 417, 421, 424, 425, 427, 449, 451, 522, 532, 537
 PMIxTool.iof_pull (Python), 521
PMIx_IOF_push, 365, 381, 416, 421, 424, 427–429, 431, 452, 454, 523, 532, 537, 548
 PMIxTool.iof_push (Python), 522
PMIx_Job_control, 200, 208, 210, 213–215, 389, 434, 492, 532, 537
 PMIxClient.job_ctrl (Python), 492
PMIx_Job_control_nb, 78, 208, 211, 306, 530
PMIx_Job_state_string, 57, 505, 538
 PMIxClient.job_state_string (Python), 505
PMIx_Link_state_string, 57, 506, 538
 PMIxClient.link_state_string (Python), 506
PMIx_Load_topology, 189, 499, 538, 558
 PMIxClient.load_topology (Python), 499
PMIx_Log, 177, 222, 224, 228, 418, 437, 491, 532, 555

PMIxClient.log (Python), 491
PMIx_Log_nb, 225, 228, 530
PMIx_Lookup, 118, 123, 125–127, 486, 487, 529
 PMIxClient.lookup (Python), 487
pmix_lookup_cbfnc_t, 130, 130, 358
PMIx_Lookup_nb, 125, 130, 529
pmix_modex_cbfnc_t, 52, 350, 353, 353, 354
pmix_notification_fn_t, 138, 142, 142, 468
 evhandler (Python), 467
PMIx_Notify_event, 145, 375, 498, 530, 537
 PMIxClient.notify_event (Python), 497
pmix_op_cbfnc_t, 53, 53, 120, 133, 145, 146, 149, 184, 187, 225, 245, 257, 278–280, 297, 317,
 318, 320, 321, 328, 334, 335, 337, 345–347, 349, 356, 360, 367, 369, 371, 373, 374, 383,
 399, 402, 452
PMIx_Parse_cpuset_string, 192, 339, 501, 538
 PMIxClient.parse_cpuset_string (Python), 500
PMIx_Persistence_string, 56, 503, 530
 PMIxClient.persistence_string (Python), 503
PMIx_Proc_state_string, 55, 502, 530
 PMIxClient.proc_state_string (Python), 502
PMIx_Process_monitor, 200, 216, 220, 493, 532, 537
 PMIxClient.monitor (Python), 492
PMIx_Process_monitor_nb, 218, 221, 530
PMIx_Progress, 62, 65, 292, 294, 444, 507, 538, 547
 PMIxClient.progress (Python), 507
PMIx_Publish, 118, 120–123, 357, 487, 529
 PMIxClient.publish (Python), 486
PMIx_Publish_nb, 120, 123, 529
PMIx_Put, 28, 67–71, 95, 98, 113, 113–116, 181, 240, 249, 323, 324, 351, 355, 485, 529
 PMIxClient.put (Python), 484
PMIx_Query_info, 8, 80, 85, 89, 90, 92, 95, 230, 232, 272, 273, 408, 412, 432, 433, 457, 491, 543
 PMIxClient.query (Python), 490
PMIx_Query_info_nb, 78, 85, 85, 96, 181, 308, 329, 530, 531
PMIx_Register_attributes, 328, 515, 537, 538
 PMIxServer.register_attributes (Python), 514
PMIx_Register_event_handler, 78, 137, 177, 418, 438, 497, 530, 537, 554
 PMIxClient.register_event_handler (Python), 496
pmix_release_cbfnc_t, 52, 52
PMIx_Resolve_nodes, 79, 490, 529
 PMIxClient.resolve_nodes (Python), 490
PMIx_Resolve_peers, 79, 106, 303, 490, 529
 PMIxClient.resolve_peers (Python), 489
PMIx_Scope_string, 55, 502, 530

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, 53, 281, 320, 344, 345, 345–347, 469, 538, 555, 556
 clientconnected2 (Python), 468
pmix_server_client_finalized_fn_t, 347, 348, 469
 clientfinalized (Python), 469
PMIx_server_collect_inventory, 336, 338, 516, 532
 PMIxServer.collect_inventory (Python), 516
pmix_server_connect_fn_t, 181, 367, 368, 370, 473
 connect (Python), 473
PMIx_server_define_process_set, 230, 341, 517, 538
 PMIxServer.define_process_set (Python), 517
PMIx_server_delete_process_set, 230, 342, 518, 538
 PMIxServer.delete_process_set (Python), 517
PMIx_server_deliver_inventory, 337, 517, 532
 PMIxServer.deliver_inventory (Python), 516
PMIx_server_deregister_client, 321, 513, 529
 PMIxServer.deregister_client (Python), 513
pmix_server_deregister_events_fn_t, 372, 475
 deregister_events (Python), 475
PMIx_server_deregister_nspace, 317, 321, 511, 529
 PMIxServer.deregister_nspace (Python), 511
PMIx_server_deregister_resources, 318, 512, 518, 539
 PMIxServer.deregister_resources (Python), 512, 518
pmix_server_disconnect_fn_t, 368, 370, 474
 disconnect (Python), 474
pmix_server_dmodex_req_fn_t, 107, 116, 117, 353, 353, 471, 532, 534
 dmodex (Python), 471
PMIx_server_dmodex_request, 322, 323, 324, 514, 529
 PMIxServer.dmodex_request (Python), 513
pmix_server_fabric_fn_t, 268, 275, 406, 481, 538
 fabric (Python), 481
pmix_server_fencenb_fn_t, 350, 352, 353, 470, 534
 fence (Python), 470
PMIx_server_finalize, 293, 508, 529
 PMIxServer.finalize (Python), 508
PMIx_server_generate_cpuset_string, 193, 339, 510, 539
 PMIxServer.generate_cpuset_string (Python), 510
PMIx_server_generate_locality_string, 189, 190, 338, 510, 538
 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](#), [538](#)
 group (Python), [480](#)

PMIx_server_init, [59](#), [289](#), [293](#), [329](#), [343](#), [409](#), [410](#), [414](#), [508](#), [529](#), [538](#)
 PMIxServer.init (Python), [507](#)

PMIx_server_IOF_deliver, [335](#), [426](#), [516](#), [532](#)
 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](#), [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](#), [320](#), [346](#), [348](#), [512](#), [529](#)
 PMIxServer.register_client (Python), [512](#)

pmix_server_register_events_fn_t, [370](#), [474](#)
 register_events (Python), [474](#)

PMIx_server_register_nspace, [10](#), [53](#), [295](#), [296](#), [297](#), [298](#), [307](#), [308](#), [311](#), [318](#), [335](#), [339](#), [511](#), [529](#), [531](#), [545](#)
 PMIxServer.register_nspace (Python), [510](#)

PMIx_server_register_resources, [299](#), [302](#), [303](#), [317](#), [512](#), [518](#), [539](#)
 PMIxServer.register_resources (Python), [511](#), [518](#)

PMIx_server_setup_application, [324](#), [327](#), [328](#), [334](#), [338](#), [514](#), [530](#), [534](#)
 PMIxServer.setup_application (Python), [514](#)

PMIx_server_setup_fork, [321](#), [513](#), [529](#)
 PMIxServer.setup_fork (Python), [513](#)

PMIx_server_setup_local_support, [333](#), [515](#), [530](#)
 PMIxServer.setup_local_support (Python), [515](#)

pmix_server_spawn_fn_t, [180](#), [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](#), [327](#)

PMIx_Spawn, [102](#), [104](#), [105](#), [163](#), [163](#), [169](#), [174](#), [178](#), [203](#), [206](#), [304](#), [306](#), [322](#), [362](#), [363](#), [365](#), [366](#),
 [386](#), [414](#), [416](#), [419](#), [420](#), [423–425](#), [431–436](#), [439](#), [440](#), [473](#), [488](#), [529](#), [535](#), [546](#)
 PMIxClient.spawn (Python), [488](#)

pmix_spawn_cbfunc_t, [169](#), [180](#), [180](#), [362](#)

PMIx_Spawn_nb, [169](#), [178](#), [180](#), [181](#), [529](#)

PMIx_Store_internal, [113](#), [114](#), [114](#), [484](#), [529](#)
 PMIxClient.store_internal (Python), [484](#)

PMIx_tool_attach_to_server, [411](#), [414](#), [423](#), [444](#), [446](#), [520](#), [539](#), [555](#)
 PMIxTool.attach_to_server (Python), [520](#)

PMIx_tool_connect_to_server, [532](#), [557](#)

pmix_tool_connection_cbfunc_t, [380](#), [381](#), [382](#)

PMIx_tool_disconnect, [445](#), [520](#), [539](#)
 PMIxTool.disconnect (Python), [519](#)

PMIx_tool_finalize, [444](#), [519](#), [530](#)
 PMIxTool.finalize (Python), [519](#)

PMIx_tool_get_servers, [447](#), [521](#), [539](#)
 PMIxTool.get_servers (Python), [520](#)

PMIx_tool_init, [59](#), [408](#), [411](#), [413–415](#), [422–424](#), [426](#), [441](#), [444](#), [519](#), [530](#)
 PMIxTool.init (Python), [519](#)

PMIx_tool_set_server, [410](#), [423](#), [424](#), [447](#), [448](#), [521](#), [539](#)
 PMIxTool.set_server (Python), [521](#)

PMIx_Unpublish, [131](#), [133](#), [134](#), [488](#), [529](#)
 PMIxClient.unpublish (Python), [487](#)

PMIx_Unpublish_nb, [133](#), [529](#)

PMIx_Validate_credential, [285](#), [494](#), [532](#), [537](#)
 PMIxClient.validate_credential (Python), [493](#)

PMIx_Validate_credential_nb, [286](#)

pmix_validation_cbfunc_t, [287](#), [396](#), [397](#)

pmix_value_cbfunc_t, [53](#), [53](#)

pmix_evhdlr_reg_cbfunc_t
 (Deprecated), [555](#)

pmix_server_client_connected_fn_t
 (Deprecated), [344](#), [555](#)

PMIx_tool_connect_to_server

(Deprecated), [555](#)

Index of Support Macros

PMIX_APP_CONSTRUCT, [179](#)
PMIX_APP_CREATE, [179](#)
PMIX_APP_DESTRUCT, [179](#)
PMIX_APP_FREE, [180](#)
PMIX_APP_INFO_CREATE, [180](#), [532](#), [534](#)
PMIX_APP_RELEASE, [179](#)
PMIX_ARGV_APPEND, [46](#)
PMIX_ARGV_APPEND_UNIQUE, [47](#)
PMIX_ARGV_COPY, [49](#)
PMIX_ARGV_COUNT, [49](#)
PMIX_ARGV_FREE, [47](#)
PMIX_ARGV_JOIN, [48](#)
PMIX_ARGV_PREPEND, [46](#)
PMIX_ARGV_SPLIT, [48](#)
PMIX_BYTE_OBJECT_CONSTRUCT, [43](#)
PMIX_BYTE_OBJECT_CREATE, [43](#)
PMIX_BYTE_OBJECT_DESTRUCT, [43](#)
PMIX_BYTE_OBJECT_FREE, [44](#)
PMIX_BYTE_OBJECT_LOAD, [44](#)
PMIX_CHECK_KEY, [17](#)
PMIX_CHECK_NSPACE, [18](#)
PMIX_CHECK_PROCID, [22](#)
PMIX_CHECK_RANK, [20](#)
PMIX_CHECK_RESERVED_KEY, [17](#), [555](#)
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, [45](#)
PMIX_DATA_ARRAY_CREATE, [45](#)
PMIX_DATA_ARRAY_DESTRUCT, [45](#)
PMIX_DATA_ARRAY_FREE, [45](#)
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, [198](#)
PMIX_DEVICE_DIST_CREATE, [198](#)
PMIX_DEVICE_DIST_DESTRUCT, [198](#)
PMIX_DEVICE_DIST_FREE, [199](#)
PMIX_ENDPOINT_CONSTRUCT, [263](#)
PMIX_ENDPOINT_CREATE, [263](#)
PMIX_ENDPOINT_DESTRUCT, [263](#)
PMIX_ENDPOINT_FREE, [264](#)
PMIX_ENVAR_CONSTRUCT, [41](#)
PMIX_ENVAR_CREATE, [42](#)
PMIX_ENVAR_DESTRUCT, [13](#), [41](#)
PMIX_ENVAR_FREE, [42](#)
PMIX_ENVAR_LOAD, [42](#)
PMIX_FABRIC_CONSTRUCT, [272](#)
PMIX_GEOMETRY_CONSTRUCT, [266](#)
PMIX_GEOMETRY_CREATE, [267](#)
PMIX_GEOMETRY_DESTRUCT, [267](#)
PMIX_GEOMETRY_FREE, [267](#)
PMIx_Heartbeat, [220](#), [530](#)
PMIX_INFO_CONSTRUCT, [33](#)
PMIX_INFO_CREATE, [33](#), [38](#), [40](#)
PMIX_INFO_DESTRUCT, [33](#)
PMIX_INFO_FREE, [34](#)
PMIX_INFO_IS_END, [40](#), [532](#), [534](#)
PMIX_INFO_IS_OPTIONAL, [40](#)
PMIX_INFO_IS_REQUIRED, [38](#), [39](#), [39](#)
PMIX_INFO_LIST_ADD, [36](#), [555](#)
PMIX_INFO_LIST_CONVERT, [37](#), [555](#)
PMIX_INFO_LIST_RELEASE, [37](#), [555](#)
PMIX_INFO_LIST_START, [36](#), [36](#), [37](#), [555](#)
PMIX_INFO_LIST_XFER, [37](#), [555](#)
PMIX_INFO_LOAD, [34](#)
PMIX_INFO_OPTIONAL, [39](#)
PMIX_INFO_PROCESSED, [40](#), [555](#)
PMIX_INFO_REQUIRED, [38](#), [39](#)
PMIX_INFO_TRUE, [35](#)
PMIX_INFO_WAS_PROCESSED, [40](#), [555](#)
PMIX_INFO_XFER, [35](#), [308](#)
PMIX_LOAD_KEY, [17](#)

PMIX_LOAD_NSPACE, [19](#)
PMIX_LOAD_PROCID, [22](#), [23](#)
PMIX_MULTICLUSTER_NSPACE_CONSTRUCT, [24](#)
PMIX_MULTICLUSTER_NSPACE_PARSE, [24](#)
PMIX_NSPACE_INVALID, [19](#), [559](#)
PMIX_PDATA_CONSTRUCT, [127](#)
PMIX_PDATA_CREATE, [128](#)
PMIX_PDATA_DESTRUCT, [127](#)
PMIX_PDATA_FREE, [128](#)
PMIX_PDATA_LOAD, [128](#)
PMIX_PDATA_RELEASE, [128](#)
PMIX_PDATA_XFER, [129](#)
PMIX_PROC_CONSTRUCT, [21](#)
PMIX_PROC_CREATE, [21](#)
PMIX_PROC_DESTRUCT, [21](#)
PMIX_PROC_FREE, [22](#), [79](#)
PMIX_PROC_INFO_CONSTRUCT, [26](#)
PMIX_PROC_INFO_CREATE, [27](#)
PMIX_PROC_INFO_DESTRUCT, [26](#)
PMIX_PROC_INFO_FREE, [27](#)
PMIX_PROC_INFO_RELEASE, [27](#)
PMIX_PROC_LOAD, [22](#)
PMIX_PROC_RELEASE, [21](#)
PMIX_PROCID_INVALID, [23](#), [559](#)
PMIX_PROCID_XFER, [23](#), [559](#)
PMIX_QUERY_CONSTRUCT, [93](#)
PMIX_QUERY_CREATE, [94](#)
PMIX_QUERY_DESTRUCT, [93](#)
PMIX_QUERY_FREE, [94](#)
PMIX_QUERY_QUALIFIERS_CREATE, [94](#), [532](#), [534](#)
PMIX_QUERY_RELEASE, [94](#)
PMIX_RANK_IS_VALID, [20](#), [559](#)
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_SETEENV, [49](#)
PMIX_SYSTEM_EVENT, [141](#)
PMIX_TOPOLOGY_CONSTRUCT, [191](#)
PMIX_TOPOLOGY_CREATE, [191](#)
PMIX_TOPOLOGY_DESTRUCT, [191](#)

PMIX_TOPOLOGY_FREE, [191](#)
PMIX_VALUE_CONSTRUCT, [29](#)
PMIX_VALUE_CREATE, [30](#)
PMIX_VALUE_DESTRUCT, [29](#), [73](#), [77](#), [542](#)
PMIX_VALUE_FREE, [30](#)
PMIX_VALUE_GET_NUMBER, [32](#)
PMIX_VALUE_LOAD, [30](#)
PMIX_VALUE_RELEASE, [30](#)
PMIX_VALUE_UNLOAD, [31](#)
PMIX_VALUE_XFER, [32](#)

Index of Data Structures

pmix_alloc_directive_t, 51, 56, 201, 204, 207, 207, 208, 385, 463, 504
pmix_app_t, 46, 47, 50, 164–166, 169–171, 175, 176, 178, 178–180, 362, 363, 365, 416, 418–420, 422, 423, 432, 435, 440, 464, 532, 534, 554
pmix_bind_envelope_t, 193, 193, 465, 539
pmix_byte_object_t, 43, 43, 44, 51, 158, 159, 282, 283, 285, 287, 335, 395, 396, 402, 452, 462
pmix_coord_t, 51, 264, 264–266, 465, 539
pmix_coord_view_t, 267, 465, 539
pmix_cpuset_t, 51, 194, 195, 338, 339, 340, 340, 341, 464, 539
pmix_data_array_t, 28, 37, 44, 44, 45, 51, 83, 84, 88, 90, 91, 96, 106, 202, 205, 207, 232, 235, 270, 272–276, 298, 299, 301–303, 311, 313, 314, 325, 378, 387, 407, 433, 441, 463, 532, 534, 550–553
pmix_data_buffer_t, 150, 150–154, 157–159
pmix_data_range_t, 51, 56, 122, 122, 146, 374, 462, 503
pmix_data_type_t, 31, 32, 34, 36, 45, 50, 50, 51, 56, 129, 153, 155–157, 333, 461, 504
pmix_device_distance_t, 51, 194, 196, 197, 197–199, 305, 465, 539, 552
pmix_device_type_t, 52, 58, 196, 196, 197, 275, 465, 507, 539
pmix_endpoint_t, 52, 262, 262–264, 275, 464, 550
pmix_envar_t, 13, 41, 41, 42, 51, 463
pmix_fabric_operation_t, 268, 268, 406
pmix_fabric_t, 262, 268, 269, 269, 272, 273, 276–280, 407, 464, 539, 549
pmix_geometry_t, 51, 261, 266, 266, 267, 274, 465, 539, 550, 551
pmix_group_operation_t, 404, 406, 406, 539
pmix_group_opt_t, 252, 254, 255, 255, 495, 539
pmix_info_directives_t, 38, 38, 51, 56, 463, 503
pmix_info_t, 4, 5, 8, 17, 33, 33–40, 51, 54, 60, 63, 65, 80, 84, 90, 92–94, 96, 119, 121–124, 143, 146, 149, 180, 194, 195, 201, 202, 204, 205, 207–209, 211, 214, 215, 217, 220, 224, 227, 229, 237, 239, 241, 243, 245, 247, 250, 252, 254, 256, 257, 269, 270, 273–275, 282, 283, 285, 287, 290, 292, 297–299, 301, 302, 307, 308, 311, 313, 314, 325, 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, 530, 532, 534, 543–546, 549–551
pmix_iof_channel_t, 51, 57, 335, 399, 401, 429, 429, 449, 463, 505
pmix_job_state_t, 27, 27, 51, 57, 464, 505, 539
pmix_key_t, 8, 16, 16, 72, 113, 333, 461
pmix_link_state_t, 51, 57, 262, 268, 268, 271, 274, 464, 506, 539, 551
pmix_locality_t, 52, 190, 192, 192, 464, 539, 557, 558
pmix_nspace_t, 18, 18, 19, 22–24, 51, 180, 461, 462
pmix_pdata_t, 123, 124, 127, 127–130, 463

pmix_persistence_t, 51, 56, 122, 122, 462, 503
pmix_proc_info_t, 25, 25–27, 51, 82, 83, 87, 88, 90, 91, 378, 433, 441, 462
pmix_proc_state_t, 24, 24, 51, 55, 462, 502
pmix_proc_t, 19, 20, 20–23, 51, 63, 67, 69, 70, 74, 91, 106, 129, 139–141, 143, 146, 147, 153, 154,
162, 163, 232, 235–237, 241, 247, 250, 253, 303, 319, 321, 322, 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, 552, 553
pmix_query_t, 51, 82, 87, 90, 92, 92–95, 377, 379, 464, 532, 534, 543
pmix_rank_t, 19, 19, 20, 22, 23, 51, 462
pmix_regattr_t, 51, 96, 329, 330, 330–333, 464, 537, 539, 545
pmix_scope_t, 51, 55, 114, 114, 462, 502
pmix_status_t, 14, 14, 31, 32, 36, 37, 46, 47, 49, 51, 54, 55, 138, 141, 143, 146, 149, 196, 324, 328,
371, 373, 374, 382, 395, 398, 461, 475, 501
pmix_storage_access_type_t, 457, 457, 559
pmix_storage_accessibility_t, 456, 456, 559
pmix_storage_medium_t, 455, 455, 456, 559
pmix_storage_persistence_t, 456, 456, 559
pmix_topology_t, 52, 189, 190, 190, 191, 194, 195, 539
pmix_value_t, 8, 28, 28–32, 51, 53, 73, 74, 77, 113, 463, 542

Index of Constants

PMIX_ALLOC_DIRECTIVE, [51](#)
PMIX_ALLOC_EXTEND, [208](#)
PMIX_ALLOC_EXTERNAL, [208](#)
PMIX_ALLOC_NEW, [208](#)
PMIX_ALLOC_REAQUIRE, [208](#)
PMIX_ALLOC_RELEASE, [208](#)
PMIX_APP, [51](#)
PMIX_APP_WILDCARD, [13](#)
PMIX_BOOL, [50](#)
PMIX_BUFFER, [51](#)
PMIX_BYTE, [50](#)
PMIX_BYTE_OBJECT, [51](#)
PMIX_COMMAND, [51](#)
PMIX_COMPRESSED_BYTE_OBJECT, [51](#)
PMIX_COMPRESSED_STRING, [51](#)
PMIX_COORD, [51](#)
PMIX_COORD_LOGICAL_VIEW, [268](#)
PMIX_COORD_PHYSICAL_VIEW, [268](#)
PMIX_COORD_VIEW_UNDEF, [268](#)
PMIX_CPUBIND_PROCESS, [193](#)
PMIX_CPUBIND_THREAD, [193](#)
PMIX_DATA_ARRAY, [51](#)
PMIX_DATA_RANGE, [51](#)
PMIX_DATA_TYPE, [51](#)
PMIX_DATA_TYPE_MAX, [52](#)
PMIX_DEBUG_WAITING_FOR_NOTIFY, [439](#)
PMIX_DEBUGGER_RELEASE, [439](#)
PMIX_DEVICE_DIST, [51](#)
PMIX_DEVTYPE, [52](#)
PMIX_DEVTYPE_BLOCK, [197](#)
PMIX_DEVTYPE_COPROC, [197](#)
PMIX_DEVTYPE_DMA, [197](#)
PMIX_DEVTYPE_GPU, [197](#)
PMIX_DEVTYPE_NETWORK, [197](#)
PMIX_DEVTYPE_OPENFABRICS, [197](#)
PMIX_DEVTYPE_UNKNOWN, [197](#)
PMIX_DOUBLE, [50](#)
PMIX_ENDPOINT, [52](#)

PMIX_ENVAR, [51](#)
PMIX_ERR_BAD_PARAM, [15](#)
PMIX_ERR_COMM_FAILURE, [15](#)
PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES, [214](#)
PMIX_ERR_DUPLICATE_KEY, [121](#)
PMIX_ERR_EMPTY, [15](#)
PMIX_ERR_EVENT_REGISTRATION, [140](#)
PMIX_ERR_EXISTS, [14](#)
PMIX_ERR_EXISTS_OUTSIDE_SCOPE, [14](#)
PMIX_ERR_INIT, [15](#)
PMIX_ERR_INVALID_CRED, [14](#)
PMIX_ERR_INVALID_OPERATION, [15](#)
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, [15](#)
PMIX_ERR_NO_PERMISSIONS, [14](#)
PMIX_ERR_NOMEM, [15](#)
PMIX_ERR_NOT_FOUND, [15](#)
PMIX_ERR_NOT_SUPPORTED, [15](#)
PMIX_ERR_OUT_OF_RESOURCE, [15](#)
PMIX_ERR_PACK_FAILURE, [14](#)
PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED, [15](#)
PMIX_ERR_PARTIAL_SUCCESS, [15](#)
PMIX_ERR_PROC_CHECKPOINT, [214](#)
PMIX_ERR_PROC_MIGRATE, [214](#)
PMIX_ERR_PROC_RESTART, [214](#)
PMIX_ERR_PROC_TERM_WO_SYNC, [438](#)
PMIX_ERR_REPEAT_ATTR_REGISTRATION, [330](#)
PMIX_ERR_RESOURCE_BUSY, [15](#)
PMIX_ERR_TIMEOUT, [14](#)

PMIX_ERR_TYPE_MISMATCH, [14](#)
PMIX_ERR_UNKNOWN_DATA_TYPE, [14](#)
PMIX_ERR_UNPACK_FAILURE, [14](#)
PMIX_ERR_UNPACK_INADEQUATE_SPACE, [14](#)
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, [14](#)
PMIX_ERR_UNREACH, [15](#)
PMIX_ERR_WOULD_BLOCK, [14](#)
PMIX_ERROR, [14](#)
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, [140](#)
PMIX_EVENT_NODE_OFFLINE, [140](#)
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, [140](#)
PMIX_EVENT_SYS_OTHER, [140](#)
PMIX_EXTERNAL_ERR_BASE, [15](#)
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, [50](#)
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, [51](#)
PMIX_GLOBAL, [114](#)
PMIX_GROUP_ACCEPT, [255](#)
PMIX_GROUP_CONSTRUCT, [406](#)
PMIX_GROUP_CONSTRUCT_ABORT, [235](#)
PMIX_GROUP_CONSTRUCT_COMPLETE, [235](#)
PMIX_GROUP_CONTEXT_ID_ASSIGNED, [235](#)
PMIX_GROUP_DECLINE, [255](#)
PMIX_GROUP_DESTRUCT, [406](#)

PMIX_GROUP_INVITE_ACCEPTED, [235](#)
PMIX_GROUP_INVITE_DECLINED, [235](#)
PMIX_GROUP_INVITE_FAILED, [235](#)
PMIX_GROUP_INVITED, [234](#)
PMIX_GROUP_LEADER_FAILED, [235](#)
PMIX_GROUP_LEADER_SELECTED, [235](#)
PMIX_GROUP_LEFT, [234](#)
PMIX_GROUP_MEMBER_FAILED, [234](#)
PMIX_GROUP_MEMBERSHIP_UPDATE, [235](#)
PMIX_INFO, [51](#)
PMIX_INFO_ARRAY_END, [38](#)
PMIX_INFO_DIR_RESERVED, [38](#)
PMIX_INFO_DIRECTIVES, [51](#)
PMIX_INFO_REQD, [38](#)
PMIX_INFO_REQD_PROCESSED, [38](#)
PMIX_INT, [50](#)
PMIX_INT16, [50](#)
PMIX_INT32, [50](#)
PMIX_INT64, [50](#)
PMIX_INT8, [50](#)
PMIX_INTERNAL, [114](#)
PMIX_IOF_CHANNEL, [51](#)
PMIX_JCTRL_CHECKPOINT, [214](#)
PMIX_JCTRL_CHECKPOINT_COMPLETE, [214](#)
PMIX_JCTRL_PREEMPT_ALERT, [214](#)
PMIX_JOB_STATE, [51](#)
PMIX_JOB_STATE_AWAITING_ALLOC, [28](#)
PMIX_JOB_STATE_CONNECTED, [28](#)
PMIX_JOB_STATE_LAUNCH_UNDERWAY, [28](#)
PMIX_JOB_STATE_RUNNING, [28](#)
PMIX_JOB_STATE_SUSPENDED, [28](#)
PMIX_JOB_STATE_TERMINATED, [28](#)
PMIX_JOB_STATE_TERMINATED_WITH_ERROR, [28](#)
PMIX_JOB_STATE_UNDEF, [28](#)
PMIX_JOB_STATE_UNTERMINATED, [28](#)
PMIX_KVAL, [51](#)
PMIX_LAUNCH_COMPLETE, [438](#)
PMIX_LAUNCHER_READY, [425](#)
PMIX_LINK_DOWN, [268](#)
PMIX_LINK_STATE, [51](#)
PMIX_LINK_STATE_UNKNOWN, [268](#)
PMIX_LINK_UP, [268](#)
PMIX_LOCAL, [114](#)

PMIX_LOCALITY_NONLOCAL, [192](#)
PMIX_LOCALITY_SHARE_CORE, [192](#)
PMIX_LOCALITY_SHARE_HWTHREAD, [192](#)
PMIX_LOCALITY_SHARE_L1CACHE, [192](#)
PMIX_LOCALITY_SHARE_L2CACHE, [192](#)
PMIX_LOCALITY_SHARE_L3CACHE, [192](#)
PMIX_LOCALITY_SHARE_NODE, [192](#)
PMIX_LOCALITY_SHARE_NUMA, [192](#)
PMIX_LOCALITY_SHARE_PACKAGE, [192](#)
PMIX_LOCALITY_UNKNOWN, [192](#)
PMIX_LOCTYPE, [52](#)
PMIX_MAX_KEYLEN, [13](#)
PMIX_MAX_NSLEN, [13](#)
PMIX_MODEL_DECLARED, [63](#)
PMIX_MODEL_RESOURCES, [63](#)
PMIX_MONITOR_FILE_ALERT, [221](#)
PMIX_MONITOR_HEARTBEAT_ALERT, [221](#)
PMIX_OPENMP_PARALLEL_ENTERED, [63](#)
PMIX_OPENMP_PARALLEL_EXITED, [63](#)
PMIX_OPERATION_IN_PROGRESS, [15](#)
PMIX_OPERATION_SUCCEEDED, [15](#)
PMIX_PDATA, [51](#)
PMIX_PERSIST, [51](#)
PMIX_PERSIST_APP, [122](#)
PMIX_PERSIST_FIRST_READ, [122](#)
PMIX_PERSIST_INDEF, [122](#)
PMIX_PERSIST_INVALID, [122](#)
PMIX_PERSIST_PROC, [122](#)
PMIX_PERSIST_SESSION, [122](#)
PMIX_PID, [50](#)
PMIX_POINTER, [51](#)
PMIX_PROC, [51](#)
PMIX_PROC_CPUSSET, [51](#)
PMIX_PROC_INFO, [51](#)
PMIX_PROC_NSPACE, [51](#)
PMIX_PROC_RANK, [51](#)
PMIX_PROC_STATE, [51](#)
PMIX_PROC_STATE_ABORTED, [25](#)
PMIX_PROC_STATE_ABORTED_BY_SIG, [25](#)
PMIX_PROC_STATE_CALLED_ABORT, [25](#)
PMIX_PROC_STATE_CANNOT_RESTART, [25](#)
PMIX_PROC_STATE_COMM_FAILED, [25](#)
PMIX_PROC_STATE_CONNECTED, [25](#)

PMIX_PROC_STATE_ERROR, [25](#)
PMIX_PROC_STATE_FAILED_TO_LAUNCH, [25](#)
PMIX_PROC_STATE_FAILED_TO_START, [25](#)
PMIX_PROC_STATE_HEARTBEAT_FAILED, [25](#)
PMIX_PROC_STATE_KILLED_BY_CMD, [25](#)
PMIX_PROC_STATE_LAUNCH_UNDERWAY, [25](#)
PMIX_PROC_STATE_MIGRATING, [25](#)
PMIX_PROC_STATE_PREPPED, [25](#)
PMIX_PROC_STATE_RESTART, [25](#)
PMIX_PROC_STATE_RUNNING, [25](#)
PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED, [25](#)
PMIX_PROC_STATE_TERM_NON_ZERO, [25](#)
PMIX_PROC_STATE_TERM_WO_SYNC, [25](#)
PMIX_PROC_STATE_TERMINATE, [25](#)
PMIX_PROC_STATE_TERMINATED, [25](#)
PMIX_PROC_STATE_UNDEF, [25](#)
PMIX_PROC_STATE_UNTERMINATED, [25](#)
PMIX_PROCESS_SET_DEFINE, [231](#)
PMIX_PROCESS_SET_DELETE, [231](#)
PMIX_QUERY, [51](#)
PMIX_QUERY_PARTIAL_SUCCESS, [89](#)
PMIX_RANGE_CUSTOM, [122](#)
PMIX_RANGE_GLOBAL, [122](#)
PMIX_RANGE_INVALID, [122](#)
PMIX_RANGE_LOCAL, [122](#)
PMIX_RANGE_NAMESPACE, [122](#)
PMIX_RANGE_PROC_LOCAL, [122](#)
PMIX_RANGE_RM, [122](#)
PMIX_RANGE_SESSION, [122](#)
PMIX_RANGE_UNDEF, [122](#)
PMIX_RANK_INVALID, [20](#)
PMIX_RANK_LOCAL_NODE, [19](#)
PMIX_RANK_LOCAL_PEERS, [20](#)
PMIX_RANK_UNDEF, [19](#)
PMIX_RANK_VALID, [20](#)
PMIX_RANK_WILDCARD, [19](#)
PMIX_REGATTR, [51](#)
PMIX_REGEX, [51](#)
PMIX_REMOTE, [114](#)
PMIX_SCOPE, [51](#)
PMIX_SCOPE_UNDEF, [114](#)
PMIX_SIZE, [50](#)
PMIX_STATUS, [51](#)

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, [50](#)
PMIX_SUCCESS, [14](#)
PMIX_TIME, [51](#)
PMIX_TIMEVAL, [51](#)
PMIX_TOPO, [52](#)
PMIX_UINT, [50](#)
PMIX_UINT16, [50](#)
PMIX_UINT32, [50](#)
PMIX_UINT64, [50](#)
PMIX_UINT8, [50](#)
PMIX_UNDEF, [50](#)
PMIX_VALUE, [51](#)

PMIX_CONNECT_REQUESTED
 Deprecated, [556](#)
PMIX_ERR_DATA_VALUE_NOT_FOUND
 Deprecated, [535](#)
 Removed, [557](#)
PMIX_ERR_DEBUGGER_RELEASE
 Deprecated, [556](#)
PMIX_ERR_HANDSHAKE_FAILED

Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_IN_ERRNO
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_ARG
Deprecated, [535](#)
Removed, [557](#)

PMIX_ERR_INVALID_ARGS
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_KEY
Deprecated, [535](#)
Removed, [557](#)

PMIX_ERR_INVALID_KEY_LENGTH
Deprecated, [535](#)
Removed, [557](#)

PMIX_ERR_INVALID_KEYVALP
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_LENGTH
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_NAMESPACE
Deprecated, [535](#)
Removed, [557](#)

PMIX_ERR_INVALID_NUM_ARGS
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_NUM_PARSED
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_SIZE
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_INVALID_TERMINATION
Deprecated, [556](#)

PMIX_ERR_INVALID_VAL
Deprecated, [535](#)
Removed, [557](#)

PMIX_ERR_INVALID_VAL_LENGTH
Deprecated, [535](#)
Removed, [556](#)

PMIX_ERR_JOB_TERMINATED
 Deprecated, [556](#)

PMIX_ERR_LOST_CONNECTION_TO_CLIENT
 Deprecated, [556](#)

PMIX_ERR_LOST_CONNECTION_TO_SERVER
 Deprecated, [556](#)

PMIX_ERR_LOST_PEER_CONNECTION
 Deprecated, [556](#)

PMIX_ERR_NODE_DOWN
 Deprecated, [556](#)

PMIX_ERR_NODE_OFFLINE
 Deprecated, [556](#)

PMIX_ERR_NOT_IMPLEMENTED
 Deprecated, [535](#)
 Removed, [557](#)

PMIX_ERR_PACK_MISMATCH
 Deprecated, [535](#)
 Removed, [557](#)

PMIX_ERR_PROC_ABORTED
 Deprecated, [556](#)

PMIX_ERR_PROC_ABORTING
 Deprecated, [556](#)

PMIX_ERR_PROC_ENTRY_NOT_FOUND
 Deprecated, [535](#)
 Removed, [556](#)

PMIX_ERR_PROC_REQUESTED_ABORT
 Deprecated, [535](#)
 Removed, [556](#)

PMIX_ERR_READY_FOR_HANDSHAKE
 Deprecated, [535](#)
 Removed, [556](#)

PMIX_ERR_SERVER_FAILED_REQUEST
 Deprecated, [535](#)
 Removed, [556](#)

PMIX_ERR_SERVER_NOT_AVAIL
 Deprecated, [535](#)
 Removed, [557](#)

PMIX_ERR_SILENT
 Deprecated, [535](#)
 Removed, [557](#)

PMIX_ERR_SYS_OTHER
 Deprecated, [556](#)

PMIX_EXISTS

Deprecated, [556](#)
PMIX_GDS_ACTION_COMPLETE
 Deprecated, [535](#)
 Removed, [557](#)
PMIX_INFO_ARRAY
 Deprecated, [530](#)
PMIX_MODEX
 Deprecated, [530](#)
PMIX_NOTIFY_ALLOC_COMPLETE
 Deprecated, [535](#)
 Removed, [557](#)
PMIX_PROC_HAS_CONNECTED
 Deprecated, [556](#)
PMIX_PROC_TERMINATED
 Deprecated, [556](#)

Index of Environmental Variables

PMIX_KEEPALIVE_PIPE, [414](#), [423](#), [424](#), [555](#)
PMIX_LAUNCHER_RNDZ_FILE, [410](#), [414](#), [555](#)
PMIX_LAUNCHER_RNDZ_URI, [414](#), [423](#), [424](#), [555](#)

Index of Attributes

PMIX_ACCESS_GRPIDS, [122](#), [546](#)
PMIX_ACCESS_PERMISSIONS, [119](#), [121](#), [121](#), [546](#)
PMIX_ACCESS_USERIDS, [122](#), [546](#)
PMIX_ADD_ENVAR, [167](#), [172](#), [178](#)
PMIX_ADD_HOST, [165](#), [170](#), [175](#), [364](#)
PMIX_ADD_HOSTFILE, [165](#), [170](#), [175](#), [364](#)
PMIX_ALL_CLONES_PARTICIPATE, [68](#), [71](#), [71](#), [183](#), [185](#), [186](#), [188](#), [542](#)
PMIX_ALLOC_BANDWIDTH, [167](#), [173](#), [202](#), [205](#), [207](#), [207](#), [325](#), [326](#), [387](#)
PMIX_ALLOC_CPU_LIST, [167](#), [173](#), [202](#), [205](#), [206](#), [386](#)
PMIX_ALLOC_FABRIC, [202](#), [205](#), [207](#), [325](#), [386](#), [557](#)
PMIX_ALLOC_FABRIC_ENDPTS, [168](#), [173](#), [202](#), [203](#), [205](#), [206](#), [207](#), [207](#), [325](#), [326](#), [386](#), [557](#)
PMIX_ALLOC_FABRIC_ENDPTS_NODE, [168](#), [173](#), [203](#), [206](#), [207](#), [326](#), [557](#)
PMIX_ALLOC_FABRIC_ID, [202](#), [205](#), [207](#), [207](#), [325](#), [386](#), [557](#)
PMIX_ALLOC_FABRIC_PLANE, [168](#), [173](#), [202](#), [205](#), [206](#), [207](#), [207](#), [325](#), [326](#), [387](#), [557](#)
PMIX_ALLOC_FABRIC_QOS, [167](#), [173](#), [202](#), [205](#), [207](#), [207](#), [325](#), [326](#), [387](#), [557](#)
PMIX_ALLOC_FABRIC_SEC_KEY, [202](#), [203](#), [205](#), [206](#), [207](#), [207](#), [325](#), [387](#), [557](#)
PMIX_ALLOC_FABRIC_TYPE, [168](#), [173](#), [202](#), [205](#), [206](#), [207](#), [207](#), [325](#), [326](#), [386](#), [387](#), [557](#)
PMIX_ALLOC_ID, [203](#), [206](#), [386](#), [534](#), [535](#), [546](#)
PMIX_ALLOC_MEM_SIZE, [167](#), [173](#), [202](#), [205](#), [207](#), [386](#)
PMIX_ALLOC_NODE_LIST, [167](#), [173](#), [202](#), [205](#), [206](#), [386](#)
PMIX_ALLOC_NUM_CPU_LIST, [167](#), [173](#), [202](#), [205](#), [206](#), [386](#)
PMIX_ALLOC_NUM_CPUS, [167](#), [173](#), [201](#), [204](#), [206](#), [386](#)
PMIX_ALLOC_NUM_NODES, [167](#), [173](#), [201](#), [204](#), [206](#), [386](#)
PMIX_ALLOC_QUEUE, [83](#), [88](#), [90](#), [167](#), [172](#), [206](#), [378](#), [546](#)
PMIX_ALLOC_REQ_ID, [201](#), [204](#), [206](#), [534](#)
PMIX_ALLOC_TIME, [167](#), [172](#), [201](#), [205](#), [207](#), [386](#)
PMIX_ALLOCATED_NODELIST, [100](#), [299](#)
PMIX_ANL_MAP, [101](#), [102](#), [300](#)
PMIX_APP_ARGV, [103](#), [301](#), [546](#)
PMIX_APP_INFO, [73](#), [76](#), [81](#), [86](#), [98](#), [103](#), [106](#), [301](#), [302](#)
PMIX_APP_INFO_ARRAY, [298](#), [301](#), [307](#), [307](#), [313](#), [545](#)
PMIX_APP_MAP_REGEX, [104](#), [301](#)
PMIX_APP_MAP_TYPE, [104](#), [301](#)
PMIX_APP_RANK, [104](#), [304](#)
PMIX_APP_SIZE, [103](#), [301](#), [312](#)
PMIX_APPEND_ENVAR, [167](#), [172](#), [178](#)
PMIX_APPLDR, [103](#), [301](#), [313](#)
PMIX_APPNUM, [73](#), [76](#), [81](#), [86](#), [99](#), [103](#), [104](#), [106](#), [298](#), [301–303](#), [307](#), [313](#), [545](#)

PMIX_ATTR_UNDEF, [5](#)
PMIX_AVAIL_PHYS_MEMORY, [92](#), [106](#), [303](#)
PMIX_BINDTO, [165](#), [171](#), [175](#), [300](#), [364](#)
PMIX_CLEANUP_EMPTY, [210](#), [213](#), [216](#)
PMIX_CLEANUP_IGNORE, [210](#), [213](#), [216](#)
PMIX_CLEANUP_LEAVE_TOPDIR, [210](#), [213](#), [216](#)
PMIX_CLEANUP_RECURSIVE, [210](#), [213](#), [216](#)
PMIX_CLIENT_ATTRIBUTES, [82](#), [87](#), [92](#), [96](#), [432](#), [537](#), [544](#)
PMIX_CLIENT_AVG_MEMORY, [84](#), [89](#), [92](#)
PMIX_CLIENT_FUNCTIONS, [82](#), [87](#), [91](#), [92](#), [96](#), [543](#), [544](#)
PMIX_CLUSTER_ID, [99](#), [299](#)
PMIX_CMD_LINE, [102](#), [546](#)
PMIX_COLLECT_DATA, [68](#), [70](#), [71](#), [112](#), [351](#)
PMIX_COLLECT_GENERATED_JOB_INFO, [68](#), [70](#), [71](#), [71](#), [260](#), [351](#), [542](#)
PMIX_COLLECTIVE_ALGO, [531](#)
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](#)
PMIX_CPU_LIST, [166](#), [172](#), [176](#), [366](#)
PMIX_CPUS_PER_PROC, [166](#), [171](#), [176](#), [366](#)
PMIX_CPUSET, [105](#), [193](#), [305](#), [339](#)
PMIX_CPUSET_BITMAP, [105](#), [305](#), [547](#)
PMIX_CRED_TYPE, [284](#), [394](#)
PMIX_CREDENTIAL, [105](#), [380](#)
PMIX_CRYPTO_KEY, [284](#), [301](#)
PMIX_DAEMON_MEMORY, [84](#), [89](#), [92](#)
PMIX_DATA_SCOPE, [73](#), [76](#), [78](#)
PMIX_DEBUG_DAEMONS_PER_NODE, [365](#), [434](#), [435](#), [440](#), [441](#), [549](#)
PMIX_DEBUG_DAEMONS_PER_PROC, [365](#), [434](#), [435](#), [440](#), [441](#), [548](#)
PMIX_DEBUG_STOP_IN_INIT, [418](#), [422](#), [424](#), [432](#), [435](#), [440](#)
PMIX_DEBUG_STOP_ON_EXEC, [418](#), [424](#), [431](#), [433](#), [439](#)
PMIX_DEBUG_TARGET, [365](#), [433](#)–[435](#), [440](#), [440](#), [441](#), [548](#), [549](#), [557](#)
PMIX_DEBUG_WAIT_FOR_NOTIFY, [419](#), [432](#), [439](#), [440](#)
PMIX_DEBUGGER_DAEMONS, [365](#), [434](#), [435](#), [440](#)
PMIX_DEVICE_DISTANCES, [199](#), [275](#), [305](#), [552](#)
PMIX_DEVICE_ID, [199](#), [260](#), [261](#), [271](#), [274](#), [275](#), [319](#), [550](#)–[552](#)
PMIX_DEVICE_TYPE, [199](#), [552](#)
PMIX_DISPLAY_MAP, [165](#), [170](#), [175](#), [364](#)
PMIX_EMBED_BARRIER, [65](#), [65](#)
PMIX_ENUM_VALUE, [331](#), [331](#), [537](#), [545](#)
PMIX_EVENT_ACTION_TIMEOUT, [142](#), [148](#)

PMIX_EVENT_AFFECTED_PROC, 139, [141](#), [147](#), 438
PMIX_EVENT_AFFECTED_PROCS, 139, [141](#), [147](#), 438
PMIX_EVENT_BASE, [62](#), [292](#), 444
PMIX_EVENT_CUSTOM_RANGE, 139, [141](#), [147](#)
PMIX_EVENT_DO_NOT_CACHE, [142](#), [147](#)
PMIX_EVENT_HDLR_AFTER, 139, [141](#)
PMIX_EVENT_HDLR_APPEND, 139, [141](#)
PMIX_EVENT_HDLR_BEFORE, 139, [141](#)
PMIX_EVENT_HDLR_FIRST, 139, [141](#)
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, 139, [141](#)
PMIX_EVENT_HDLR_LAST, 139, [141](#)
PMIX_EVENT_HDLR_LAST_IN_CATEGORY, 139, [141](#)
PMIX_EVENT_HDLR_NAME, 139, [141](#)
PMIX_EVENT_HDLR_PREPEND, 139, [141](#)
PMIX_EVENT_NON_DEFAULT, [141](#), [147](#)
PMIX_EVENT_PROXY, [142](#), [147](#)
PMIX_EVENT_RETURN_OBJECT, 139, [141](#)
PMIX_EVENT_SILENT_TERMINATION, 168, 173, [177](#)
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](#), 418, 419, 437, 438, 554, 555
PMIX_EXEC_AGENT, [422](#), [425](#), 436, 548
PMIX_EXIT_CODE, [104](#), [177](#), 418, 419, 437, 555
PMIX_EXTERNAL_PROGRESS, 61, [292](#), [294](#), 444, 547
PMIX_FABRIC_COORDINATES, [273](#), 549
PMIX_FABRIC_COST_MATRIX, [270](#), [272](#), 549
PMIX_FABRIC_DEVICE, [260](#), [270](#), [273](#), [274](#), 551
PMIX_FABRIC_DEVICE_ADDRESS, [261](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_BUS_TYPE, [261](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_COORDINATES, [261](#), [274](#), 551
PMIX_FABRIC_DEVICE_DRIVER, [261](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_FIRMWARE, [261](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_INDEX, [261](#), [274](#), 407, 551
PMIX_FABRIC_DEVICE_MTU, [262](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_NAME, [260](#), [261](#), [270](#), [274](#), 319, 551
PMIX_FABRIC_DEVICE_PCI_DEVID, [261](#), [271](#), [274](#), [275](#), 552
PMIX_FABRIC_DEVICE_SPEED, [262](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_STATE, [262](#), [271](#), [274](#), 551
PMIX_FABRIC_DEVICE_TYPE, [262](#), [271](#), [274](#), 552
PMIX_FABRIC_DEVICE_VENDOR, [261](#), [271](#), [274](#), 551

PMIX_FABRIC_DEVICE_VENDORID, 261, [274](#), 551
PMIX_FABRIC_DEVICES, [260](#), [273](#)
PMIX_FABRIC_DIMS, [270](#), [273](#), 550
PMIX_FABRIC_ENDPT, [275](#), 550
PMIX_FABRIC_GROUPS, [270](#), [272](#), 549
PMIX_FABRIC_IDENTIFIER, 269, [273](#), 276, 407, 549
PMIX_FABRIC_INDEX, [268](#), [273](#), 273, 549
PMIX_FABRIC_NUM_DEVICES, 269, [273](#), 549
PMIX_FABRIC_PLANE, [270](#), [272](#), 273, 276, 277, 407, 550
PMIX_FABRIC_SHAPE, [270](#), [273](#), 550
PMIX_FABRIC_SHAPE_STRING, [270](#), [273](#), 550
PMIX_FABRIC_SWITCH, [272](#), [275](#), 550
PMIX_FABRIC_VENDOR, 269, [273](#), 276, 407, 549
PMIX_FIRST_ENVAR, 167, 172, [178](#), 555
PMIX_FORKEXEC_AGENT, 422, 424, [425](#), 436, 548
PMIX_FWD_STDDIAG, 417, 421, [425](#), 534
PMIX_FWD_STDERR, 365, 381, 416, 421, [425](#), 426
PMIX_FWD_STDIN, 364, 381, 416, 420, [424](#), 426
PMIX_FWD_STDOUT, 365, 381, 416, 421, [424](#), 424, 426
PMIX_GET_POINTER_VALUES, 73, 74, 76, [77](#), 542
PMIX_GET_REFRESH_CACHE, 73, 76, [77](#), 116, 542
PMIX_GET_STATIC_VALUES, 72–74, [77](#), 77, 542
PMIX_GLOBAL_RANK, [104](#), 304
PMIX_GROUP_ASSIGN_CONTEXT_ID, [236](#), 238, 242, 248, 251, 405, 406, 553
PMIX_GROUP_CONTEXT_ID, [236](#), 405, 553
PMIX_GROUP_ENDPT_DATA, [236](#), 405, 554
PMIX_GROUP_FT_COLLECTIVE, [236](#), 238, 242, 247, 251, 553
PMIX_GROUP_ID, [235](#), 235, 405, 553
PMIX_GROUP_LEADER, [236](#), 238, 239, 242, 249, 253, 553
PMIX_GROUP_LOCAL_ONLY, [236](#), 238, 242, 405, 553
PMIX_GROUP_MEMBERSHIP, [236](#), 239, 405
PMIX_GROUP_NAMES, [236](#), 554
PMIX_GROUP_NOTIFY_TERMINATION, [236](#), 238, 239, 242, 244, 248, 251, 553
PMIX_GROUP_OPTIONAL, [236](#), 238, 239, 242, 247, 251, 405, 553
PMIX_GRPID, 82, 87, 119, 120, 124, 126, 132, 133, 201, 204, 209, 212, 217, 219, 223, 226, 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](#), 545
PMIX_HOST, 164, 170, [174](#), 363
PMIX_HOST_ATTRIBUTES, 82, 87, [92](#), 96, 433, 537, 544
PMIX_HOST_FUNCTIONS, 82, 87, 91, [92](#), 96, 543, 544
PMIX_HOSTFILE, 164, 170, [174](#), 363
PMIX_HOSTNAME, 73, 76, 81, 83, 84, 86, 88, 89, 91, 92, 99, [106](#), 106, 260, 261, 271, 275, 298,

302–305, 307, 319, 378, 433, 441, 545, 552
PMIX_HOSTNAME_ALIASES, 106, 302, 547
PMIX_HOSTNAME_KEEP_FQDN, 100, 300, 547
PMIX_IMMEDIATE, 72, 76, 77, 107, 116
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, 548
PMIX_IOF_COPY, 427, 431, 548
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, 548
PMIX_IOF_REDIRECT, 427, 431, 548
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, 176, 366
PMIX_JOB_CTRL_CANCEL, 210, 213, 215, 389
PMIX_JOB_CTRL_CHECKPOINT, 210, 213, 215, 389
PMIX_JOB_CTRL_CHECKPOINT_EVENT, 210, 213, 215, 390
PMIX_JOB_CTRL_CHECKPOINT_METHOD, 211, 214, 215, 390
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL, 210, 213, 215, 390
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT, 211, 213, 215, 390
PMIX_JOB_CTRL_ID, 209, 210, 212, 213, 215, 215, 389
PMIX_JOB_CTRL_KILL, 210, 212, 215, 389
PMIX_JOB_CTRL_PAUSE, 209, 212, 215, 389
PMIX_JOB_CTRL_PREEMPTIBLE, 211, 214, 215, 390
PMIX_JOB_CTRL_PROVISION, 211, 214, 215, 390
PMIX_JOB_CTRL_PROVISION_IMAGE, 211, 214, 215, 390
PMIX_JOB_CTRL_RESTART, 210, 213, 215, 389
PMIX_JOB_CTRL_RESUME, 209, 212, 215, 389
PMIX_JOB_CTRL_SIGNAL, 210, 213, 215, 389
PMIX_JOB_CTRL_TERMINATE, 210, 213, 215, 389
PMIX_JOB_INFO, 73, 76, 81, 86, 98, 101
PMIX_JOB_INFO_ARRAY, 297, 299, 307, 307, 308, 311, 532, 544
PMIX_JOB_NUM_APPS, 102, 300, 311
PMIX_JOB_RECOVERABLE, 166, 172, 176, 366
PMIX_JOB_SIZE, 102, 299, 311, 312, 531, 534
PMIX_JOB_TERM_STATUS, 177, 418, 419, 437, 438, 439, 555
PMIX_JOBID, 101, 297–299, 307, 311, 438, 545
PMIX_LAUNCH_DIRECTIVES, 424, 425, 549
PMIX_LAUNCHER, 409, 414, 415

PMIX_LAUNCHER_DAEMON, 422, [425](#), [547](#)
PMIX_LAUNCHER_RENDEZVOUS_FILE, 410, [415](#), [547](#)
PMIX_LOCAL_CPUSETS, [106](#), [303](#), [315](#)
PMIX_LOCAL_PEERS, [106](#), [106](#), [303](#), [314](#)
PMIX_LOCAL_PROCS, [106](#), [303](#)
PMIX_LOCAL_RANK, [105](#), [304](#), [434–436](#), [441](#), [548](#), [549](#)
PMIX_LOCAL_SIZE, [106](#), [302](#)
PMIX_LOCALITY_STRING, 190, [192](#), [304](#), [339](#)
PMIX_LOCALLDR, [106](#), [302](#)
PMIX_LOG_COMPLETION, [177](#), [418](#), [437](#), [555](#)
PMIX_LOG_EMAIL, [224](#), [227](#), [229](#), [384](#)
PMIX_LOG_EMAIL_ADDR, [224](#), [227](#), [229](#), [384](#)
PMIX_LOG_EMAIL_MSG, [224](#), [227](#), [229](#), [384](#)
PMIX_LOG_EMAIL_SENDER_ADDR, [224](#), [227](#), [229](#)
PMIX_LOG_EMAIL_SERVER, [224](#), [227](#), [229](#)
PMIX_LOG_EMAIL_SRVR_PORT, [224](#), [227](#), [229](#)
PMIX_LOG_EMAIL_SUBJECT, [224](#), [227](#), [229](#), [384](#)
PMIX_LOG_GENERATE_TIMESTAMP, [223](#), [226](#), [228](#)
PMIX_LOG_GLOBAL_DATASTORE, [224](#), [227](#), [229](#)
PMIX_LOG_GLOBAL_SYSLOG, [223](#), [226](#), [228](#)
PMIX_LOG_JOB_EVENTS, [177](#), [418](#), [437](#), [554](#)
PMIX_LOG_JOB_RECORD, [224](#), [227](#), [229](#)
PMIX_LOG_LOCAL_SYSLOG, [223](#), [226](#), [228](#)
PMIX_LOG_MSG, [228](#), [384](#)
PMIX_LOG_ONCE, [223](#), [226](#), [228](#)
PMIX_LOG_PROC_ABNORMAL_TERMINATION, [177](#), [554](#)
PMIX_LOG_PROC_TERMINATION, [177](#), [554](#)
PMIX_LOG_SOURCE, [223](#), [226](#), [228](#)
PMIX_LOG_STDERR, [223](#), [226](#), [228](#), [383](#)
PMIX_LOG_STDOUT, [223](#), [226](#), [228](#), [383](#)
PMIX_LOG_SYSLOG, [223](#), [226](#), [228](#), [383](#)
PMIX_LOG_SYSLOG_PRI, [223](#), [226](#), [228](#)
PMIX_LOG_TAG_OUTPUT, [224](#), [227](#), [228](#)
PMIX_LOG_TIMESTAMP, [223](#), [226](#), [228](#)
PMIX_LOG_TIMESTAMP_OUTPUT, [224](#), [227](#), [228](#)
PMIX_LOG_XML_OUTPUT, [224](#), [227](#), [228](#)
PMIX_MAPBY, [165](#), [171](#), [174](#), [175](#), [300](#), [364](#)
PMIX_MAX_PROCS, [100](#), [100–103](#), [298–301](#), [303](#), [330](#), [534](#)
PMIX_MAX_RESTARTS, [166](#), [172](#), [176](#), [366](#)
PMIX_MAX_VALUE, [331](#), [331](#), [537](#), [545](#)
PMIX_MERGE_STDERR_STDOUT, [166](#), [171](#), [175](#), [365](#)
PMIX_MIN_VALUE, [331](#), [331](#), [537](#), [545](#)
PMIX_MODEL_AFFINITY_POLICY, [62](#), [64](#)

PMIX_MODEL_CPU_TYPE, 62, 64
PMIX_MODEL_LIBRARY_NAME, 62, 64, 302, 326
PMIX_MODEL_LIBRARY_VERSION, 62, 64, 302, 327
PMIX_MODEL_NUM_CPUS, 62, 64
PMIX_MODEL_NUM_THREADS, 62, 64
PMIX_MODEL_PHASE_NAME, 64, 142
PMIX_MODEL_PHASE_TYPE, 64, 142
PMIX_MONITOR_APP_CONTROL, 217, 220, 221, 392
PMIX_MONITOR_CANCEL, 217, 219, 221, 392
PMIX_MONITOR_FILE, 217, 218, 220, 221, 392
PMIX_MONITOR_FILE_ACCESS, 218, 220, 222, 392
PMIX_MONITOR_FILE_CHECK_TIME, 218, 220, 222, 393
PMIX_MONITOR_FILE_DROPS, 218, 220, 222, 393
PMIX_MONITOR_FILE MODIFY, 218, 220, 222, 392
PMIX_MONITOR_FILE_SIZE, 217, 220, 221, 392
PMIX_MONITOR_HEARTBEAT, 217, 220, 221, 392
PMIX_MONITOR_HEARTBEAT_DROPS, 217, 220, 221, 392
PMIX_MONITOR_HEARTBEAT_TIME, 217, 220, 221, 392
PMIX_MONITOR_ID, 217, 219, 221, 392
PMIX_NO_OVERSUBSCRIBE, 166, 172, 176, 366
PMIX_NO_PROCS_ON_HEAD, 166, 171, 176, 366
PMIX_NODE_INFO, 73, 76, 81, 86, 99, 106, 303
PMIX_NODE_INFO_ARRAY, 298, 302, 307, 308, 312, 314, 318, 319, 545
PMIX_NODE_LIST, 100, 102, 104
PMIX_NODE_MAP, 100, 102, 103, 300, 311–313, 326, 327, 534
PMIX_NODE_MAP_RAW, 101, 547
PMIX_NODE_RANK, 105, 304, 435
PMIX_NODE_SIZE, 106, 302
PMIX_NODEID, 73, 76, 81, 84, 86, 89, 92, 99, 106, 106, 261, 271, 275, 298, 302–304, 307, 319, 545, 552
PMIX_NOHUP, 417, 422, 425, 547
PMIX_NOTIFY_COMPLETION, 177, 418, 437
PMIX_NOTIFY_JOB_EVENTS, 176, 417, 437, 554
PMIX_NOTIFY_PROC_ABNORMAL_TERMINATION, 177, 554
PMIX_NOTIFY_PROC_TERMINATION, 177, 554
PMIX_NPROC_OFFSET, 101, 300
PMIX_NSDIR, 102, 105, 303, 305
PMIX_NSPACE, 82–84, 87–92, 101, 297–299, 307, 312, 378, 379, 433, 438, 441, 543–545
PMIX_NUM_ALLOCATED_NODES, 100, 546
PMIX_NUM_NODES, 98, 100, 102, 103, 311, 312, 546
PMIX_NUM_SLOTS, 100, 101, 103
PMIX_OPTIONAL, 72, 75, 77, 116
PMIX_OUTPUT_TO_DIRECTORY, 176, 554

PMIX_OUTPUT_TO_FILE, 166, 171, [176](#), 365
PMIX_PACKAGE_RANK, [105](#), 305, 546
PMIX_PARENT_ID, [104](#), 363, 423
PMIX_PERSISTENCE, 119, [121](#), 121, 356, 462
PMIX_PERSONALITY, 165, 170, [174](#), 364
PMIX_PPR, 165, 170, [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](#)
PMIX_PRIMARY_SERVER, [415](#), 447, 547
PMIX_PROC_INFO, 81, 86, [99](#)
PMIX_PROC_INFO_ARRAY, 298, 303, [307](#), 313, 545, 557
PMIX_PROC_MAP, [101](#), 101, 102, 104, 300, 311, 312, 326, 327, 534
PMIX_PROC_MAP_RAW, [101](#), 547
PMIX_PROC_PID, 84, 89, [105](#)
PMIX_PROC_STATE_STATUS, 84, 89, [439](#)
PMIX_PROC_TERM_STATUS, [438](#), [439](#)
PMIX_PROCDIR, [105](#), 305
PMIX_PROCID, 81–83, 86–88, 91, [104](#), 177, 298, 307, 379, 418, 419, 437, 545, 555
PMIX_PROGRAMMING_MODEL, 62, [64](#), 302, 326
PMIX_PSET_MEMBERS, 231, [232](#), 552
PMIX_PSET_NAME, 231, [232](#), 552
PMIX_PSET_NAMES, 230, [232](#), 301, 552
PMIX_QUERY_ALLOC_STATUS, 83, 88, [91](#), 379
PMIX_QUERY_ATTRIBUTE_SUPPORT, 82, 87, [91](#), 95, 432, 543
PMIX_QUERY_AUTHORIZATIONS, 84, 89, [91](#)
PMIX_QUERY_AVAIL_SERVERS, [91](#), 412, 543
PMIX_QUERY_DEBUG_SUPPORT, 83, 88, [91](#), 378
PMIX_QUERY_GROUP_MEMBERSHIP, [235](#), 553
PMIX_QUERY_GROUP_NAMES, [235](#), 553
PMIX_QUERY_JOB_STATUS, 83, 88, [90](#), 378
PMIX_QUERY_LOCAL_ONLY, [91](#), 379
PMIX_QUERY_LOCAL_PROC_TABLE, 83, 88, 90, 378, 433, [441](#)
PMIX_QUERY_MEMORY_USAGE, 83, 88, [91](#), 379
PMIX_QUERY_NAMESPACE_INFO, [90](#), 543
PMIX_QUERY_NAMESPACES, 83, 88, [90](#), 378, 433
PMIX_QUERY_NUM_GROUPS, [235](#), 552
PMIX_QUERY_NUM_PSETS, [91](#), [232](#), 552
PMIX_QUERY_PROC_TABLE, 83, 88, 90, 378, 433, [441](#)
PMIX_QUERY_PSET_MEMBERSHIP, [91](#), [232](#), 552
PMIX_QUERY_PSET_NAMES, [91](#), [232](#), 552
PMIX_QUERY_QUALIFIERS, 84, [90](#), 90, 543

PMIX_QUERY_QUEUE_LIST, 83, 88, [90](#), 378
PMIX_QUERY_QUEUE_STATUS, 83, 88, [90](#), 378
PMIX_QUERY_REFRESH_CACHE, 81, 85, 86, [90](#), 95
PMIX_QUERY_REPORT_AVG, 83, 88, [91](#), 379
PMIX_QUERY_REPORT_MINMAX, 83, 88, [91](#), 379
PMIX_QUERY_RESULTS, 84, [90](#), 543
PMIX_QUERY_SPAWN_SUPPORT, 83, 88, [91](#), 378
PMIX_QUERY_STORAGE_LIST, [457](#), 561
PMIX_QUERY_SUPPORTED_KEYS, [90](#), 543
PMIX_QUERY_SUPPORTED_QUALIFIERS, [90](#), 543
PMIX_RANGE, 119, [121](#), 121, 124, 126, 132, 134, 139, 218, 235, 356, 359, 361, 375, 405, 406, 462, [552](#), 553
PMIX_RANK, 82, 83, 87, 88, 91, [104](#), 168, 173, 298, 303, 307, 379, 436, 440, 545
PMIX_RANKBY, 165, 171, [175](#), 300, 364
PMIX_REGISTER_CLEANUP, 210, 213, [215](#)
PMIX_REGISTER_CLEANUP_DIR, 210, 213, [216](#)
PMIX_REGISTER_NODATA, 297, [307](#)
PMIX_REINCARNATION, [105](#), 304, 547
PMIX_REPORT_BINDINGS, 166, 172, [176](#), 366
PMIX_REQUESTOR_IS_CLIENT, 363, [367](#)
PMIX_REQUESTOR_IS_TOOL, 363, [367](#)
PMIX_REQUIRED_KEY, 354, [355](#), 546
PMIX_RM_NAME, [100](#), 299
PMIX_RM_VERSION, [100](#), 299
PMIX_SEND_HEARTBEAT, 218, 220, [221](#)
PMIX_SERVER_ATTRIBUTES, 82, 87, [92](#), 96, 537, 544
PMIX_SERVER_ENABLE_MONITORING, 292, [294](#)
PMIX_SERVER_FUNCTIONS, 82, 87, 91, [92](#), 96, 543, 544
PMIX_SERVER_GATEWAY, 290, [294](#)
PMIX_SERVER_HOSTNAME, 299, [414](#)
PMIX_SERVER_INFO_ARRAY, [92](#), 92, 543
PMIX_SERVER_NSPACE, 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](#), 544, 549
PMIX_SERVER_SESSION_SUPPORT, 290, [294](#), 544
PMIX_SERVER_SHARE_TOPOLOGY, 292, [293](#), 544
PMIX_SERVER_START_TIME, [294](#), 544
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, 84, 89, 411, 412, [414](#), 442, 446

PMIX_SESSION_ID, [99](#), [99](#), [101](#), [108](#), [297](#), [298](#), [307](#), [311](#), [438](#), [545](#)
PMIX_SESSION_INFO, [73](#), [76](#), [81](#), [86](#), [98](#), [100](#), [108](#), [298](#), [300](#), [326](#)
PMIX_SESSION_INFO_ARRAY, [297](#), [298](#), [307](#), [308](#), [311](#), [532](#)
PMIX_SET_ENVAR, [166](#), [172](#), [178](#)
PMIX_SET_SESSION_CWD, [164](#), [170](#), [175](#), [363](#)
PMIX_SETUP_APP_ALL, [325](#), [328](#)
PMIX_SETUP_APP_ENVARS, [325](#), [328](#)
PMIX_SETUP_APP_NONENVARS, [325](#), [328](#)
PMIX_SINGLE_LISTENER, [61](#), [291](#), [293](#)
PMIX_SOCKET_MODE, [61](#), [291](#), [293](#), [443](#)
PMIX_SPAWN_TOOL, [168](#), [173](#), [176](#), [424](#)
PMIX_SPAWNED, [105](#), [304](#), [363](#)
PMIX_STDIN_TGT, [165](#), [171](#), [175](#), [364](#)
PMIX_STORAGE_ACCESS_TYPE, [458](#), [562](#)
PMIX_STORAGE_ACCESSIBILITY, [457](#), [561](#)
PMIX_STORAGE_BW_CUR, [458](#), [458](#), [561](#), [562](#)
PMIX_STORAGE_BW_MAX, [458](#), [561](#)
PMIX_STORAGE_CAPACITY_LIMIT, [457](#), [561](#)
PMIX_STORAGE_CAPACITY_USED, [457](#), [561](#)
PMIX_STORAGE_ID, [457](#), [457](#), [560](#), [561](#)
PMIX_STORAGE_IOPS_CUR, [458](#), [458](#), [561](#), [562](#)
PMIX_STORAGE_IOPS_MAX, [458](#), [561](#)
PMIX_STORAGE_MEDIUM, [457](#), [560](#)
PMIX_STORAGE_MINIMAL_XFER_SIZE, [457](#), [561](#)
PMIX_STORAGE_OBJECT_LIMIT, [457](#), [561](#)
PMIX_STORAGE_OBJECTS_USED, [457](#), [561](#)
PMIX_STORAGE_PATH, [457](#), [560](#)
PMIX_STORAGE_PERSISTENCE, [457](#), [561](#)
PMIX_STORAGE_SUGGESTED_XFER_SIZE, [458](#), [458](#), [561](#), [562](#)
PMIX_STORAGE_TYPE, [457](#), [560](#)
PMIX_STORAGE_VERSION, [457](#), [560](#)
PMIX_SWITCH_PEERS, [275](#), [275](#), [550](#)
PMIX_SYSTEM_TMPDIR, [290](#), [294](#), [409](#), [411](#)
PMIX_TAG_OUTPUT, [165](#), [171](#), [175](#), [365](#)
PMIX_TCP_DISABLE_IPV4, [61](#), [64](#), [291](#), [443](#)
PMIX_TCP_DISABLE_IPV6, [61](#), [64](#), [291](#), [443](#)
PMIX_TCP_IF_EXCLUDE, [61](#), [64](#), [291](#), [443](#)
PMIX_TCP_IF_INCLUDE, [61](#), [64](#), [291](#), [443](#)
PMIX_TCP_IPV4_PORT, [61](#), [64](#), [291](#), [443](#)
PMIX_TCP_IPV6_PORT, [61](#), [64](#), [291](#), [443](#)
PMIX_TCP_REPORT_URI, [61](#), [63](#), [291](#), [443](#)
PMIX_TCP_URI, [64](#), [411](#), [412](#), [442](#), [446](#)
PMIX_TDIR_RMCLEAN, [100](#), [301](#)

PMIX_THREADING_MODEL, [62](#), [64](#)
PMIX_TIME_REMAINING, [78](#), [84](#), [89](#), [91](#), [379](#)
PMIX_TIMEOUT, [4](#), [68](#), [71](#), [74](#), [77](#), [78](#), [107](#), [116](#), [117](#), [119](#), [121](#), [124](#), [126](#), [132](#), [134](#), [183](#), [185](#),
 [186](#), [188](#), [234](#), [238](#), [240](#), [243](#), [244](#), [246](#), [248](#), [251](#), [253](#), [255](#), [283](#), [284](#), [286](#), [288](#), [351](#), [355](#),
 [357](#), [359](#), [361](#), [366](#), [368](#), [370](#), [394](#), [397](#), [415](#), [449](#), [549](#)
PMIX_TIMEOUT_REPORT_STATE, [176](#), [554](#)
PMIX_TIMEOUT_STACKTRACES, [176](#), [554](#)
PMIX_TIMESTAMP_OUTPUT, [165](#), [171](#), [175](#), [365](#)
PMIX_TMPDIR, [100](#), [102](#), [303](#)
PMIX_TOOL_ATTACHMENT_FILE, [411](#), [412](#), [415](#), [442](#), [446](#), [547](#)
PMIX_TOOL_ATTRIBUTES, [82](#), [87](#), [92](#), [96](#), [537](#), [544](#)
PMIX_TOOL_CONNECT_OPTIONAL, [415](#), [547](#)
PMIX_TOOL_DO_NOT_CONNECT, [411](#), [415](#), [442](#), [444](#)
PMIX_TOOL_FUNCTIONS, [82](#), [87](#), [91](#), [92](#), [96](#), [543](#), [544](#)
PMIX_TOOL_NSPACE, [380](#), [410](#), [413](#), [442](#), [444](#)
PMIX_TOOL_RANK, [380](#), [410](#), [414](#), [442](#), [444](#)
PMIX_TOPOLOGY2, [292](#), [293](#), [544](#), [557](#)
PMIX_UNIV_SIZE, [8](#), [99](#), [298](#), [311](#), [531](#), [532](#), [534](#)
PMIX_UNSET_ENVAR, [167](#), [172](#), [178](#)
PMIX_USERID, [82](#), [87](#), [119](#), [120](#), [124](#), [126](#), [132](#), [133](#), [201](#), [204](#), [209](#), [212](#), [217](#), [219](#), [223](#), [226](#), [282](#),
 [284](#), [286](#), [287](#), [356](#)–[359](#), [361](#), [363](#), [371](#), [378](#), [380](#), [382](#), [383](#), [385](#), [389](#), [392](#), [394](#), [396](#), [398](#),
 [399](#), [403](#)
PMIX_USOCK_DISABLE, [61](#), [291](#), [293](#)
PMIX_VERSION_INFO, [381](#), [382](#)
PMIX_WAIT, [76](#), [78](#), [124](#), [126](#), [359](#)
PMIX_WAIT_FOR_CONNECTION, [415](#), [449](#), [549](#)
PMIX_WDIR, [164](#), [170](#), [175](#), [301](#), [363](#)

PMIX_ALLOC_NETWORK
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_ENDPTS
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_ENDPTS_NODE
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_ID
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_PLANE
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_QOS
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_SEC_KEY
 Deprecated, [557](#)
PMIX_ALLOC_NETWORK_TYPE
 Deprecated, [557](#)

PMIX_ARCH
 Deprecated, [536](#)
 Removed, [559](#)

PMIX_COLLECTIVE_ALGO
 Deprecated, [536](#)
 Removed, [558](#)

PMIX_COLLECTIVE_ALGO_REQD
 Deprecated, [533](#)
 Removed, [558](#)

PMIX_DEBUG_JOB
 Deprecated, [557](#)

PMIX_DSTPATH
 Deprecated, [536](#)
 Removed, [558](#)

PMIX_ERROR_GROUP_ABORT
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_COMM
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_GENERAL
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_LOCAL
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_MIGRATE
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_NODE
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_RESOURCE
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_GROUP_SPAWN
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_HANDLER_ID
 Deprecated, [531](#)
 Removed, [533](#)

PMIX_ERROR_NAME
 Deprecated, [531](#)

Removed, [533](#)
PMIX_HWLOC_HOLE_KIND
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_HWLOC_SHARE_TOPO
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_HWLOC_SHMEM_ADDR
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_HWLOC_SHMEM_FILE
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_HWLOC_SHMEM_SIZE
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_HWLOC_XML_V1
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_HWLOC_XML_V2
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_LOCAL_TOPO
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_LOCALITY
 Deprecated, [557](#)
PMIX_MAP_BLOB
 Deprecated, [536](#)
 Removed, [559](#)
PMIX_MAPPER
 Deprecated, [536](#)
 Removed, [559](#)
PMIX_NON_PMI
 Deprecated, [536](#)
 Removed, [559](#)
PMIX_PROC_BLOB
 Deprecated, [536](#)
 Removed, [558](#)
PMIX_PROC_DATA
 Deprecated, [557](#)
PMIX_PROC_URI
 Deprecated, [536](#)

Removed, [559](#)

PMIX_RECONNECT_SERVER

Deprecated, [557](#)

PMIX_TOPOLOGY

Deprecated, [557](#)

PMIX_TOPOLOGY_FILE

Deprecated, [536](#)

Removed, [558](#)

PMIX_TOPOLOGY_SIGNATURE

Deprecated, [537](#)

Removed, [558](#)

PMIX_TOPOLOGY_XML

Deprecated, [537](#)

Removed, [558](#)