Process Management Interface for Exascale (PMIx) Standard

Version 4.0 (Draft)

Created on June 15, 2020

Unofficial Draft

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

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

Copyright © 2018-2019 PMIx Standard Review Board.
Permission to copy without fee all or part of this material is granted, provided the PMIx Standard Review Board copyright notice and the title of this document appear, and notice is given that copying is by permission of PMIx Standard Review Board.

This page intentionally left blank

Contents

1. Introduction **1**

1.1. Charter . . . 2
1.2. PMIx Standard Overview . . . 2
1.2.1. Who should use the standard? . . . 2
1.2.2. What is defined in the standard? . . . 3
1.2.3. What is *not* defined in the standard? . . . 3
1.2.4. General Guidance for PMIx Users and Implementors . . . 4
1.3. PMIx Architecture Overview . . . 4
1.3.1. The PMIx Reference Implementation (PRI) . . . 6
1.3.2. The PMIx Reference RunTime Environment (PRRTE) . . . 7
1.4. Organization of this document . . . 7
1.5. Version 1.0: June 12, 2015 . . . 8
1.6. Version 2.0: Sept. 2018 . . . 9
1.7. Version 2.1: Dec. 2018 . . . 9
1.8. Version 2.2: Jan 2019 . . . 10
1.9. Version 3.0: Dec. 2018 . . . 11
1.10. Version 3.1: Jan. 2019 . . . 11
1.11. Version 3.2: Oct. 2019 . . . 12
1.12. Version 4.0: June 2019 . . . 12

2. PMIx Terms and Conventions **14**

2.1. Notational Conventions . . . 16
2.2. Semantics . . . 17
2.3. Naming Conventions . . . 18
2.4. Procedure Conventions . . . 18
2.5. Standard vs Reference Implementation . . . 19

3. Data Structures and Types **20**

3.1. Constants . . . 21
3.1.1. PMIx Error Constants . . . 22

Unofficial Draft

3.1.2. Macros for use with PMIx constants . . . 26
3.2. Data Types . . . 26
3.2.1. Key Structure . . . 27
3.2.2. Namespace Structure . . . 28
3.2.3. Rank Structure . . . 29
3.2.4. Process Structure . . . 29
3.2.5. Process structure support macros . . . 29
3.2.6. Process State Structure . . . 31
3.2.7. Process Information Structure . . . 32
3.2.8. Process Information Structure support macros . . . 33
3.2.9. Scope of Put Data . . . 34
3.2.10. Range of Published Data . . . 35
3.2.11. Data Persistence Structure . . . 35
3.2.12. Data Array Structure . . . 36
3.2.13. Data array structure support macros . . . 36
3.2.14. Value Structure . . . 37
3.2.15. Value structure support macros . . . 38
3.2.16. Info Structure . . . 42
3.2.17. Info structure support macros . . . 42
3.2.18. Info Type Directives . . . 45
3.2.19. Info Directive support macros . . . 46
3.2.20. Job Allocation Directives . . . 48
3.2.21. IO Forwarding Channels . . . 48
3.2.22. Environmental Variable Structure . . . 48
3.2.23. Environmental variable support macros . . . 49
3.2.24. Lookup Returned Data Structure . . . 50
3.2.25. Lookup data structure support macros . . . 50
3.2.26. Application Structure . . . 53
3.2.27. App structure support macros . . . 54
3.2.28. Query Structure . . . 55
3.2.29. Query structure support macros . . . 55
3.2.30. Attribute registration structure . . . 57
3.2.31. Attribute registration structure support macros . . . 58

3.2.32. PMIx Group Directives . . . 60
3.2.33. Byte Object Type . . . 60
3.2.34. Byte object support macros . . . 60
3.2.35. Data Array Structure . . . 62
3.2.36. Data array support macros . . . 62
3.2.37. Argument Array Macros . . . 63
3.2.38. Set Environment Variable . . . 67
3.3. Generalized Data Types Used for Packing/Unpacking . . . 68
3.4. Reserved attributes . . . 69
3.4.1. Initialization attributes . . . 70
3.4.2. Tool-related attributes . . . 70
3.4.3. Identification attributes . . . 71
3.4.4. Programming model attributes . . . 72
3.4.5. UNIX socket rendezvous socket attributes . . . 72
3.4.6. TCP connection attributes . . . 73
3.4.7. Global Data Storage (GDS) attributes . . . 73
3.4.8. General process-level attributes . . . 73
3.4.9. Scratch directory attributes . . . 74
3.4.10. Relative Rank Descriptive Attributes . . . 74
3.4.11. Information retrieval attributes . . . 75
3.4.12. Information storage attributes . . . 76
3.4.13. Size information attributes . . . 77
3.4.14. Memory information attributes . . . 78
3.4.15. Topology information attributes . . . 78
3.4.16. Request-related attributes . . . 79
3.4.17. Server-to-PMIx library attributes . . . 80
3.4.18. Server-to-Client attributes . . . 81
3.4.19. Event handler registration and notification attributes . . . 81
3.4.20. Fault tolerance attributes . . . 82
3.4.21. Spawn attributes . . . 83
3.4.22. Query attributes . . . 85
3.4.23. Log attributes . . . 86
3.4.24. Debugger attributes . . . 87

3.4.25. Resource manager attributes . . . 88
3.4.26. Environment variable attributes . . . 88
3.4.27. Job Allocation attributes . . . 89
3.4.28. Job control attributes . . . 90
3.4.29. Monitoring attributes . . . 91
3.4.30. Security attributes . . . 92
3.4.31. IO Forwarding attributes . . . 92
3.4.32. Application setup attributes . . . 93
3.4.33. Attribute support level attributes . . . 93
3.4.34. Descriptive attributes . . . 94
3.4.35. Process group attributes . . . 94
3.5. Callback Functions . . . 95
3.5.1. Release Callback Function . . . 95
3.5.2. Modex Callback Function . . . 96
3.5.3. Spawn Callback Function . . . 96
3.5.4. Op Callback Function . . . 97
3.5.5. Lookup Callback Function . . . 98
3.5.6. Value Callback Function . . . 98
3.5.7. Info Callback Function . . . 99
3.5.8. Event Handler Registration Callback Function . . . 99
3.5.9. Notification Handler Completion Callback Function . . . 100
3.5.10. Notification Function . . . 101
3.5.11. Server Setup Application Callback Function . . . 103
3.5.12. Server Direct Modex Response Callback Function . . . 104
3.5.13. PMIx Client Connection Callback Function . . . 105
3.5.14. PMIx Tool Connection Callback Function . . . 105
3.5.15. Credential callback function . . . 106
3.5.16. Credential validation callback function . . . 107
3.5.17. IOF delivery function . . . 108
3.5.18. IOF and Event registration function . . . 109
3.6. Constant String Functions . . . 109

4. Initialization and Finalization **112**
4.1. Query . . . 112
4.1.1. **PMIx_Initialized** . . . 112
4.1.2. **PMIx_Get_version** . . . 113
4.2. Client Initialization and Finalization . . . 113
4.2.1. **PMIx_Init** . . . 113
4.2.2. **PMIx_Finalize** . . . 116
4.3. Tool Initialization and Finalization . . . 116
4.3.1. **PMIx_tool_init** . . . 116
4.3.2. **PMIx_tool_finalize** . . . 120
4.3.3. **PMIx_tool_connect_to_server** . . . 120
4.4. Server Initialization and Finalization . . . 122
4.4.1. **PMIx_server_init** . . . 122
4.4.2. **PMIx_server_finalize** . . . 124

5. Key/Value Management **125**
5.1. Setting and Accessing Key/Value Pairs . . . 125
5.1.1. **PMIx_Put** . . . 125
5.1.2. **PMIx_Get** . . . 126
5.1.3. **PMIx_Get_nb** . . . 129
5.1.4. **PMIx_Store_internal** . . . 132
5.1.5. Accessing information: examples . . . 133
5.2. Exchanging Key/Value Pairs . . . 137
5.2.1. **PMIx_Commit** . . . 138
5.2.2. **PMIx_Fence** . . . 138
5.2.3. **PMIx_Fence_nb** . . . 140
5.3. Publish and Lookup Data . . . 143
5.3.1. **PMIx_Publish** . . . 143
5.3.2. **PMIx_Publish_nb** . . . 145
5.3.3. **PMIx_Lookup** . . . 146
5.3.4. **PMIx_Lookup_nb** . . . 149
5.3.5. **PMIx_Unpublish** . . . 150
5.3.6. **PMIx_Unpublish_nb** . . . 152

6. Process Management **154**
6.1. Abort . . . 154
6.1.1. **PMIx_Abort** . . . 154
6.2. Process Creation . . . 155
6.2.1. **PMIx_Spawn** . . . 155
6.2.2. **PMIx_Spawn_nb** . . . 160
6.3. Connecting and Disconnecting Processes . . . 164
6.3.1. **PMIx_Connect** . . . 165
6.3.2. **PMIx_Connect_nb** . . . 167
6.3.3. **PMIx_Disconnect** . . . 169
6.3.4. **PMIx_Disconnect_nb** . . . 171
6.4. IO Forwarding . . . 173
6.4.1. **PMIx_IOF_pull** . . . 174
6.4.2. **PMIx_IOF_deregister** . . . 176
6.4.3. **PMIx_IOF_push** . . . 177

7. Job Management and Reporting **180**
7.1. Query . . . 180
7.1.1. **PMIx_Resolve_peers** . . . 180
7.1.2. **PMIx_Resolve_nodes** . . . 181
7.1.3. **PMIx_Query_info** . . . 182
7.1.4. **PMIx_Query_info_nb** . . . 186
7.2. Allocation Requests . . . 192
7.2.1. **PMIx_Allocation_request** . . . 193
7.2.2. **PMIx_Allocation_request_nb** . . . 196
7.3. Job Control . . . 199
7.3.1. **PMIx_Job_control** . . . 199
7.3.2. **PMIx_Job_control_nb** . . . 202
7.4. Process and Job Monitoring . . . 205
7.4.1. **PMIx_Process_monitor** . . . 205
7.4.2. **PMIx_Process_monitor_nb** . . . 207
7.4.3. **PMIx_Heartbeat** . . . 209
7.5. Logging . . . 210
7.5.1. **PMIx_Log** . . . 210

7.5.2. **PMIx_Log_nb** . . . 212

8. Event Notification **216**

8.1. Notification and Management . . . 216

8.1.1. **PMIx_Register_event_handler** . . . 218

8.1.2. **PMIx_Deregister_event_handler** . . . 221

8.1.3. **PMIx_Notify_event** . . . 222

9. Data Packing and Unpacking **226**

9.1. Data Buffer Type . . . 226

9.2. Support Macros . . . 227

9.2.1. **PMIX_DATA_BUFFER_CREATE** . . . 227

9.2.2. **PMIX_DATA_BUFFER_RELEASE** . . . 227

9.2.3. **PMIX_DATA_BUFFER_CONSTRUCT** . . . 227

9.2.4. **PMIX_DATA_BUFFER_DESTRUCT** . . . 228

9.2.5. **PMIX_DATA_BUFFER_LOAD** . . . 228

9.2.6. **PMIX_DATA_BUFFER_UNLOAD** . . . 229

9.3. General Routines . . . 229

9.3.1. **PMIx_Data_pack** . . . 229

9.3.2. **PMIx_Data_unpack** . . . 231

9.3.3. **PMIx_Data_copy** . . . 233

9.3.4. **PMIx_Data_print** . . . 233

9.3.5. **PMIx_Data_copy_payload** . . . 234

10. Security **236**

10.1. Obtaining Credentials . . . 237

10.1.1. **PMIx_Get_credential** . . . 237

10.1.2. **PMIx_Get_credential_nb** . . . 238

10.2. Validating Credentials . . . 240

10.2.1. **PMIx_Validate_credential** . . . 240

10.2.2. **PMIx_Validate_credential_nb** . . . 242

11. Server-Specific Interfaces **245**

11.1. Server Support Functions . . . 245

11.1.1. **PMIx_generate_regex** . . . 245

11.1.2. **PMIx_generate_ppn** . . . 246
11.1.3. **PMIx_server_register_nspace** . . . 247
11.1.4. **PMIx_server_deregister_nspace** . . . 260
11.1.5. **PMIx_server_register_client** . . . 262
11.1.6. **PMIx_server_deregister_client** . . . 263
11.1.7. **PMIx_server_setup_fork** . . . 264
11.1.8. **PMIx_server_dmodex_request** . . . 264
11.1.9. **PMIx_server_setup_application** . . . 266
11.1.10. **PMIx_Register_attributes** . . . 268
11.1.11. **PMIx_server_setup_local_support** . . . 270
11.1.12. **PMIx_server_IOF_deliver** . . . 271
11.1.13. **PMIx_server_collect_inventory** . . . 272
11.1.14. **PMIx_server_deliver_inventory** . . . 273
11.2. Server Function Pointers . . . 275
11.2.1. **pmix_server_module_t** Module . . . 275
11.2.2. **pmix_server_client_connected_fn_t** . . . 276
11.2.3. **pmix_server_client_finalized_fn_t** . . . 278
11.2.4. **pmix_server_abort_fn_t** . . . 279
11.2.5. **pmix_server_fencenb_fn_t** . . . 280
11.2.6. **pmix_server_dmodex_req_fn_t** . . . 284
11.2.7. **pmix_server_publish_fn_t** . . . 285
11.2.8. **pmix_server_lookup_fn_t** . . . 287
11.2.9. **pmix_server_unpublish_fn_t** . . . 289
11.2.10. **pmix_server_spawn_fn_t** . . . 291
11.2.11. **pmix_server_connect_fn_t** . . . 296
11.2.12. **pmix_server_disconnect_fn_t** . . . 298
11.2.13. **pmix_server_register_events_fn_t** . . . 299
11.2.14. **pmix_server_deregister_events_fn_t** . . . 301
11.2.15. **pmix_server_notify_event_fn_t** . . . 303
11.2.16. **pmix_server_listener_fn_t** . . . 304
11.2.17. **pmix_server_query_fn_t** . . . 305
11.2.18. **pmix_server_tool_connection_fn_t** . . . 307
11.2.19. **pmix_server_log_fn_t** . . . 308

11.2.20. **pmix_server_alloc_fn_t** . . . 310
11.2.21. **pmix_server_job_control_fn_t** . . . 313
11.2.22. **pmix_server_monitor_fn_t** . . . 316
11.2.23. **pmix_server_get_cred_fn_t** . . . 319
11.2.24. **pmix_server_validate_cred_fn_t** . . . 320
11.2.25. **pmix_server_iof_fn_t** . . . 322
11.2.26. **pmix_server_stdin_fn_t** . . . 325
11.2.27. **pmix_server_grp_fn_t** . . . 326

12. Scheduler-Specific Interfaces **330**
12.1. Scheduler Support Datatypes . . . 330
12.1.1. Fabric registration structure . . . 330
12.1.2. Scheduler Support Error Constants . . . 331
12.1.3. Scheduler Support Attributes . . . 331
12.2. Scheduler Support Functions . . . 331
12.2.1. **PMIx_server_register_fabric** . . . 331
12.2.2. **PMIx_server_deregister_fabric** . . . 332
12.2.3. **PMIx_server_get_vertex_info** . . . 333
12.2.4. **PMIx_server_get_index** . . . 333

13. Process Sets and Groups **335**
13.1. Process Sets . . . 335
13.2. Process Groups . . . 336
13.2.1. Group Operation Constants . . . 338
13.2.2. **PMIx_Group_construct** . . . 339
13.2.3. **PMIx_Group_construct_nb** . . . 342
13.2.4. **PMIx_Group_destruct** . . . 345
13.2.5. **PMIx_Group_destruct_nb** . . . 347
13.2.6. **PMIx_Group_invite** . . . 349
13.2.7. **PMIx_Group_invite_nb** . . . 353
13.2.8. **PMIx_Group_join** . . . 355
13.2.9. **PMIx_Group_join_nb** . . . 358
13.2.10. **PMIx_Group_leave** . . . 360
13.2.11. **PMIx_Group_leave_nb** . . . 361

14. Network Coordinates **363**
14.1. Network Coordinate Datatypes . . . 363
14.1.1. Network Coordinate Structure . . . 363
14.1.2. Network Coordinate Support Macros . . . 364
14.1.3. Network Coordinate Views . . . 365
14.1.4. Network Coordinate Error Constants . . . 366
14.1.5. Network Descriptive Attributes . . . 366

A. Python Bindings **368**
A.1. Design Considerations . . . 368
A.1.1. Error Codes vs Python Exceptions . . . 368
A.1.2. Representation of Structured Data . . . 368
A.2. Datatype Definitions . . . 369
A.2.1. Example . . . 374
A.3. Function Definitions . . . 374
A.3.1. IOF Delivery Function . . . 374
A.3.2. Event Handler . . . 375
A.3.3. Server Module Functions . . . 376
A.4. PMIxClient . . . 388
A.4.1. Client.init . . . 388
A.4.2. Client.initialized . . . 389
A.4.3. Client.get_version . . . 389
A.4.4. Client.finalize . . . 389
A.4.5. Client.abort . . . 390
A.4.6. Client.store_internal . . . 390
A.4.7. Client.put . . . 391
A.4.8. Client.commit . . . 391
A.4.9. Client.fence . . . 391
A.4.10. Client.get . . . 392
A.4.11. Client.publish . . . 392
A.4.12. Client.lookup . . . 393
A.4.13. Client.unpublish . . . 393
A.4.14. Client.spawn . . . 394
A.4.15. Client.connect . . . 394

A.4.16. Client.disconnect . . . 395
A.4.17. Client.resolve_peers . . . 395
A.4.18. Client.resolve_nodes . . . 396
A.4.19. Client.query . . . 396
A.4.20. Client.log . . . 397
A.4.21. Client.allocate . . . 397
A.4.22. Client.job_ctrl . . . 398
A.4.23. Client.monitor . . . 398
A.4.24. Client.get_credential . . . 399
A.4.25. Client.validate_credential . . . 399
A.4.26. Client.group_construct . . . 400
A.4.27. Client.group_invite . . . 400
A.4.28. Client.group_join . . . 401
A.4.29. Client.group_leave . . . 402
A.4.30. Client.group_destruct . . . 402
A.4.31. Client.register_event_handler . . . 402
A.4.32. Client.deregister_event_handler . . . 403
A.4.33. Client.notify_event . . . 403
A.4.34. Client.error_string . . . 404
A.4.35. Client.proc_state_string . . . 404
A.4.36. Client.scope_string . . . 405
A.4.37. Client.persistence_string . . . 405
A.4.38. Client.data_range_string . . . 406
A.4.39. Client.info_directives_string . . . 406
A.4.40. Client.data_type_string . . . 406
A.4.41. Client.alloc_directive_string . . . 407
A.4.42. Client.iof_channel_string . . . 407
A.5. PMIxServer . . . 408
A.5.1. Server.init . . . 408
A.5.2. Server.finalize . . . 408
A.5.3. Server.generate_regex . . . 409
A.5.4. Server.generate_ppn . . . 409
A.5.5. Server.register_nspace . . . 409

A.5.6. Server.deregister_nspace . . . 410
A.5.7. Server.register_client . . . 410
A.5.8. Server.deregister_client . . . 411
A.5.9. Server.setup_fork . . . 411
A.5.10. Server.dmodex_request . . . 412
A.5.11. Server.setup_application . . . 412
A.5.12. Server.register_attributes . . . 413
A.5.13. Server.setup_local_support . . . 413
A.5.14. Server.iof_deliver . . . 414
A.5.15. Server.collect_inventory . . . 414
A.5.16. Server.deliver_inventory . . . 415
A.6. PMIxTool . . . 415
A.6.1. Tool.init . . . 415
A.6.2. Tool.finalize . . . 416
A.6.3. Tool.connect_to_server . . . 416
A.6.4. Tool.iof_pull . . . 417
A.6.5. Tool.iof_deregister . . . 417
A.6.6. Tool.iof_push . . . 418
A.7. Example Usage . . . 418
A.7.1. Python Client . . . 419
A.7.2. Python Server . . . 421

B. Acknowledgements **425**
B.1. Version 3.0 . . . 425
B.2. Version 2.0 . . . 426
B.3. Version 1.0 . . . 427

Bibliography **428**

Index **429**

Index of APIs **430**

Index of Support Macros **435**

Index of Data Structures **438**

Index of Constants 439

Index of Attributes 445

CHAPTER 1

Introduction

The Process Management Interface (PMI) has been used for quite some time as a means of exchanging wireup information needed for inter-process communication. Two versions (PMI-1 and PMI-2) have been released as part of the MPICH effort, with PMI-2 demonstrating better scaling properties than its PMI-1 predecessor. However, two significant challenges face the High Performance Computing (HPC) community as it continues to move towards machines capable of exaflop and higher performance levels:

- the physical scale of the machines, and the corresponding number of total processes they support, is expected to reach levels approaching 1 million processes executing across 100 thousand nodes. Prior methods for initiating applications relied on exchanging communication endpoint information between the processes, either directly or in some form of hierarchical collective operation. Regardless of the specific mechanism employed, the exchange across such large applications would consume considerable time, with estimates running in excess of 5-10 minutes; and
- whether it be hybrid applications that combine OpenMP threading operations with MPI, or application-steered workflow computations, the HPC community is experiencing an unprecedented wave of new approaches for computing at exascale levels. One common thread across the proposed methods is an increasing need for orchestration between the application and the system management software stack (SMS) comprising the scheduler (a.k.a. the workload manager (WLM)), the resource manager (RM), global file system, fabric, and other subsystems. The lack of available support for application-to-SMS integration has forced researchers to develop "virtual" environments that hide the SMS behind a customized abstraction layer, but this results in considerable duplication of effort and a lack of portability.

Process Management Interface - Exascale (PMIx) represents an attempt to resolve these questions by providing an extended version of the PMI definitions specifically designed to support clusters up to exascale and larger sizes. The overall objective of the project is not to branch the existing definitions – in fact, PMIx fully supports both of the existing PMI-1 and PMI-2 Application Programming Interfaces (APIs) – but rather to:

a) add flexibility to the existing APIs by adding an array of key-value “attribute” pairs to each API signature that allows implementers to customize the behavior of the API as future needs emerge without having to alter or create new variants of it;
b) add new APIs that provide extended capabilities such as asynchronous event notification plus dynamic resource allocation and management;

Unofficial Draft

c) establish a collaboration between SMS subsystem providers including resource manager, fabric, file system, and programming library developers to define integration points between the various subsystems as well as agreed upon definitions for associated APIs, attribute names, and data types;
d) form a standards-like body for the definitions; and
e) provide a reference implementation of the PMIx standard.

Complete information about the PMIx standard and affiliated projects can be found at the PMIx web site: https://pmix.org

1.1 Charter

The charter of the PMIx community is to:

- Define a set of agnostic APIs (not affiliated with any specific programming model or code base) to support interactions between application processes and the SMS.
- Develop an open source (non-copy-left licensed) standalone "reference" library implementation to facilitate adoption of the PMIx standard.
- Retain transparent backward compatibility with the existing PMI-1 and PMI-2 definitions, any future PMI releases, and across all PMIx versions.
- Support the "Instant On" initiative for rapid startup of applications at exascale and beyond.
- Work with the HPC community to define and implement new APIs that support evolving programming model requirements for application interactions with the SMS.

Participation in the PMIx community is open to anyone, and not restricted to only code contributors to the reference implementation.

1.2 PMIx Standard Overview

The PMIx Standard defines and describes the interface developed by the PMIx Reference Implementation (PRI). Much of this document is specific to the PMIx Reference Implementation (PRI)'s design and implementation. Specifically the standard describes the functionality provided by the PRI, and what the PRI requires of the clients and resource managers (RMs) that use it's interface.

1.2.1 Who should use the standard?

The PMIx Standard informs PMIx clients and RMs of the syntax and semantics of the PMIx APIs.

PMIx clients (e.g., tools, Message Passing Environment (MPE) libraries) can use this standard to understand the set of attributes provided by various APIs of the PRI and their intended behavior.

Additional information about the rationale for the selection of specific interfaces and attributes is also provided.

PMIx-enabled RMs can use this standard to understand the expected behavior required of them when they support various interfaces/attributes. In addition, optional features and suggestions on behavior are also included in the discussion to help guide RM design and implementation.

1.2.2 What is defined in the standard?

The PMIx Standard defines and describes the interface developed by the PMIx Reference Implementation (PRI). It defines the set of attributes that the PRI supports; the set of attributes that are required of a RM to support, for a given interface; and the set of optional attributes that an RM may choose to support, for a given interface.

1.2.3 What is *not* defined in the standard?

No standards body can require an implementer to support something in their standard, and PMIx is no different in that regard. While an implementer of the PMIx library itself must at least include the standard PMIx headers and instantiate each function, they are free to return "not supported" for any function they choose not to implement.

This also applies to the host environments. Resource managers and other system management stack components retain the right to decide on support of a particular function. The PMIx community continues to look at ways to assist SMS implementers in their decisions by highlighting functions that are critical to basic application execution (e.g., **`PMIx_Get`**), while leaving flexibility for tailoring a vendor's software for their target market segment.

One area where this can become more complicated is regarding the attributes that provide information to the client process and/or control the behavior of a PMIx standard API. For example, the **`PMIX_TIMEOUT`** attribute can be used to specify the time (in seconds) before the requested operation should time out. The intent of this attribute is to allow the client to avoid "hanging" in a request that takes longer than the client wishes to wait, or may never return (e.g., a **`PMIx_Fence`** that a blocked participant never enters).

If an application (for example) truly relies on the **`PMIX_TIMEOUT`** attribute in a call to **`PMIx_Fence`** , it should set the required flag in the **`pmix_info_t`** for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, ignoring it if support is not available.

It is therefore critical that users and application implementers:

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

While a PMIx library implementer, or an SMS component server, may choose to support a particular PMIx API, they are not required to support every attribute that might apply to it. This would pose a significant barrier to entry for an implementer as there can be a broad range of applicable attributes to a given API, at least some of which may rarely be used. The PMIx community is attempting to help differentiate the attributes by indicating those that are generally used (and therefore, of higher importance to support) vs those that a "complete implementation" would support.

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

The flip side of that statement is also true: Users who find that their current vendor does not support a function or attribute they require may raise that concern with their vendor and request that the implementation be expanded. Alternatively, users may wish to utilize the PMIx-based Reference RunTime Environment (PRRTE) as a "shim" between their application and the host environment as it might provide the desired support until the vendor can respond. Finally, in the extreme, one can exploit the portability of PMIx-based applications to change vendors.

1.2.4 General Guidance for PMIx Users and Implementors

The PMIx Standard defines the behavior of the PMIx Reference Implementation (PRI). A complete system harnessing the PMIx interface requires an agreement between the PMIx client, be it a tool or library, and the PMIx-enabled RM. The PRI acts as an intermediary between these two entities by providing a standard API for the exchange of requests and responses. The degree to which the PMIx client and the PMIx-enabled RM may interact needs to be defined by those developer communities. The PMIx standard can be used to define the specifics of this interaction.

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

PMIx-enabled RMs may choose to implement a subset of the PMIx standard and/or define attributes beyond those defined herein. PMIx-enabled RMs are strongly advised to define a document itemizing the PMIx interfaces and associated attributes they support, with any annotations about behavior limitations. The PMIx standard cannot define this list for all given PMIx-enabled RMs, but such a list is valuable to PMIx clients desiring to support a broad range of PMIx-enabled RMs.

1.3 PMIx Architecture Overview

This section presents a brief overview of the PMIx Architecture [1]. Note that this is a conceptual model solely used to help guide the standards process — it does not represent a design requirement

on any PMIx implementation. Instead, the model is used by the PMIx community as a sounding board for evaluating proposed interfaces and avoid unintentionally imposing constraints on implementers. Built into the model are two guiding principles also reflected in the standard. First, PMIx operates in the mode of a *messenger*, and not a *doer* — i.e., the role of PMIx is to provide communication between the various participants, relaying requests and returning responses. The intent of the standard is not to suggest that PMIx itself actually perform any of the defined operations — this is left to the various SMS elements and/or the application. Any exceptions to that intent are left to the discretion of the particular implementation.

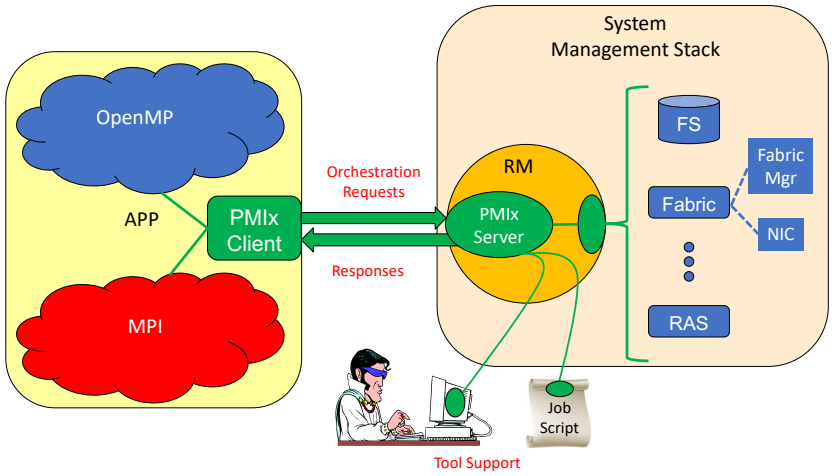

Figure 1.1.: PMIx-SMS Interactions

Thus, as the diagram in Fig. 1.1 shows, the application is built against a PMIx client library that contains the client-side APIs, attribute definitions, and communication support for interacting with the local PMIx server. Intra-process cross-library interactions are supported at the client level to avoid unnecessary burdens on the server. Orchestration requests are sent to the local PMIx server, which subsequently passes them to the host SMS (here represented by an RM daemon) using the PMIx server callback functions the host SMS registered during PMIx_server_init. The host SMS can indicate its lack of support for any operation by simply providing a *NULL* for the associated callback function, or can create a function entry that returns *not supported* when called.

The conceptual model places the burden of fulfilling the request on the host SMS. This includes performing any inter-node communications, or interacting with other SMS elements. Thus, a client request for a network traffic report does not go directly from the client to the Fabric Manager (FM), but instead is relayed to the PMIx server, and then passed to the host SMS for execution. This architecture reflects the second principle underlying the standard — namely, that connectivity is to be minimized by channeling all application interactions with the SMS through the local PMIx server.

Recognizing the burden this places on SMS vendors, the PMIx community has included interfaces

by which the host can request support from local SMS elements. Once the SMS has transferred the request to an appropriate location, a PMIx server interface can be used to pass the request between SMS subsystems. For example, a request for network traffic statistics can utilize the PMIx networking abstractions to retrieve the information from the FM. This reduces the portability and interoperability issues between the individual subsystems by transferring the burden of defining the interoperable interfaces from the SMS subsystems to the PMIx community, which continues to work with those providers to develop the necessary support.

Tools, whether standalone or embedded in job scripts, are an exception to the communication rule and can connect to any PMIx server providing they are given adequate rendezvous information. The PMIx conceptual model views the collection of PMIx servers as a cloud-like conglomerate — i.e., orchestration and information requests can be given to any server regardless of location. However, tools frequently execute on locations that may not house an operating PMIx server — e.g., a users notebook computer. Thus, tools need the ability to remotely connect to the PMIx server "cloud".

The scope of the PMIx standard therefore spans the range of these interactions, between client-and-SMS and between SMS subsystems. Note again that this does not impose a requirement on any given PMIx implementation to cover the entire range — implementers are free to return *not supported* from any PMIx function.

1.3.1 The PMIx Reference Implementation (PRI)

The PMIx community has committed to providing a complete, reference implementation of each version of the standard. Note that the definition of the PMIx Standard is not contingent upon use of the PMIx Reference Implementation (PRI) — any implementation that supports the defined APIs is a PMIx Standard compliant implementation. The PRI is provided solely for the following purposes:

- Validation of the standard.
  No proposed change and/or extension to the PMIx standard is accepted without an accompanying prototype implementation in the PRI. This ensures that the proposal has undergone at least some minimal level of scrutiny and testing before being considered.

- Ease of adoption.
  The PRI is designed to be particularly easy for resource managers (and the SMS in general) to adopt, thus facilitating a rapid uptake into that community for application portability. Both client and server PMIx libraries are included, along with examples of client usage and server-side integration. A list of supported environments and versions is maintained on the PMIx web site https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/

The PRI does provide some internal implementations that lie outside the scope of the PMIx standard. This includes several convenience macros as well as support for consolidating collectives for optimization purposes (e.g., the PMIx server aggregates all local **`PMIx_Fence`** calls before passing them to the SMS for global execution). In a few additional cases, the PMIx community (in partnership with the SMS subsystem providers) have determined that a base level of support for a given operation can best be portably provided by including it in the PRI.

Instructions for downloading, and installing the PRI are available on the community's web site https://pmix.org/code/getting-the-reference-implementation/.The PRI targets support for the Linux operating system. A reasonable effort is made to support all major, modern Linux distributions; however, validation is limited to the most recent 2-3 releases of RedHat Enterprise Linux (RHEL), Fedora, CentOS, and SUSE Linux Enterprise Server (SLES). In addition, development support is maintained for Mac OSX. Production support for vendor-specific operating systems is included as provided by the vendor.

1.3.2 The PMIx Reference RunTime Environment (PRRTE)

The PMIx community has also released PRRTE — i.e., a runtime environment containing the reference implementation and capable of operating within a host SMS. PRRTE provides an easy way of exploring PMIx capabilities and testing PMIx-based applications outside of a PMIx-enabled environment by providing a "shim" between the application and the host environment that includes full support for the PRI. The intent of PRRTE is not to replace any existing production environment, but rather to enable developers to work on systems that do not yet feature a PMIx-enabled host SMS or one that lacks a PMIx feature of interest. Instructions for downloading, installing, and using PRRTE are available on the community's web site https://pmix.org/code/getting-the-pmix-reference-server/

1.4 Organization of this document

The remainder of this document is structured as follows:

- Introduction and Overview in Chapter 1 on page 1
- Terms and Conventions in Chapter 2 on page 14
- Data Structures and Types in Chapter 3 on page 20
- PMIx Initialization and Finalization in Chapter 4 on page 112
- Key/Value Management in Chapter 5 on page 125
- Process Management in Chapter 6 on page 154
- Job Management in Chapter 7 on page 180
- Event Notification in Chapter 8 on page 216
- Data Packing and Unpacking in Chapter 9 on page 226
- Security in Chapter 10 on page 236
- PMIx Server Specific Interfaces in Chapter 11 on page 245
- Scheduler-Specific Interface in Chapter 12 on page 330
- Process Sets and Groups in Chapter 13 on page 335

- Network Coordinates in Chapter 14 on page 363
- Python Bindings in Appendix A on page 368

1.5 Version 1.0: June 12, 2015

The PMIx version 1.0 *ad hoc* standard was defined in the PMIx Reference Implementation (PRI) header files as part of the PRI v1.0.0 release prior to the creation of the formal PMIx 2.0 standard. Below are a summary listing of the interfaces defined in the 1.0 headers.

- Client APIs
  - PMIx_Init, **`PMIx_Initialized`** , **`PMIx_Abort`** , **`PMIx_Finalize`**
  - **`PMIx_Put`** , **`PMIx_Commit`** ,
  - **`PMIx_Fence`** , **`PMIx_Fence_nb`**
  - **`PMIx_Get`** , **`PMIx_Get_nb`**
  - **`PMIx_Publish`** , **`PMIx_Publish_nb`**
  - **`PMIx_Lookup`** , **`PMIx_Lookup`**
  - **`PMIx_Unpublish`** , **`PMIx_Unpublish_nb`**
  - **`PMIx_Spawn`** , **`PMIx_Spawn_nb`**
  - **`PMIx_Connect`** , **`PMIx_Connect_nb`**
  - **`PMIx_Disconnect`** , **`PMIx_Disconnect_nb`**
  - **`PMIx_Resolve_nodes`** , **`PMIx_Resolve_peers`**
- Server APIs
  - **`PMIx_server_init`** , **`PMIx_server_finalize`**
  - **`PMIx_generate_regex`** , **`PMIx_generate_ppn`**
  - **`PMIx_server_register_nspace`** , **`PMIx_server_deregister_nspace`**
  - **`PMIx_server_register_client`** , **`PMIx_server_deregister_client`**
  - **`PMIx_server_setup_fork`** , **`PMIx_server_dmodex_request`**
- Common APIs
  - **`PMIx_Get_version`** , **`PMIx_Store_internal`** , **`PMIx_Error_string`**
  - PMIx_Register_errhandler, PMIx_Deregister_errhandler, PMIx_Notify_error

The **`PMIx_Init`** API was subsequently modified in the PRI release v1.1.0.

1.6 Version 2.0: Sept. 2018

The following APIs were introduced in v2.0 of the PMIx Standard:

- Client APIs
  - **PMIx_Query_info_nb** , **PMIx_Log_nb**
  - **PMIx_Allocation_request_nb** , **PMIx_Job_control_nb** , **PMIx_Process_monitor_nb** , **PMIx_Heartbeat**
- Server APIs
  - **PMIx_server_setup_application** , **PMIx_server_setup_local_support**
- Tool APIs
  - **PMIx_tool_init** , **PMIx_tool_finalize**
- Common APIs
  - **PMIx_Register_event_handler** , **PMIx_Deregister_event_handler**
  - **PMIx_Notify_event**
  - **PMIx_Proc_state_string** , **PMIx_Scope_string**
  - **PMIx_Persistence_string** , **PMIx_Data_range_string**
  - **PMIx_Info_directives_string** , **PMIx_Data_type_string**
  - **PMIx_Alloc_directive_string**
  - **PMIx_Data_pack** , **PMIx_Data_unpack** , **PMIx_Data_copy**
  - **PMIx_Data_print** , **PMIx_Data_copy_payload**

The **PMIx_Init** API was modified in v2.0 of the standard from its *ad hoc* v1.0 signature to include passing of a **pmix_info_t** array for flexibility and "future-proofing" of the API. In addition, the PMIx_Notify_error, PMIx_Register_errhandler, and PMIx_Deregister_errhandler APIs were replaced.

1.7 Version 2.1: Dec. 2018

The v2.1 update includes clarifications and corrections from the v2.0 document, plus addition of examples:

- Clarify description of **PMIx_Connect** and **PMIx_Disconnect** APIs.
- Explain that values for the **PMIX_COLLECTIVE_ALGO** are environment-dependent
- Identify the namespace/rank values required for retrieving attribute-associated information using the **PMIx_Get** API

- Provide definitions for **session** , **job** , **application** , and other terms used throughout the document
- Clarify definitions of **PMIX_UNIV_SIZE** versus **PMIX_JOB_SIZE**
- Clarify server module function return values
- Provide examples of the use of **PMIx_Get** for retrieval of information
- Clarify the use of **PMIx_Get** versus **PMIx_Query_info_nb**
- Clarify return values for non-blocking APIs and emphasize that callback functions must not be invoked prior to return from the API
- Provide detailed example for construction of the **PMIx_server_register_nspace** input information array
- Define information levels (e.g., **session** vs **job**) and associated attributes for both storing and retrieving values
- Clarify roles of PMIx server library and host environment for collective operations
- Clarify definition of **PMIX_UNIV_SIZE**

1.8 Version 2.2: Jan 2019

The v2.2 update includes the following clarifications and corrections from the v2.1 document:

- Direct modex upcall function (**pmix_server_dmodex_req_fn_t**) cannot complete atomically as the API cannot return the requested information except via the provided callback function
- Add missing **pmix_data_array_t** definition and support macros
- Add a rule divider between implementer and host environment required attributes for clarity
- Add **PMIX_QUERY_QUALIFIERS_CREATE** macro to simplify creation of **pmix_query_t** qualifiers
- Add **PMIX_APP_INFO_CREATE** macro to simplify creation of **pmix_app_t** directives
- Add flag and **PMIX_INFO_IS_END** macro for marking and detecting the end of a **pmix_info_t** array
- Clarify the allowed hierarchical nesting of the **PMIX_SESSION_INFO_ARRAY** , **PMIX_JOB_INFO_ARRAY** , and associated attributes

1.9 Version 3.0: Dec. 2018

The following APIs were introduced in v3.0 of the PMIx Standard:

- Client APIs
  - **PMIx_Log** , **PMIx_Job_control**
  - **PMIx_Allocation_request** , **PMIx_Process_monitor**
  - **PMIx_Get_credential** , **PMIx_Validate_credential**
- Server APIs
  - **PMIx_server_IOF_deliver**
  - **PMIx_server_collect_inventory** , **PMIx_server_deliver_inventory**
- Tool APIs
  - **PMIx_IOF_pull** , **PMIx_IOF_push** , **PMIx_IOF_deregister**
  - **PMIx_tool_connect_to_server**
- Common APIs
  - **PMIx_IOF_channel_string**

The document added a chapter on security credentials, a new section for Input/Output (IO) forwarding to the Process Management chapter, and a few blocking forms of previously-existing non-blocking APIs. Attributes supporting the new APIs were introduced, as well as additional attributes for a few existing functions.

1.10 Version 3.1: Jan. 2019

The v3.1 update includes clarifications and corrections from the v3.0 document:

- Direct modex upcall function (**pmix_server_dmodex_req_fn_t**) cannot complete atomically as the API cannot return the requested information except via the provided callback function
- Fix typo in name of **PMIX_FWD_STDDIAG** attribute
- Correctly identify the information retrieval and storage attributes as “new” to v3 of the standard
- Add missing **pmix_data_array_t** definition and support macros
- Add a rule divider between implementer and host environment required attributes for clarity
- Add **PMIX_QUERY_QUALIFIERS_CREATE** macro to simplify creation of **pmix_query_t** qualifiers
- Add **PMIX_APP_INFO_CREATE** macro to simplify creation of **pmix_app_t** directives

- Add new attributes to specify the level of information being requested where ambiguity may exist (see 3.4.11)
- Add new attributes to assemble information by its level for storage where ambiguity may exist (see 3.4.12)
- Add flag and **PMIX_INFO_IS_END** macro for marking and detecting the end of a **pmix_info_t** array
- 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.
- Clarify difference between **PMIX_JOB_SIZE** and **PMIX_MAX_PROCS**
- 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.
- 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

1.11 Version 3.2: Oct. 2019

The v3.2 update includes clarifications and corrections from the v3.1 document:

- Correct an error in the **PMIx_Allocation_request** function signature, and clarify the allocation ID attributes
- Rename the **PMIX_ALLOC_ID** attribute to **PMIX_ALLOC_REQ_ID** to clarify that this is a 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**

1.12 Version 4.0: June 2019

The following changes were introduced in v4.0 of the PMIx Standard:

- Clarified that the **PMIx_Fence_nb** operation can immediately return **PMIX_OPERATION_SUCCEEDED** in lieu of passing the request to a PMIx server if only the calling process is involved in the operation

- Added the **PMIx_Register_attributes** API by which a host environment can register the attributes it supports for each server-to-host operation
- Added the ability to query supported attributes from the PMIx tool, client and server libraries, as well as the host environment via the new **pmix_regattr_t** structure. Both human-readable and machine-parsable output is supported. New attributes to support this operation include:
  - **PMIX_CLIENT_ATTRIBUTES** , **PMIX_SERVER_ATTRIBUTES** , **PMIX_TOOL_ATTRIBUTES** , and **PMIX_HOST_ATTRIBUTES** to identify which library supports the attribute; and
  - **PMIX_MAX_VALUE** , **PMIX_MIN_VALUE** , and **PMIX_ENUM_VALUE** to provide machine-parsable description of accepted values
- Add **PMIX_APP_WILDCARD** to reference all applications within a given job
- Fix signature of blocking APIs **PMIx_Allocation_request** , **PMIx_Job_control** , **PMIx_Process_monitor** , **PMIx_Get_credential** , and **PMIx_Validate_credential** to allow return of results
- Update description to provide an option for blocking behavior of the **PMIx_Register_event_handler** , **PMIx_Deregister_event_handler** , **PMIx_Notify_event** , **PMIx_IOF_pull** , **PMIx_IOF_deregister** , and **PMIx_IOF_push** APIs. The need for blocking forms of these functions was not initially anticipated but has emerged over time. For these functions, the return value is sufficient to provide the caller with information otherwise returned via callback. Thus, use of a **NULL** value as the callback function parameter was deemed a minimal disruption method for providing the desired capability

CHAPTER 2

PMIx Terms and Conventions

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 community 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.

The PMIx community has further adopted a policy that modification of existing released APIs will only be permitted under extreme circumstances. In its effort to avoid introduction of any such backward incompatibility, the community has avoided the definitions of large numbers of APIs that each focus on a narrow scope of functionality, and instead relied on the definition of fewer generic APIs that include arrays of directives for "tuning" the function's behavior. Thus, modifications to the PMIx standard increasingly 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.

One area where this can become more complicated relates to the attributes that provide directives to the client process and/or control the behavior of a PMIx standard API. For example, the **`PMIX_TIMEOUT`** attribute can be used to specify the time (in seconds) before the requested operation should time out. The intent of this attribute is to allow the client to avoid hanging in a request that takes longer than the client wishes to wait, or may never return (e.g., a **`PMIx_Fence`** that a blocked participant never enters).

If an application truly relies on the **`PMIX_TIMEOUT`** attribute in a call to **`PMIx_Fence`** , it should set the *required* flag in the **`pmix_info_t`** for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, silently ignoring it if support is not available.

Advice to users

It is critical that users and application developers consider whether or not a given attribute is required (marking it accordingly) and always check the return status on all PMIx function calls to ensure support was present and that the request was accepted. Note that for non-blocking APIs, a return of **`PMIX_SUCCESS`** only indicates that the request had no obvious errors and is being processed. The eventual callback will return the status of the requested operation itself.

While a PMIx library implementer, or an SMS component server, may choose to support a particular PMIx API, they are not required to support every attribute that might apply to it. This would pose a significant barrier to entry for an implementer as there can be a broad range of applicable attributes to a given API, at least some of which may rarely be used in a specific market area. The PMIx community is attempting to help differentiate the attributes by indicating in the standard those that are generally used (and therefore, of higher importance to support) versus those that a "complete implementation" would support.

In addition, the document refers to the following entities and process stages when describing use-cases or operations involving PMIx:

- *session* refers to an allocated set of resources assigned to a particular user by the system WLM. Historically, HPC sessions have consisted of a static allocation of resources - i.e., a block of resources are 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 the current block of assigned resources and is a potentially dynamic entity.
- *slot* refers to an allocated entry for a process. WLMs frequently allocate entire nodes to a *session*, but can also be configured to define the maximum number of processes that can simultaneously be executed on each node. This often corresponds to the number of hardware Processing Units (PUs) (typically cores, but can also be defined as hardware threads) on the node. However, the correlation between hardware PUs and slot allocations strictly depends upon system configuration.
- *job* refers to a set of one or more *applications* executed as a single invocation by the user within a session. For example, "*mpiexec -n 1 app1 : -n 2 app2*" is considered a single Multiple Program Multiple Data (MPMD) job containing two applications.
- *namespace* refers to a character string value assigned by the RM 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.
- *application* refers to a single executable (binary, script, etc.) member of a *job*. Applications consist of one or more *processes*, either operating independently or in parallel at any given time during their execution.
- *rank* refers to the numerical location (starting from zero) of a process within the defined scope. Thus, global rank is the rank of a process within its *job*, while *application rank* is the rank of that process within its *application*.
- *workflow* refers to an orchestrated execution plan frequently spanning 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's RM, a programming library's RunTime Environment (RTE), or some other agent.
- *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.
- *network plane* refers to a collection of Network Interface Cards (NICs) and switches in a common logical or physical configuration. Network planes are often implemented in HPC clusters as separate overlay or physical networks controlled by a dedicated fabric manager.

This document borrows freely from other standards (most notably from the Message Passing Interface (MPI) and OpenMP standards) in its use of notation and conventions in an attempt to reduce confusion. 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

C specific text...

```
int foo = 42;
```

C

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

Note: General text...

Rationale

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

Advice to users

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

2.2 Semantics

The following terms will be taken to mean:

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

2.3 Naming Conventions

The PMIx standard has adopted the following conventions:

- PMIx constants and attributes are prefixed with **PMIX**.
- Structures and type definitions are prefixed with **pmix**.
- Underscores are used to separate words in a function or variable name.
- Lowercase letters are used in PMIx client APIs except for the PMIx prefix (noted below) and the first letter of the word following it. For example, **PMIx_Get_version** .
- PMIx server and tool APIs are all lower case letters following the prefix - e.g., **PMIx_server_register_nspace** .
- The **PMIx_** prefix is used to denote functions.
- The **pmix_** prefix is used to denote function pointer and type definitions.

Users should not use the **PMIX**, **PMIx**, or **pmix** prefixes in their applications or libraries so as to avoid symbol conflicts with current and later versions of the PMIx standard and implementations such as the PRI.

2.4 Procedure Conventions

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

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

Many PMIx interfaces, particularly nonblocking interfaces, use a **void*** cbdata object passed to the function that is then passed to the associated callback. In a client-side API, the cbdata is a client-provided context (opaque object) that the client can pass to the nonblocking call (e.g., **PMIx_Get_nb**). When the nonblocking call (e.g., **pmix_value_cbfunc_t**) completes, the cbdata is passed back to the client without modification by the PMIx library, thus allowing the client to associate a context with that callback. This is useful if there are many outstanding nonblocking calls.

A similar model is used for the server module functions (see 11.2.1). In this case, the PMIx library is making an upcall into its host via the PMIx server module function and passing a specific cbfunc

and cbdata. The PMIx library expects the host to call the cbfunc with the necessary arguments and pass back the original cbdata upon completing the operation. This gives the server-side PMIx library the ability to associate a context with the call back (since multiple operations may be outstanding). The host has no visibility into the contents of the cbdata object, nor is permitted to alter it in any way.

2.5 Standard vs Reference Implementation

The *PMIx Standard* is implementation independent. The *PMIx Reference Implementation* (PRI) is one implementation of the Standard and the PMIx community strives to ensure that it fully implements the Standard. Given its role as the community's testbed and its widespread use, this document cites the attributes supported by the PRI for each API where relevant by marking them in red. This is not meant to imply nor confer any special role to the PRI with respect to the Standard itself, but instead to provide a convenience to users of the Standard and PRI.

Similarly, the *PMIx Reference RunTime Environment* (PRRTE) is provided by the community to enable users operating in non-PMIx environments to develop and execute PMIx-enabled applications and tools. Attributes supported by the PRRTE are marked in green.

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:

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

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

```
define PMIX_INFO_LOAD(m, k, v, t)                          \
  do {                                                     \
    if (NULL != (k)) {                                     \
      pmix_strncpy((m)->key, (k), PMIX_MAX_KEYLEN);        \
    }                                                      \
    (m)->flags = 0;                                        \
    pmix_value_load(&((m)->value), (v), (t));              \
  } 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:

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

3.1 Constants

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

PMIX_MAX_NSLEN Maximum namespace string length as an integer.

Advice to PMIx library implementers

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

PMIX_MAX_KEYLEN Maximum key string length as an integer.

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

Advice to PMIx library implementers

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

3.1.1 PMIx Error 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.

3.1.1.1 General Error Constants

These are general constants originally defined in versions 1 and 2 of the PMIx Standard.

PMIX_SUCCESS Success
PMIX_ERROR General Error
PMIX_ERR_SILENT Silent error
PMIX_ERR_DEBUGGER_RELEASE Error in debugger release
PMIX_ERR_PROC_RESTART Fault tolerance: Error in process restart
PMIX_ERR_PROC_CHECKPOINT Fault tolerance: Error in process checkpoint
PMIX_ERR_PROC_MIGRATE Fault tolerance: Error in process migration
PMIX_ERR_PROC_ABORTED Process was aborted
PMIX_ERR_PROC_REQUESTED_ABORT Process is already requested to abort
PMIX_ERR_PROC_ABORTING Process is being aborted
PMIX_ERR_SERVER_FAILED_REQUEST Failed to connect to the server
PMIX_EXISTS Requested operation would overwrite an existing value
PMIX_ERR_INVALID_CRED Invalid security credentials
PMIX_ERR_HANDSHAKE_FAILED Connection handshake failed
PMIX_ERR_READY_FOR_HANDSHAKE Ready for handshake
PMIX_ERR_WOULD_BLOCK Operation would block
PMIX_ERR_UNKNOWN_DATA_TYPE Unknown data type
PMIX_ERR_PROC_ENTRY_NOT_FOUND Process not found
PMIX_ERR_TYPE_MISMATCH Invalid type
PMIX_ERR_UNPACK_INADEQUATE_SPACE Inadequate space to unpack data

PMIX_ERR_UNPACK_FAILURE Unpack failed
PMIX_ERR_PACK_FAILURE Pack failed
PMIX_ERR_PACK_MISMATCH Pack mismatch
PMIX_ERR_NO_PERMISSIONS No permissions
PMIX_ERR_TIMEOUT Timeout expired
PMIX_ERR_UNREACH Unreachable
PMIX_ERR_IN_ERRNO Error defined in **errno**
PMIX_ERR_BAD_PARAM Bad parameter
PMIX_ERR_RESOURCE_BUSY Resource busy
PMIX_ERR_OUT_OF_RESOURCE Resource exhausted
PMIX_ERR_DATA_VALUE_NOT_FOUND Data value not found
PMIX_ERR_INIT Error during initialization
PMIX_ERR_NOMEM Out of memory
PMIX_ERR_INVALID_ARG Invalid argument
PMIX_ERR_INVALID_KEY Invalid key
PMIX_ERR_INVALID_KEY_LENGTH Invalid key length
PMIX_ERR_INVALID_VAL Invalid value
PMIX_ERR_INVALID_VAL_LENGTH Invalid value length
PMIX_ERR_INVALID_LENGTH Invalid argument length
PMIX_ERR_INVALID_NUM_ARGS Invalid number of arguments
PMIX_ERR_INVALID_ARGS Invalid arguments
PMIX_ERR_INVALID_NUM_PARSED Invalid number parsed
PMIX_ERR_INVALID_KEYVALP Invalid key/value pair
PMIX_ERR_INVALID_SIZE Invalid size
PMIX_ERR_INVALID_NAMESPACE Invalid namespace
PMIX_ERR_SERVER_NOT_AVAIL Server is not available
PMIX_ERR_NOT_FOUND Not found
PMIX_ERR_NOT_SUPPORTED Not supported
PMIX_ERR_NOT_IMPLEMENTED Not implemented
PMIX_ERR_COMM_FAILURE Communication failure
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER Unpacking past the end of the buffer provided
PMIX_ERR_LOST_CONNECTION_TO_SERVER Lost connection to server
PMIX_ERR_LOST_PEER_CONNECTION Lost connection to peer
PMIX_ERR_LOST_CONNECTION_TO_CLIENT Lost connection to client
PMIX_QUERY_PARTIAL_SUCCESS Query partial success (used by query system)
PMIX_NOTIFY_ALLOC_COMPLETE Notify that allocation is complete
PMIX_JCTRL_CHECKPOINT Job control: Monitored by PMIx client to trigger checkpoint operation
PMIX_JCTRL_CHECKPOINT_COMPLETE Job control: Sent by PMIx client and monitored by PMIx server to notify that requested checkpoint operation has completed.
PMIX_JCTRL_PREEMPT_ALERT Job control: Monitored by PMIx client to detect an RM intending to preempt the job.

PMIX_MONITOR_HEARTBEAT_ALERT Job monitoring: Heartbeat alert

PMIX_MONITOR_FILE_ALERT Job monitoring: File alert

PMIX_PROC_TERMINATED Process terminated - can be either normal or abnormal termination

PMIX_ERR_INVALID_TERMINATION Process terminated without calling **PMIx_Finalize** , or was a member of an assemblage formed via **PMIx_Connect** and terminated or called **PMIx_Finalize** without first calling **PMIx_Disconnect** (or its non-blocking form) from that assemblage.

3.1.1.2 Operational Error Constants

PMIX_ERR_EVENT_REGISTRATION Error in event registration

PMIX_ERR_JOB_TERMINATED Error job terminated

PMIX_ERR_UPDATE_ENDPOINTS Error updating endpoints

PMIX_MODEL_DECLARED Model declared

PMIX_GDS_ACTION_COMPLETE The global data storage (GDS) action has completed

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).

PMIX_PROC_HAS_CONNECTED A tool or client has connected to the PMIx server

PMIX_CONNECT_REQUESTED Connection has been requested by a PMIx-based tool

PMIX_MODEL_RESOURCES Resource usage by a programming model has changed

PMIX_OPENMP_PARALLEL_ENTERED An OpenMP parallel code region has been entered

PMIX_OPENMP_PARALLEL_EXITED An OpenMP parallel code region has completed

PMIX_LAUNCH_DIRECTIVE Launcher directives have been received from a PMIx-enabled tool

PMIX_LAUNCHER_READY Application launcher (e.g., mpiexec) is ready to receive directives from a PMIx-enabled tool

PMIX_LAUNCH_COMPLETE A job has been launched - the nspace of the launched job will be included in the notification

PMIX_OPERATION_IN_PROGRESS A requested operation is already in progress

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)

PMIX_ERR_DUPLICATE_KEY The provided key has already been published on a different data range

PMIX_ERR_INVALID_OPERATION The requested operation is not valid - this can possibly indicate the inclusion of conflicting directives or a request to perform an operation that conflicts with an ongoing one.

PMIX_GROUP_INVITED The process has been invited to join a PMIx Group - the identifier of the group and the ID's of other invited (or already joined) members will be included in the notification

PMIX_GROUP_LEFT A process has asynchronously left a PMIx Group - the process identifier of the departing process will in included in the notification

PMIX_GROUP_MEMBER_FAILED A member of a PMIx Group has abnormally terminated (i.e., without formally leaving the group prior to termination) - the process identifier of the failed process will in included in the notification

PMIX_GROUP_INVITE_ACCEPTED A process has accepted an invitation to join a PMIx Group - the identifier of the group being joined will be included in the notification

PMIX_GROUP_INVITE_DECLINED A process has declined an invitation to join a PMIx Group - the identifier of the declined group will be included in the notification

PMIX_GROUP_INVITE_FAILED An invited process failed or terminated prior to responding to the invitation - the identifier of the failed process will be included in the notification.

PMIX_GROUP_MEMBERSHIP_UPDATE The membership of a PMIx group has changed - the identifiers of the revised membership will be included in the notification.

PMIX_GROUP_CONSTRUCT_ABORT Any participant in a PMIx group construct operation that returns **PMIX_GROUP_CONSTRUCT_ABORT** from the *leader failed* event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning this error code from the handler assigned to the **PMIX_GROUP_INVITE_ACCEPTED** or **PMIX_GROUP_INVITE_DECLINED** codes, or by generating an event for the abort code. Abort events will be sent to all invited or existing members of the group.

PMIX_GROUP_CONSTRUCT_COMPLETE The group construct operation has completed - the final membership will be included in the notification.

PMIX_GROUP_LEADER_FAILED The current *leader* of a group including this process has abnormally terminated - the group identifier will be included in the notification.

PMIX_GROUP_LEADER_SELECTED A new *leader* of a group including this process has been selected - the identifier of the new leader will be included in the notification

PMIX_GROUP_CONTEXT_ID_ASSIGNED A new Process Group Context IDentifier (PGCID) has been assigned by the host environment to a group that includes this process - the group identifier will be included in the notification.

PMIX_ERR_REPEAT_ATTR_REGISTRATION The attributes for an identical function have already been registered at the specified level (host, server, or client)

PMIX_ERR_IOF_FAILURE An IO forwarding operation failed - the affected channel will be included in the notification

PMIX_ERR_IOF_COMPLETE IO forwarding of the standard input for this process has completed - i.e., the stdin file descriptor has closed

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

3.1.1.3 System error constants

PMIX_ERR_SYS_BASE Mark the beginning of a dedicated range of constants for system event reporting.

PMIX_ERR_NODE_DOWN A node has gone down - the identifier of the affected node will be included in the notification

PMIX_ERR_NODE_OFFLINE A node has been marked as *offline* - the identifier of the affected node will be included in the notification

PMIX_ERR_SYS_OTHER Mark the end of a dedicated range of constants for system event reporting.

3.1.1.4 Event handler error constants

PMIX_EVENT_NO_ACTION_TAKEN Event handler: No action taken

PMIX_EVENT_PARTIAL_ACTION_TAKEN Event handler: Partial action taken

PMIX_EVENT_ACTION_DEFERRED Event handler: Action deferred

PMIX_EVENT_ACTION_COMPLETE Event handler: Action complete

3.1.1.5 User-Defined Error Constants

PMIx establishes an error code boundary for constants defined in the PMIx standard. Negative values larger 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 constants. Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions should always be based on the **PMIX_EXTERNAL_ERR_BASE** constant and not a specific value as the value of the constant may change.

3.1.2 Macros for use with PMIx constants

3.1.2.1 Detect system event constant

Test a given error constant to see if it falls within the dedicated range of constants for system event reporting.

PMIx v2.2

C

```
PMIX_SYSTEM_EVENT(a)
```

C

IN **a**
Error constant to be checked (**pmix_status_t**)

Returns **true** if the provided values falls within the dedicated range of constants for system event reporting

3.2 Data Types

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

3.2.1 Key Structure

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

PMIx v2.0

C

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

C

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

Advice to users

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

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

3.2.1.1 Key support macro

Compare the key in a **pmix_info_t** to a given value

PMIx v3.0

C

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

C

IN **a**
Pointer to the structure whose key is to be checked (pointer to **pmix_info_t**)

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

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

3.2.2 Namespace Structure

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

PMIx v2.0

C

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

C

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

Advice to users

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

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

3.2.2.1 Namespace support macro

Compare the string in a **pmix_nspace_t** to a given value

PMIx v3.0

C

```
PMIX_CHECK_NSPACE(a, b)
```

C

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

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

Returns **true** if the namespace matches the given value

3.2.3 Rank Structure

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

PMIx v1.0

C

```
typedef uint32_t pmix_rank_t;
```

C

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

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

PMIX_RANK_WILDCARD A value to indicate that the user wants the data for the given key from every rank that posted that key.

PMIX_RANK_LOCAL_NODE Special rank value used to define groups of ranks. This constant defines the group of all ranks on a local node.

PMIX_RANK_LOCAL_PEERS Special rank value used to define groups of rankss. This constant defines the group of all ranks on a local node within the same namespace as the current process.

PMIX_RANK_INVALID An invalid rank value.

PMIX_RANK_VALID Define an upper boundary for valid rank values.

3.2.4 Process Structure

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

PMIx v1.0

C

```
typedef struct pmix_proc {
    pmix_nspace_t nspace;
    pmix_rank_t rank;
} pmix_proc_t;
```

C

3.2.5 Process structure support macros

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

3.2.5.1 Initialize the pmix_proc_t structure

PMIX_PROC_CONSTRUCT

Initialize the pmix_proc_t fields

PMIx v1.0

C

```
PMIX_PROC_CONSTRUCT(m)
```

C

IN m
Pointer to the structure to be initialized (pointer to pmix_proc_t)

3.2.5.2 Destruct the pmix_proc_t structure

There is nothing to release here as the fields in pmix_proc_t are all declared *static*. However, the macro is provided for symmetry in the code and for future-proofing should some allocated field be included some day.

3.2.5.3 Create a pmix_proc_t array

Allocate and initialize an array of pmix_proc_t structures

PMIx v1.0

C

```
PMIX_PROC_CREATE(m, n)
```

C

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)

3.2.5.4 Free a pmix_proc_t array

Release an array of pmix_proc_t structures

PMIx v1.0

C

```
PMIX_PROC_FREE(m, n)
```

C

IN m
Pointer to the array of pmix_proc_t structures (handle)
IN n
Number of structures in the array (size_t)

3.2.5.5 Load a `pmix_proc_t` structure

Load values into a `pmix_proc_t`

PMIx v2.0

C

```
PMIX_PROC_LOAD(m, n, r)
```

C

IN `m`
Pointer to the structure to be loaded (pointer to `pmix_proc_t`)

IN `n`
Namespace to be loaded (`pmix_nspace_t`)

IN `r`
Rank to be assigned (`pmix_rank_t`)

3.2.5.6 Compare identifiers

Compare two `pmix_proc_t` identifiers

PMIx v3.0

C

```
PMIX_CHECK_PROCID(a, b)
```

C

IN `a`
Pointer to a structure whose ID is to be compared (pointer to `pmix_proc_t`)

IN `b`
Pointer to a structure whose ID is to be compared (pointer to `pmix_proc_t`)

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

- the ranks are the same value
- one of the ranks is `PMIX_RANK_WILDCARD`

3.2.6 Process State Structure

PMIx v2.0

The `pmix_proc_state_t` structure is a **`uint8_t`** type for process state values. The following constants can be used to set a variable of the type `pmix_proc_state_t` . All values were originally defined in version 2 of the standard unless otherwise marked.

Advice to users

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

PMIX_PROC_STATE_UNDEF Undefined process state
PMIX_PROC_STATE_PREPPED Process is ready to be launched
PMIX_PROC_STATE_LAUNCH_UNDERWAY Process launch is underway
PMIX_PROC_STATE_RESTART Process is ready for restart
PMIX_PROC_STATE_TERMINATE Process is marked for termination
PMIX_PROC_STATE_RUNNING Process has been locally **fork**'ed by the RM
PMIX_PROC_STATE_CONNECTED Process has connected to PMIx server
PMIX_PROC_STATE_UNTERMINATED Define a "boundary" between the terminated states and **PMIX_PROC_STATE_CONNECTED** so users can easily and quickly determine if a process is still running or not. Any value less than this constant means that the process has not terminated.
PMIX_PROC_STATE_TERMINATED Process has terminated and is no longer running
PMIX_PROC_STATE_ERROR Define a boundary so users can easily and quickly determine if a process abnormally terminated. Any value above this constant means that the process has terminated abnormally.
PMIX_PROC_STATE_KILLED_BY_CMD Process was killed by a command
PMIX_PROC_STATE_ABORTED Process was aborted by a call to **PMIx_Abort**
PMIX_PROC_STATE_FAILED_TO_START Process failed to start
PMIX_PROC_STATE_ABORTED_BY_SIG Process aborted by a signal
PMIX_PROC_STATE_TERM_WO_SYNC Process exited without calling **PMIx_Finalize**
PMIX_PROC_STATE_COMM_FAILED Process communication has failed
PMIX_PROC_STATE_CALLED_ABORT Process called **PMIx_Abort**
PMIX_PROC_STATE_MIGRATING Process failed and is waiting for resources before restarting
PMIX_PROC_STATE_CANNOT_RESTART Process failed and cannot be restarted
PMIX_PROC_STATE_TERM_NON_ZERO Process exited with a non-zero status
PMIX_PROC_STATE_FAILED_TO_LAUNCH Unable to launch process

3.2.7 Process Information Structure

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

C

```
typedef struct pmix_proc_info {
    /** Process structure */
    pmix_proc_t proc;
    /** Hostname where process resides */
    char *hostname;
    /** Name of the executable */
    char *executable_name;
    /** Process ID on the host */
    pid_t pid;
    /** Exit code of the process. Default: 0 */
    int exit_code;
    /** Current state of the process */
    pmix_proc_state_t state;
} pmix_proc_info_t;
```

C

3.2.8 Process Information Structure support macros

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

3.2.8.1 Initialize the pmix_proc_info_t structure

Initialize the **pmix_proc_info_t** fields

PMIx v2.0

C

```
PMIX_PROC_INFO_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_proc_info_t**)

3.2.8.2 Destruct the pmix_proc_info_t structure

Destruct the **pmix_proc_info_t** fields

PMIx v2.0

C

```
PMIX_PROC_INFO_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_proc_info_t**)

3.2.8.3 Create a pmix_proc_info_t array

Allocate and initialize a **pmix_proc_info_t** array

PMIx v2.0

C

```
PMIX_PROC_INFO_CREATE(m, n)
```

C

INOUT **m**
: Address where the pointer to the array of **pmix_proc_info_t** structures shall be stored (handle)

IN **n**
: Number of structures to be allocated (**size_t**)

3.2.8.4 Free a pmix_proc_info_t array

Release an array of **pmix_proc_info_t** structures

PMIx v2.0

C

```
PMIX_PROC_INFO_FREE(m, n)
```

C

IN **m**
: Pointer to the array of **pmix_proc_info_t** structures (handle)

IN **n**
: Number of structures in the array (**size_t**)

3.2.9 Scope of Put Data

PMIx v1.0

The **pmix_scope_t** structure is a **uint8_t** type that defines the scope for 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 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 requestors — i.e., it is only available to processes on the local node.

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

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

PMIx v2.0

PMIX_INTERNAL The data is intended solely for this process and is not shared with other processes.

3.2.10 Range of Published Data

PMIx v1.0

The **pmix_data_range_t** structure is a **uint8_t** type that defines a range for data *published* via functions other than **PMIx_Put** - e.g., the **PMIx_Publish** API. The following constants can be used to set a variable of the type **pmix_data_range_t** . Several values were initially defined in version 1 of the standard but subsequently renamed and other values added in version 2. Thus, all values shown below are as they were defined in version 2 except where noted.

PMIX_RANGE_UNDEF Undefined range
PMIX_RANGE_RM Data is intended for the host resource manager.
PMIX_RANGE_LOCAL Data is only available to processes on the local node.
PMIX_RANGE_NAMESPACE Data is only available to processes in the same namespace.
PMIX_RANGE_SESSION Data is only available to all processes in the session.
PMIX_RANGE_GLOBAL Data is available to all processes.
PMIX_RANGE_CUSTOM Range is specified in the **pmix_info_t** associated with this call.
PMIX_RANGE_PROC_LOCAL Data is only available to this process.
PMIX_RANGE_INVALID Invalid value

Advice to users

The names of the **pmix_data_range_t** values changed between version 1 and version 2 of the standard, thereby breaking backward compatibility

3.2.11 Data Persistence Structure

PMIx v1.0

The **pmix_persistence_t** structure is a **uint8_t** type that defines the policy for data published by clients via the **PMIx_Publish** API. The following constants can be used to set a variable of the type **pmix_persistence_t** . All definitions were introduced in version 1 of the standard unless otherwise marked.

PMIX_PERSIST_INDEF Retain data until specifically deleted.
PMIX_PERSIST_FIRST_READ Retain data until the first access, then the data is deleted.
PMIX_PERSIST_PROC Retain data until the publishing process terminates.
PMIX_PERSIST_APP Retain data until the application terminates.
PMIX_PERSIST_SESSION Retain data until the session/allocation terminates.
PMIX_PERSIST_INVALID Invalid value

3.2.12 Data Array Structure

PMIx v2.0

C

```
typedef struct pmix_data_array
    pmix_data_type_t type;
    size_t size;
    void *array;
 pmix_data_array_t;
```

C

The **pmix_data_array_t** structure is used to pass arrays of related values. Any PMIx data type (including complex structures) can be included in the array.

3.2.13 Data array structure support macros

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

3.2.13.1 Initialize the pmix_data_array_t structure

Initialize the **pmix_data_array_t** fields, allocating memory for the array itself.

PMIx v2.2

C

```
PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_data_array_t**)

IN **n**
Number of elements in the array (**size_t**)

IN **t**
PMIx data type for the array elements (**pmix_data_type_t**)

3.2.13.2 Destruct the pmix_data_array_t structure

Destruct the **pmix_data_array_t** fields, releasing the array's memory.

PMIx v2.2

C

```
PMIX_DATA_ARRAY_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_data_array_t**)

3.2.13.3 Create and initialize a pmix_data_array_t object

Allocate and initialize a **pmix_data_array_t** structure and initialize it, allocating memory for the array itself as well.

PMIx v2.2

C

```
PMIX_DATA_ARRAY_CREATE(m, n, t)
```

C

INOUT `m`
: Address where the pointer to the **pmix_data_array_t** structure shall be stored (handle)

IN `n`
: Number of elements in the array (**size_t**)

IN `t`
: PMIx data type for the array elements (**pmix_data_type_t**)

3.2.13.4 Free a pmix_data_array_t object

Release a **pmix_data_array_t** structure, including releasing the array's memory.

PMIx v2.2

C

```
PMIX_DATA_ARRAY_FREE(m)
```

C

IN `m`
: Pointer to the **pmix_data_array_t** structure (handle)

3.2.14 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

```
typedef struct pmix_value {
    pmix_data_type_t type;
    union {
        bool flag;
        uint8_t byte;
        char *string;
        size_t size;
        pid_t pid;
        int integer;
        int8_t int8;
        int16_t int16;
        int32_t int32;
        int64_t int64;
        unsigned int uint;
        uint8_t uint8;
        uint16_t uint16;
        uint32_t uint32;
        uint64_t uint64;
        float fval;
        double dval;
        struct timeval tv;
        time_t time;                    // version 2.0
        pmix_status_t status;           // version 2.0
        pmix_rank_t rank;               // version 2.0
        pmix_proc_t *proc;              // version 2.0
        pmix_byte_object_t bo;
        pmix_persistence_t persist;     // version 2.0
        pmix_scope_t scope;             // version 2.0
        pmix_data_range_t range;        // version 2.0
        pmix_proc_state_t state;        // version 2.0
        pmix_proc_info_t *pinfo;        // version 2.0
        pmix_data_array_t *darray;      // version 2.0
        void *ptr;                      // version 2.0
        pmix_alloc_directive_t adir;    // version 2.0
    } data;
} pmix_value_t;
```

C

3.2.15 Value structure support macros

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

3.2.15.1 Initialize the pmix_value_t structure

Initialize the **pmix_value_t** fields

PMIx v1.0 C

```
PMIX_VALUE_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_value_t**)

3.2.15.2 Destruct the pmix_value_t structure

Destruct the **pmix_value_t** fields

PMIx v1.0 C

```
PMIX_VALUE_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_value_t**)

3.2.15.3 Create a pmix_value_t array

Allocate and initialize an array of **pmix_value_t** structures

PMIx v1.0 C

```
PMIX_VALUE_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_value_t** structures shall be stored (handle)
IN **n**
Number of structures to be allocated (**size_t**)

3.2.15.4 Free a pmix_value_t array

Release an array of **pmix_value_t** structures

PMIx v1.0 C

```
PMIX_VALUE_FREE(m, n)
```

C

IN **m**
Pointer to the array of **pmix_value_t** structures (handle)
IN **n**
Number of structures in the array (**size_t**)

3.2.15.5 Load a value structure

Summary

Load data into a **pmix_value_t** structure.

PMIx v2.0

C

```
PMIX_VALUE_LOAD(v, d, t);
```

C

IN **v**
The **pmix_value_t** into which the data is to be loaded (pointer to **pmix_value_t**)

IN **d**
Pointer to the data value to be loaded (handle)

IN **t**
Type of the provided data value (**pmix_data_type_t**)

Description

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

Advice to users

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

3.2.15.6 Unload a pmix_value_t structure

Summary

Unload data from a **pmix_value_t** structure.

PMIx v2.2

C

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

C

OUT **r**
Status code indicating result of the operation **pmix_status_t**

IN **v**
The **pmix_value_t** from which the data is to be unloaded (pointer to **pmix_value_t**)

INOUT **d**
Pointer to the location where the data value is to be returned (handle)

INOUT **t**
Pointer to return the data type of the unloaded value (handle)

Description

This macro simplifies the unloading of data from a **pmix_value_t** .

Advice to users

Memory will be allocated and the data will be in the **pmix_value_t** returned - the source **pmix_value_t** will not be altered.

3.2.15.7 Transfer data between pmix_value_t structures

Summary

Transfer the data value between two **pmix_value_t** structures.

PMIx v2.0

C

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

C

OUT **r**

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

IN **d**

Pointer to the **pmix_value_t** destination (handle)

IN **s**

Pointer to the **pmix_value_t** source (handle)

Description

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

Advice to users

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.

3.2.15.8 Retrieve a numerical value from a pmix_value_t

Retrieve a numerical value from a **pmix_value_t** structure

PMIx v3.0

C

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

C

OUT **s**
: Status code for the request (**pmix_status_t**)

IN **m**
: Pointer to the **pmix_value_t** structure (handle)

OUT **n**
: Variable to be set to the value (match expected type)

IN **t**
: Type of number expected in *m* (**pmix_data_type_t**)

Sets the provided variable equal to the numerical value contained in the given **pmix_value_t** , returning success if the data type of the value matches the expected type and **PMIX_ERR_BAD_PARAM** if it doesn't

3.2.16 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.

PMIx v1.0

C

```
typedef struct pmix_info_t {
    pmix_key_t key;
    pmix_info_directives_t flags;    // version 2.0
    pmix_value_t value;
} pmix_info_t;
```

C

3.2.17 Info structure support macros

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

3.2.17.1 Initialize the pmix_info_t structure

Initialize the **pmix_info_t** fields

PMIx v1.0

C

```
PMIX_INFO_CONSTRUCT(m)
```

C

IN **m**
: Pointer to the structure to be initialized (pointer to **pmix_info_t**)

3.2.17.2 Destruct the `pmix_info_t` structure

Destruct the **`pmix_info_t`** fields

PMIx v1.0 C

```
PMIX_INFO_DESTRUCT(m)
```

C

IN **`m`**
Pointer to the structure to be destructed (pointer to **`pmix_info_t`**)

3.2.17.3 Create a `pmix_info_t` array

Allocate and initialize an array of **`pmix_info_t`** structures

PMIx v1.0 C

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

C

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`**)

3.2.17.4 Free a `pmix_info_t` array

Release an array of **`pmix_info_t`** structures

PMIx v1.0 C

```
PMIX_INFO_FREE(m, n)
```

C

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

3.2.17.5 Load key and value data into a pmix_info_t

PMIx v1.0

C

```
PMIX_INFO_LOAD(v, k, d, t);
```

C

IN **v**
Pointer to the **pmix_info_t** into which the key and data are to be loaded (pointer to **pmix_info_t**)

IN **k**
String key to be loaded - must be less than or equal to **PMIX_MAX_KEYLEN** in length (handle)

IN **d**
Pointer to the data value to be loaded (handle)

IN **t**
Type of the provided data value (**pmix_data_type_t**)

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

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.

3.2.17.6 Copy data between pmix_info_t structures

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

PMIx v2.0

C

```
PMIX_INFO_XFER(d, s);
```

C

IN **d**
Pointer to the destination **pmix_info_t** (pointer to **pmix_info_t**)

IN **s**
Pointer to the source **pmix_info_t** (pointer to **pmix_info_t**)

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

Advice to users

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

3.2.17.7 Test a boolean pmix_info_t

A special macro for checking if a boolean **pmix_info_t** is **true**

PMIx v2.0

C

```
PMIX_INFO_TRUE(m)
```

C

IN **m**
Pointer to a **pmix_info_t** structure (handle)

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

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

3.2.18 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 if it is not explicitly marked as **PMIX_INFO_REQD** . This is because the values specified default to optional, meaning they can be ignored. This may lead to unexpected behavior if the user is relying on the behavior specified by the **pmix_info_t** value. If the user relies on the behavior defined by the **pmix_info_t** then they must 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_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.

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.

3.2.19 Info Directive support macros

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

3.2.19.1 Mark an info structure as required

Summary

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

PMIx v2.0

C

```
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.

3.2.19.2 Mark an info structure as optional

Summary

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

PMIx v2.0

C

```
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*.

3.2.19.3 Test an info structure for *required* directive

Summary

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

PMIx v2.0

C

```
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.

3.2.19.4 Test an info structure for *optional* directive

Summary

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

PMIx v2.0

C

```
PMIX_INFO_IS_OPTIONAL(info);
```

C

IN **info**
Pointer to the **pmix_info_t** (pointer to **pmix_info_t**)

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

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

Summary

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

PMIx v2.2

C

```
PMIX_INFO_IS_END(info);
```

C

IN **info**
Pointer to the **pmix_info_t** (pointer to **pmix_info_t**)

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

3.2.20 Job Allocation Directives

PMIx v2.0

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 **pmix_alloc_directive_t** . All definitions were introduced in version 2 of the standard unless otherwise marked.

PMIX_ALLOC_NEW A new allocation is being requested. The resulting allocation will be disjoint (i.e., not connected in a job sense) from the requesting allocation.

PMIX_ALLOC_EXTEND Extend the existing allocation, either in time or as additional resources.

PMIX_ALLOC_RELEASE Release part of the existing allocation. Attributes in the accompanying **pmix_info_t** array may be used to specify permanent release of the identified resources, or "lending" of those resources for some period of time.

PMIX_ALLOC_REAQUIRE Reacquire resources that were previously "lent" back to the scheduler.

PMIX_ALLOC_EXTERNAL A value boundary above which implementers are free to define their own directive values.

3.2.21 IO Forwarding Channels

PMIx v3.0

The **pmix_iof_channel_t** structure is a **uint16_t** type that defines a set of bit-mask flags for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple channels.

PMIX_FWD_NO_CHANNELS Forward no channels

PMIX_FWD_STDIN_CHANNEL Forward stdin

PMIX_FWD_STDOUT_CHANNEL Forward stdout

PMIX_FWD_STDERR_CHANNEL Forward stderr

PMIX_FWD_STDDIAG_CHANNEL Forward stddiag, if available

PMIX_FWD_ALL_CHANNELS Forward all available channels

3.2.22 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.

```c
typedef struct
    char *envar;
    char *value;
    char separator;
 pmix_envar_t;
```

3.2.23 Environmental variable support macros

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

3.2.23.1 Initialize the pmix_envar_t structure

Initialize the **pmix_envar_t** fields

PMIx v3.0

```c
PMIX_ENVAR_CONSTRUCT(m)
```

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

3.2.23.2 Destruct the pmix_envar_t structure

Clear the **pmix_envar_t** fields

PMIx v3.0

```c
PMIX_ENVAR_DESTRUCT(m)
```

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

3.2.23.3 Create a pmix_envar_t array

Allocate and initialize an array of **pmix_envar_t** structures

PMIx v3.0

```c
PMIX_ENVAR_CREATE(m, n)
```

INOUT **m**
Address where the pointer to the array of **pmix_envar_t** structures shall be stored (handle)

IN **n**
Number of structures to be allocated (**size_t**)

3.2.23.4 Free a pmix_envar_t array

Release an array of **pmix_envar_t** structures

PMIx v3.0

C

```
PMIX_ENVAR_FREE(m, n)
```

C

IN **m**
Pointer to the array of **pmix_envar_t** structures (handle)
IN **n**
Number of structures in the array (**size_t**)

3.2.23.5 Load a pmix_envar_t structure

Load values into a **pmix_envar_t**

PMIx v2.0

C

```
PMIX_ENVAR_LOAD(m, e, v, s)
```

C

IN **m**
Pointer to the structure to be loaded (pointer to **pmix_envar_t**)
IN **e**
Environmental variable name (**char***)
IN **v**
Value of variable (**char***)
IN **v**
Separator character (**char**)

3.2.24 Lookup Returned Data Structure

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

PMIx v1.0

C

```
typedef struct pmix_pdata {
    pmix_proc_t proc;
    pmix_key_t key;
    pmix_value_t value;
} pmix_pdata_t;
```

C

3.2.25 Lookup data structure support macros

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

3.2.25.1 Initialize the pmix_pdata_t structure

Initialize the **pmix_pdata_t** fields

PMIx v1.0

C

```
PMIX_PDATA_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_pdata_t**)

3.2.25.2 Destruct the pmix_pdata_t structure

Destruct the **pmix_pdata_t** fields

PMIx v1.0

C

```
PMIX_PDATA_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_pdata_t**)

3.2.25.3 Create a pmix_pdata_t array

Allocate and initialize an array of **pmix_pdata_t** structures

PMIx v1.0

C

```
PMIX_PDATA_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_pdata_t** structures shall be stored (handle)
IN **n**
Number of structures to be allocated (**size_t**)

3.2.25.4 Free a pmix_pdata_t array

Release an array of **pmix_pdata_t** structures

PMIx v1.0

C

```
PMIX_PDATA_FREE(m, n)
```

C

IN **m**
Pointer to the array of **pmix_pdata_t** structures (handle)
IN **n**
Number of structures in the array (**size_t**)

3.2.25.5 Load a lookup data structure

Summary

Load key, process identifier, and data value into a **pmix_pdata_t** structure.

PMIx v1.0

C

```
PMIX_PDATA_LOAD(m, p, k, d, t);
```

C

IN **m**
Pointer to the **pmix_pdata_t** structure into which the key and data are to be loaded (pointer to **pmix_pdata_t**)

IN **p**
Pointer to the **pmix_proc_t** structure containing the identifier of the process being referenced (pointer to **pmix_proc_t**)

IN **k**
String key to be loaded - must be less than or equal to **PMIX_MAX_KEYLEN** in length (handle)

IN **d**
Pointer to the data value to be loaded (handle)

IN **t**
Type of the provided data value (**pmix_data_type_t**)

This macro simplifies the loading of key, process identifier, and data into a **pmix_proc_t** by correctly assigning values to the structure's fields.

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.

3.2.25.6 Transfer a lookup data structure

Summary

Transfer key, process identifier, and data value between two **pmix_pdata_t** structures.

PMIx v2.0

C

```
PMIX_PDATA_XFER(d, s);
```

C

IN **d**
Pointer to the destination **pmix_pdata_t** (pointer to **pmix_pdata_t**)

IN **s**
Pointer to the source **pmix_pdata_t** (pointer to **pmix_pdata_t**)

This macro simplifies the transfer of key and data between two **pmix_pdata_t** structures.

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.

3.2.26 Application Structure

The **pmix_app_t** structure describes the application context for the **PMIx_Spawn** and **PMIx_Spawn_nb** operations.

PMIx v1.0

C

```
typedef struct pmix_app {
    /** Executable */
    char *cmd;
    /** Argument set, NULL terminated */
    char **argv;
    /** Environment set, NULL terminated */
    char **env;
    /** Current working directory */
    char *cwd;
    /** Maximum processes with this profile */
    int maxprocs;
    /** Array of info keys describing this application*/
    pmix_info_t *info;
    /** Number of info keys in 'info' array */
    size_t ninfo;
} pmix_app_t;
```

C

3.2.27 App structure support macros

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

3.2.27.1 Initialize the pmix_app_t structure

Initialize the **pmix_app_t** fields

PMIx v1.0

C

```
PMIX_APP_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_app_t**)

3.2.27.2 Destruct the pmix_app_t structure

Destruct the **pmix_app_t** fields

PMIx v1.0

C

```
PMIX_APP_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_app_t**)

3.2.27.3 Create a pmix_app_t array

Allocate and initialize an array of **pmix_app_t** structures

PMIx v1.0

C

```
PMIX_APP_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_app_t** structures shall be stored (handle)

IN **n**
Number of structures to be allocated (**size_t**)

3.2.27.4 Free a pmix_app_t array

Release an array of **pmix_app_t** structures

PMIx v1.0

C

```
PMIX_APP_FREE(m, n)
```

C

IN **m**
Pointer to the array of **pmix_app_t** structures (handle)

IN **n**
Number of structures in the array (**size_t**)

3.2.27.5 Create the pmix_info_t array of application directives

Create an array of **pmix_info_t** structures for passing application-level directives, updating the *ninfo* field of the **pmix_app_t** structure.

PMIx v2.2

C

```
PMIX_APP_INFO_CREATE(m, n)
```

C

IN **m**
Pointer to the **pmix_app_t** structure (handle)

IN **n**
Number of directives to be allocated (**size_t**)

3.2.28 Query Structure

The **pmix_query_t** structure is used by **PMIx_Query_info_nb** to describe a single query operation.

PMIx v2.0

C

```
typedef struct pmix_query {
    char **keys;
    pmix_info_t *qualifiers;
    size_t nqual;
} pmix_query_t;
```

C

3.2.29 Query structure support macros

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

3.2.29.1 Initialize the pmix_query_t structure

Initialize the **pmix_query_t** fields

PMIx v2.0

C

```
PMIX_QUERY_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_query_t**)

3.2.29.2 Destruct the pmix_query_t structure

Destruct the **pmix_query_t** fields

PMIx v2.0

C

```
PMIX_QUERY_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_query_t**)

3.2.29.3 Create a pmix_query_t array

Allocate and initialize an array of **pmix_query_t** structures

PMIx v2.0

C

```
PMIX_QUERY_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_query_t** structures shall be stored (handle)
IN **n**
Number of structures to be allocated (**size_t**)

3.2.29.4 Free a pmix_query_t array

Release an array of **pmix_query_t** structures

PMIx v2.0

C

```
PMIX_QUERY_FREE(m, n)
```

C

IN **m**
Pointer to the array of **pmix_query_t** structures (handle)
IN **n**
Number of structures in the array (**size_t**)

3.2.29.5 Create the pmix_info_t array of query qualifiers

Create an array of **pmix_info_t** structures for passing query qualifiers, updating the *nqual* field of the **pmix_query_t** structure.

PMIx v2.2

C

```
PMIX_QUERY_QUALIFIERS_CREATE(m, n)
```

C

IN **m**
Pointer to the **pmix_query_t** structure (handle)
IN **n**
Number of qualifiers to be allocated (**size_t**)

3.2.30 Attribute registration structure

The **pmix_regattr_t** structure is used to register attribute support for a PMIx function.

PMIx v4.0

C

```
typedef struct pmix_regattr {
    char *name;
    pmix_key_t *string;
    pmix_data_type_t type;
    pmix_info_t *info;
    size_t ninfo;
    char **description;
} pmix_regattr_t;;
```

C

Note that in this structure:

- the *name* is the actual name of the attribute - e.g., "PMIX_MAX_PROCS"; and
- the *string* is the literal string value of the attribute - e.g., "pmix.max.size" for the **PMIX_MAX_PROCS** attribute
- *type* must be a PMIx data type identifying the type of data associated with this attribute.
- the *info* array contains machine-usable information regarding the range of accepted values. This may include entries for **PMIX_MIN_VALUE** , **PMIX_MAX_VALUE** , **PMIX_ENUM_VALUE** , or a combination of them. For example, an attribute that supports all positive integers might delineate it by including a **pmix_info_t** with a key of **PMIX_MIN_VALUE** , type of **PMIX_INT** , and value of zero. The lack of an entry for **PMIX_MAX_VALUE** indicates that there is no ceiling to the range of accepted values.
- *ninfo* indicates the number of elements in the *info* array
- The *description* field consists of a **NULL**-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values - e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the *description* and the number of elements in the *info* array is implied or required.

The attribute *name* and *string* fields must be **NULL**-terminated strings composed of standard alphanumeric values supported by common utilities such as *strcmp*.

Advice to PMIx library implementers

Although not strictly required, PMIx library implementers are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes.

Advice to PMIx server hosts

Although not strictly required, host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them.

3.2.31 Attribute registration structure support macros

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

3.2.31.1 Initialize the pmix_regattr_t structure

Initialize the **pmix_regattr_t** fields

PMIx v4.0

C

```
PMIX_REGATTR_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_regattr_t**)

3.2.31.2 Destruct the pmix_regattr_t structure

Destruct the **pmix_regattr_t** fields, releasing all strings.

PMIx v4.0

C

```
PMIX_REGATTR_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_regattr_t**)

3.2.31.3 Create a pmix_regattr_t array

Allocate and initialize an array of **pmix_regattr_t** structures

PMIx v4.0

C

```
PMIX_REGATTR_CREATE(m, n)
```

C

INOUT **m**
Address where the pointer to the array of **pmix_regattr_t** structures shall be stored (handle)

IN **n**
Number of structures to be allocated (**size_t**)

3.2.31.4 Free a pmix_regattr_t array

Release an array of **pmix_regattr_t** structures

PMIx v4.0

C

```
PMIX_REGATTR_FREE(m, n)
```

C

INOUT **m**
: Pointer to the array of **pmix_regattr_t** structures (handle)

IN **n**
: Number of structures in the array (**size_t**)

3.2.31.5 Load a pmix_regattr_t structure

Load values into a **pmix_regattr_t** structure. The macro can be called multiple times to add as many strings as desired to the same structure by passing the same address and a **NULL** key to the macro. Note that the *t* type value must be given each time.

PMIx v4.0

C

```
PMIX_REGATTR_LOAD(a, n, k, t, ni, v)
```

C

IN **a**
: Pointer to the structure to be loaded (pointer to **pmix_proc_t**)

IN **n**
: String name of the attribute (string)

IN **k**
: Key value to be loaded (**pmix_key_t**)

IN **t**
: Type of data associated with the provided key (**pmix_data_type_t**)

IN **ni**
: Number of **pmix_info_t** elements to be allocated in *info* (**size_t**)

IN **v**
: One-line description to be loaded (more can be added separately) (string)

3.2.31.6 Transfer a pmix_regattr_t to another pmix_regattr_t

Non-destructively transfer the contents of a **pmix_regattr_t** structure to another one.

PMIx v4.0

C

```
PMIX_REGATTR_XFER(m, n)
```

C

INOUT **m**
: Pointer to the destination **pmix_regattr_t** structure (handle)

IN **m**
: Pointer to the source **pmix_regattr_t** structure (handle)

3.2.32 PMIx Group Directives

PMIx v4.0

The **pmix_group_opt_t** type is an enumerated type used with the **PMIx_Group_join** API to indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

PMIX_GROUP_DECLINE Decline the invitation
PMIX_GROUP_ACCEPT Accept the invitation.

3.2.33 Byte Object Type

The **pmix_byte_object_t** structure describes a raw byte sequence.

PMIx v1.0

C

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

C

3.2.34 Byte object support macros

The following macros support the **pmix_byte_object_t** structure.

3.2.34.1 Initialize the pmix_byte_object_t structure

Initialize the **pmix_byte_object_t** fields

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_byte_object_t**)

3.2.34.2 Destruct the pmix_byte_object_t structure

Clear the **pmix_byte_object_t** fields

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_byte_object_t**)

3.2.34.3 Create a pmix_byte_object_t structure

Allocate and intitialize an array of **pmix_byte_object_t** structures

PMIx v2.0

C

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

C

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

IN `n`
: Number of structures to be allocated (**size_t**)

3.2.34.4 Free a pmix_byte_object_t array

Release an array of **pmix_byte_object_t** structures

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_FREE(m, n)
```

C

IN `m`
: Pointer to the array of **pmix_byte_object_t** structures (handle)

IN `n`
: Number of structures in the array (**size_t**)

3.2.34.5 Load a pmix_byte_object_t structure

Load values into a **pmix_byte_object_t**

PMIx v2.0

C

```
PMIX_BYTE_OBJECT_LOAD(b, d, s)
```

C

IN `b`
: Pointer to the structure to be loaded (pointer to **pmix_byte_object_t**)

IN `d`
: Pointer to the data to be loaded (**char***)

IN `s`
: Number of bytes in the data array (**size_t**)

3.2.35 Data Array Structure

The **pmix_data_array_t** structure defines an array data structure.

PMIx v2.0

C

```
typedef struct pmix_data_array {
    pmix_data_type_t type;
    size_t size;
    void *array;
} pmix_data_array_t;
```

C

3.2.36 Data array support macros

The following macros support the **pmix_data_array_t** structure.

3.2.36.1 Initialize a pmix_data_array_t structure

Initialize the **pmix_data_array_t** fields, allocating memory for the array of the indicated type.

PMIx v2.2

C

```
PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_data_array_t**)

IN **n**
Number of elements in the array (**size_t**)

IN **t**
PMIx data type of the array elements (**pmix_data_type_t**)

3.2.36.2 Destruct a pmix_data_array_t structure

Destruct the **pmix_data_array_t** , releasing the memory in the array.

PMIx v2.2

C

```
PMIX_DATA_ARRAY_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_data_array_t**)

3.2.36.3 Create a pmix_data_array_t structure

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

PMIx v2.2

C

```
PMIX_DATA_ARRAY_CREATE(m, n, t)
```

C

INOUT `m`
: Variable to be set to the address of the structure (pointer to **pmix_data_array_t**)

IN `n`
: Number of elements in the array (**size_t**)

IN `t`
: PMIx data type of the array elements (**pmix_data_type_t**)

3.2.36.4 Free a pmix_data_array_t structure

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

PMIx v2.2

C

```
PMIX_DATA_ARRAY_FREE(m)
```

C

IN `m`
: Pointer to the structure to be released (pointer to **pmix_data_array_t**)

3.2.37 Argument Array Macros

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

3.2.37.1 Argument array extension

Summary

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

C

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

C

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)

Description

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 in the way that the PRI expects it to be constructed.

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.

3.2.37.2 Argument array extension - unique

Summary

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

C

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

C

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)

Description

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 in the way that the PRI expects it to be constructed.

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.

3.2.37.3 Argument array release

Summary

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

C

```
PMIX_ARGV_FREE(a);
```

C

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

Description

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

3.2.37.4 Argument array split

Summary

Split a string into a NULL-terminated argv array.

C

```
PMIX_ARGV_SPLIT(a, b, c);
```

C

OUT **a**
Resulting argv-style array (**char****)
IN **b**
String to be split (**char***)
IN **c**
Delimiter character (**char**)

Description

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

Advice to users

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

3.2.37.5 Argument array join

Summary

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

C

```
PMIX_ARGV_JOIN(a, b, c);
```

C

OUT `a`
: Resulting string (`char*`)

IN `b`
: Argv-style array to be joined (`char**`)

IN `c`
: Delimiter character (`char`)

Description

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

3.2.37.6 Argument array count

Summary

Return the length of a NULL-terminated argv array.

C

```
PMIX_ARGV_COUNT(r, a);
```

C

OUT `r`
: Number of strings in the array (integer)

IN `a`
: Argv-style array (`char**`)

Description

Count the number of elements in an argv array

3.2.37.7 Argument array copy

Summary

Copy an argv array, including copying all off its strings.

C

```
PMIX_ARGV_COPY(a, b);
```

C

OUT `a`
: New argv-style array (`char**`)

IN `b`
: Argv-style array (`char**`)

Description

Copy an argv array, including copying all off its strings.

3.2.38 Set Environment Variable

Summary

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

C

```
PMIX_SETENV(r, name, value, env);
```

C

OUT `r`
: Status code indicating success or failure of the operation (**pmix_status_t**)

IN `name`
: Argument name (string)

IN `value`
: Argument value (string)

INOUT `env`
: Environment array to update (pointer to array of strings)

Description

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

Advice to users

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

3.3 Generalized Data Types Used for Packing/Unpacking

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

Advice to PMIx library implementers

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

PMIX_UNDEF Undefined
PMIX_BOOL Boolean (converted to/from native **true**/**false**) (**bool**)
PMIX_BYTE A byte of data (**uint8_t**)
PMIX_STRING **NULL** terminated string (**char***)
PMIX_SIZE Size **size_t**
PMIX_PID Operating process identifier (PID) (**pid_t**)
PMIX_INT Integer (**int**)
PMIX_INT8 8-byte integer (**int8_t**)
PMIX_INT16 16-byte integer (**int16_t**)
PMIX_INT32 32-byte integer (**int32_t**)
PMIX_INT64 64-byte integer (**int64_t**)
PMIX_UINT Unsigned integer (**unsigned int**)
PMIX_UINT8 Unsigned 8-byte integer (**uint8_t**)
PMIX_UINT16 Unsigned 16-byte integer (**uint16_t**)
PMIX_UINT32 Unsigned 32-byte integer (**uint32_t**)
PMIX_UINT64 Unsigned 64-byte integer (**uint64_t**)
PMIX_FLOAT Float (**float**)
PMIX_DOUBLE Double (**double**)
PMIX_TIMEVAL Time value (**struct timeval**)
PMIX_TIME Time (**time_t**)
PMIX_STATUS Status code **pmix_status_t**
PMIX_VALUE Value (**pmix_value_t**)
PMIX_PROC Process (**pmix_proc_t**)
PMIX_APP Application context
PMIX_INFO Info object
PMIX_PDATA Pointer to data
PMIX_BUFFER Buffer
PMIX_BYTE_OBJECT Byte object (**pmix_byte_object_t**)
PMIX_KVAL Key/value pair

PMIX_PERSIST Persistance (**pmix_persistence_t**)
PMIX_POINTER Pointer to an object (**void***)
PMIX_SCOPE Scope (**pmix_scope_t**)
PMIX_DATA_RANGE Range for data (**pmix_data_range_t**)
PMIX_COMMAND PMIx command code (used internally)
PMIX_INFO_DIRECTIVES Directives flag for **pmix_info_t** (**pmix_info_directives_t**)
PMIX_DATA_TYPE Data type code (**pmix_data_type_t**)
PMIX_PROC_STATE Process state (**pmix_proc_state_t**)
PMIX_PROC_INFO Process information (**pmix_proc_info_t**)
PMIX_DATA_ARRAY Data array (**pmix_data_array_t**)
PMIX_PROC_RANK Process rank (**pmix_rank_t**)
PMIX_QUERY Query structure (**pmix_query_t**)
PMIX_COMPRESSED_STRING String compressed with zlib (**char***)
PMIX_ALLOC_DIRECTIVE Allocation directive (**pmix_alloc_directive_t**)
PMIX_IOF_CHANNEL Input/output forwarding channel (**pmix_iof_channel_t**)
PMIX_ENVAR Environmental variable structure (**pmix_envar_t**)
PMIX_COORD Structure containing network coordinates (**pmix_coord_t**)
PMIX_REGATTR Structure supporting attribute registrations (**pmix_regattr_t**)
PMIX_REGEX Regular expressions - can be a valid NULL-terminated string or an arbitrary array of bytes

3.4 Reserved attributes

The PMIx standard defines a relatively small set of APIs and the caller may customize the behavior of the API by passing one or more attributes to that API. Additionally, attributes may be keys passed to **PMIx_Get** calls to access the specified values from the system.

Each attribute is represented by a *key* string, and a type for the associated *value*. This section defines a set of **reserved** keys which are prefixed with **pmix.** to designate them as PMIx standard reserved keys. All definitions were introduced in version 1 of the standard unless otherwise marked.

Applications or associated libraries (e.g., MPI) may choose to define additional attributes. The attributes defined in this section are of the system and job as opposed to the attributes that the application (or associated libraries) might choose to expose. Due to this extensibility the **PMIx_Get** API will return **PMIX_ERR_NOT_FOUND** if the provided *key* cannot be found.

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

PMIX_ATTR_UNDEF NULL (NULL)
Constant representing an undefined attribute.

3.4.1 Initialization attributes

These attributes are defined to assist the caller with initialization by passing them into the appropriate initialization API - thus, they are not typically accessed via the **PMIx_Get** API.

PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)
Pointer to libevent[1] **event_base** to use in place of the internal progress thread.

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)
Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device.

PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)
The host RM wants to declare itself as being the local system server for PMIx connection requests.

PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)
Top-level temporary directory for all client processes connected to this server, and where the PMIx server will place its tool rendezvous point and contact information.

PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
Temporary directory for this system, and where a PMIx server that declares itself to be a system-level server will place a tool rendezvous point and contact information.

PMIX_SERVER_ENABLE_MONITORING "pmix.srv.monitor" (bool)
Enable PMIx internal monitoring by the PMIx server.

PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
Name of the namespace to use for this PMIx server.

PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
Rank of this PMIx server

PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)
Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes (e.g., logging to email)

3.4.2 Tool-related attributes

These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server by passing them into the **PMIx_tool_init** API - thus, they are not typically accessed via the **PMIx_Get** API.

PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
Name of the namespace to use for this tool.

PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
Rank of this tool.

PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
PID of the target PMIx server for a tool.

PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)

[1] http://libevent.org/

The requestor requires that a connection be made only to a local, system-level PMIx server.

PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool)
Preferentially, look for a system-level PMIx server first.

PMIX_SERVER_URI "pmix.srvr.uri" (char*)
uniform resource identifier (URI) of the PMIx server to be contacted.

PMIX_SERVER_HOSTNAME "pmix.srvr.host" (char*)
Host where target PMIx server is located.

PMIX_CONNECT_MAX_RETRIES "pmix.tool.mretries" (uint32_t)
Maximum number of times to try to connect to PMIx server.

PMIX_CONNECT_RETRY_DELAY "pmix.tool.retry" (uint32_t)
Time in seconds between connection attempts to a PMIx server.

PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

PMIX_RECONNECT_SERVER "pmix.tool.recon" (bool)
Tool is requesting to change server connections

PMIX_LAUNCHER "pmix.tool.launcher" (bool)
Tool is a launcher and needs rendezvous files created

3.4.3 Identification attributes

These attributes are defined to identify a process and it's associated PMIx-enabled library. They are not typically accessed via the **PMIx_Get** API, and thus are not associated with a particular rank.

PMIX_USERID "pmix.euid" (uint32_t)
Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group id.

PMIX_DSTPATH "pmix.dstpath" (char*)
Path to shared memory data storage (dstore) files.

PMIX_VERSION_INFO "pmix.version" (char*)
PMIx version of contractor.

PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)
The requesting process is a PMIx tool.

PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
The requesting process is a PMIx client.

PMIX_PSET_NAME "pmix.pset.nm" (char*)
User-assigned name for the process set containing the given process.

3.4.4 Programming model attributes

These attributes are associated with programming models.

PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*)
Programming model being initialized (e.g., “MPI” or “OpenMP”)

PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*)
Programming model implementation ID (e.g., “OpenMPI” or “MPICH”)

PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)
Programming model version string (e.g., “2.1.1”)

PMIX_THREADING_MODEL "pmix.threads" (char*)
Threading model used (e.g., “pthreads”)

PMIX_MODEL_NUM_THREADS "pmix.mdl.nthrds" (uint64_t)
Number of active threads being used by the model

PMIX_MODEL_NUM_CPUS "pmix.mdl.ncpu" (uint64_t)
Number of cpus being used by the model

PMIX_MODEL_CPU_TYPE "pmix.mdl.cputype" (char*)
Granularity - “hwthread”, “core”, etc.

PMIX_MODEL_PHASE_NAME "pmix.mdl.phase" (char*)
User-assigned name for a phase in the application execution (e.g., “cfd reduction”)

PMIX_MODEL_PHASE_TYPE "pmix.mdl.ptype" (char*)
Type of phase being executed (e.g., “matrix multiply”)

PMIX_MODEL_AFFINITY_POLICY "pmix.mdl.tap" (char*)
Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread located on cpu close to master thread), "spread" (threads load-balanced across available cpus)

3.4.5 UNIX socket rendezvous socket attributes

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

PMIX_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.

3.4.6 TCP connection attributes

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

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

PMIX_TCP_URI "pmix.tcp.uri" (char*)
: The URI of the PMIx server to connect to, or a file name containing it in the form of **file:<name of file containing it>**.

PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*)
: Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to include when establishing the TCP connection.

PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*)
: Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection.

PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)
: The IPv4 port to be used.

PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)
: The IPv6 port to be used.

PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)
: Set to **true** to disable IPv4 family of addresses.

PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)
: Set to **true** to disable IPv6 family of addresses.

3.4.7 Global Data Storage (GDS) attributes

These attributes are used to define the behavior of the GDS used to manage key/value pairs by passing them into the relevant initialization API - thus, they are not typically accessed via the **PMIx_Get** API.

PMIX_GDS_MODULE "pmix.gds.mod" (char*)
: Comma-delimited string of desired modules.

3.4.8 General process-level attributes

These attributes are used to define process attributes and are referenced by their process rank.

PMIX_CPUSET "pmix.cpuset" (char*)
: hwloc[2] bitmap to be applied to the process upon launch.

PMIX_CREDENTIAL "pmix.cred" (char*)
: Security credential assigned to the process.

PMIX_SPAWNED "pmix.spawned" (bool)

[2] https://www.open-mpi.org/projects/hwloc/

true if this process resulted from a call to **PMIx_Spawn** .

PMIX_ARCH "pmix.arch" (uint32_t)
Architecture flag.

3.4.9 Scratch directory attributes

These attributes are used to define an application scratch directory and are referenced using the **PMIX_RANK_WILDCARD** rank.

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 namespace, under **PMIX_TMPDIR** .

PMIX_PROCDIR "pmix.pdir" (char*)
Full path to the subdirectory under **PMIX_NSDIR** assigned to the process.

PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)
Resource Manager will clean session directories

3.4.10 Relative Rank Descriptive Attributes

These attributes are used to describe information about relative ranks as assigned by the RM, and thus are referenced using the process rank except where noted.

PMIX_CLUSTER_ID "pmix.clid" (char*)
A string name for the cluster this proc is executing on

PMIX_PROCID "pmix.procid" (pmix_proc_t)
Process identifier

PMIX_NSPACE "pmix.nspace" (char*)
Namespace of the job.

PMIX_JOBID "pmix.jobid" (char*)
Job identifier assigned by the scheduler.

PMIX_APPNUM "pmix.appnum" (uint32_t)
Application number within the job.

PMIX_RANK "pmix.rank" (pmix_rank_t)
Process rank within the job.

PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t)
Process rank spanning across all jobs in this session.

PMIX_APP_RANK "pmix.apprank" (pmix_rank_t)
Process rank within this application.

PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t)
Starting global rank of this job - referenced using **PMIX_RANK_WILDCARD** .

PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)
Local rank on this node within this job.

PMIX_NODE_RANK "pmix.nrank" (uint16_t)
Process rank on this node spanning all jobs.

PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)
Lowest rank on this node within this job - referenced using **PMIX_RANK_WILDCARD** .

PMIX_APPLDR "pmix.aldr" (pmix_rank_t)
Lowest rank in this application within this job - referenced using **PMIX_RANK_WILDCARD** .

PMIX_PROC_PID "pmix.ppid" (pid_t)
PID of specified process.

PMIX_SESSION_ID "pmix.session.id" (uint32_t)
Session identifier - referenced using **PMIX_RANK_WILDCARD** .

PMIX_NODE_LIST "pmix.nlist" (char*)
Comma-delimited list of nodes running processes for the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)
Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using **PMIX_RANK_WILDCARD** .

PMIX_HOSTNAME "pmix.hname" (char*)
Name of the host where the specified process is running.

PMIX_NODEID "pmix.nodeid" (uint32_t)
Node identifier where the specified process is located, expressed as the node's index (beginning at zero) in the array resulting from expansion of the **PMIX_NODE_MAP** regular expression for the **job**

PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
Comma-delimited list of ranks on this node within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)
Array of **pmix_proc_t** of all processes on the specified node - referenced using **PMIX_RANK_WILDCARD** .

PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)
Colon-delimited cpusets of local peers within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

PMIX_PROC_URI "pmix.puri" (char*)
URI containing contact information for a given process.

PMIX_LOCALITY "pmix.loc" (uint16_t)
Relative locality of the specified process to the requestor.

PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
Process identifier of the parent process of the calling process.

PMIX_EXIT_CODE "pmix.exit.code" (int)
Exit code returned when process terminated

3.4.11 Information retrieval attributes

The following attributes are used to specify the level of information (e.g., **session** , **job** , or **application**) being requested where ambiguity may exist - see 5.1.5 for examples of their use.

PMIX_SESSION_INFO "pmix.ssn.info" (bool)

Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO "pmix.job.info" (bool)
Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO "pmix.app.info" (bool)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO "pmix.node.info" (bool)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

3.4.12 Information storage attributes

The following attributes are used to assemble information by its level (e.g., **session** , **job** , or **application**) for storage where ambiguity may exist - see 11.1.3.1 for examples of their use.

PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)
Provide an array of **pmix_info_t** containing session-level information. The **PMIX_SESSION_ID** attribute is required to be included in the array.

PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
Provide an array of **pmix_info_t** containing job-level information. The **PMIX_SESSION_ID** attribute of the **session** containing the **job** is required to be included in the array whenever the PMIx server library may host multiple sessions (e.g., when executing with a host RM daemon). As information is registered one job (aka namespace) at a time via the **PMIx_server_register_nspace** API, there is no requirement that the array contain either the **PMIX_NSPACE** 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 app-level information. The **PMIX_NSPACE** or **PMIX_JOBID** attributes of the **job** containing the application, plus its **PMIX_APPNUM** attribute, are 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_NODE_INFO_ARRAY **"pmix.node.arr"** (**pmix_data_array_t**)
Provide an array of **pmix_info_t** containing node-level 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.

Note that these assemblages can be used hierarchically:

- a **PMIX_JOB_INFO_ARRAY** might contain multiple **PMIX_APP_INFO_ARRAY** elements, each describing values for a specific application within the job
- a **PMIX_JOB_INFO_ARRAY** could contain a **PMIX_NODE_INFO_ARRAY** for each node hosting processes from that job, each array describing job-level values for that node
- a **PMIX_SESSION_INFO_ARRAY** might contain multiple **PMIX_JOB_INFO_ARRAY** elements, each describing a job executing within the session. Each job array could, in turn, contain both application and node arrays, thus providing a complete picture of the active operations within the allocation

Advice to PMIx library implementers

PMIx implementations must be capable of properly parsing and storing any hierarchical depth of information arrays. The resulting stored values are must to be accessible via both **PMIx_Get** and **PMIx_Query_info_nb** APIs, assuming appropriate directives are provided by the caller.

3.4.13 Size information attributes

These attributes are used to describe the size of various dimensions of the PMIx universe - all are referenced using **PMIX_RANK_WILDCARD** .

PMIX_UNIV_SIZE **"pmix.univ.size"** (**uint32_t**)
Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is the equivalent to the combination of **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it is included in the Standard for historical reasons.

PMIX_JOB_SIZE **"pmix.job.size"** (**uint32_t**)
Total number of processes in this job across all contained applications. Note that this value can be different from **PMIX_MAX_PROCS** . For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running **job** on-they-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.

PMIX_JOB_NUM_APPS **"pmix.job.napps"** (**uint32_t**)
Number of applications in this job.

PMIX_APP_SIZE **"pmix.app.size"** (**uint32_t**)
Number of processes in this application.

PMIX_LOCAL_SIZE **"pmix.local.size"** (**uint32_t**)

Number of processes in this job or application on this node.

PMIX_NODE_SIZE "pmix.node.size" (uint32_t)
Number of processes across all jobs on this node.

PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description.

PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)
Number of slots allocated in this context (session, namespace, application, or node). Note that this attribute is the equivalent to **PMIX_MAX_PROCS** used in the corresponding context - it is included in the Standard for historical reasons.

PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)
Number of nodes in this session, or that are currently executing processes from the associated namespace or application.

3.4.14 Memory information attributes

These attributes are used to describe memory available and used in the system - all are referenced using **PMIX_RANK_WILDCARD** .

PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
Total available physical memory on this node.

PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float)
Megabytes of memory currently used by the RM daemon.

PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float)
Average Megabytes of memory used by client processes.

3.4.15 Topology information attributes

These attributes are used to describe topology information in the PMIx universe - all are referenced using **PMIX_RANK_WILDCARD** except where noted.

PMIX_LOCAL_TOPO "pmix.ltopo" (char*)
eXtensible Markup Language (XML) representation of local node topology.

PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
Pointer to the PMIx client's internal hwloc topology object.

PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*)
XML-based description of topology

PMIX_TOPOLOGY_FILE "pmix.topo.file" (char*)
Full path to file containing XML topology description

PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
Topology signature string.

PMIX_LOCALITY_STRING "pmix.locstr" (char*)

String describing a process's bound location - referenced using the process's rank. The string is of the form:
NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s
Where the **%s** is replaced with an integer index or inclusive range for hwloc. **NM** identifies the numa node(s). **SK** identifies the socket(s). **L3** identifies the L3 cache(s). **L2** identifies the L2 cache(s). **L1** identifies the L1 cache(s). **CR** identifies the cores(s). **HT** identifies the hardware thread(s). If your architecture does not have the specified hardware designation then it can be omitted from the signature.
For example: **NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39**.
This means numa node **0**, socket **0**, L3 caches **0,1,2,3,4**, L2 caches **0-4**, L1 caches **0-4**, cores **0,1,2,3,4**, and hardware threads **0-39**.

PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
Address of the HWLOC shared memory segment.

PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
Size of the HWLOC shared memory segment.

PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
Path to the HWLOC shared memory file.

PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
XML representation of local topology using HWLOC's v1.x format.

PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
XML representation of local topology using HWLOC's v2.x format.

PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)
Share the HWLOC topology via shared memory

PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)
Kind of VM "hole" HWLOC should use for shared memory

3.4.16 Request-related attributes

These attributes are used to influence the behavior of various PMIx operations - they do not represent values accessed using the **PMIx_Get** API.

PMIX_COLLECT_DATA "pmix.collect" (bool)
Collect data and return it at the end of the operation.

PMIX_TIMEOUT "pmix.timeout" (int)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_IMMEDIATE "pmix.immediate" (bool)
Specified operation should immediately return an error from the PMIx server if the requested data cannot be found - do not request it from the host RM.

PMIX_WAIT "pmix.wait" (int)
Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)
If **true**, indicates that the requested choice of algorithm is mandatory.

PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)
Notify the parent process upon termination of child job.

PMIX_RANGE "pmix.range" (pmix_data_range_t)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)
Value for calls to **PMIx_Publish** .

PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
Scope of the data to be found in a **PMIx_Get** call.

PMIX_OPTIONAL "pmix.optional" (bool)
Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.

PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)
Request that any pointers in the returned value point directly to values in the key-value store

PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)
Execute a blocking fence operation before executing the specified operation. For example, **PMIx_Finalize** does not include an internal barrier operation by default. This attribute would direct **PMIx_Finalize** to execute a barrier as part of the finalize operation.

PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)
Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the **PMIX_ERR_JOB_TERMINATED** event is optional and host environments may choose to provide it only upon request.

PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)
State of the specified process as of the last report - may not be the actual current state based on update rate.

PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)
Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the **PMIX_PROC_TERMINATED** event is optional and host environments may choose to provide it only upon request.

3.4.17 Server-to-PMIx library attributes

Attributes used by the host environment to pass data to its PMIx server library. The data will then be parsed and provided to the local PMIx clients. These attributes are all referenced using **PMIX_RANK_WILDCARD** except where noted.

PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)

Registration is for this namespace only, do not copy job data - this attribute is not accessed using the **PMIx_Get**

PMIX_PROC_DATA "pmix.pdata" (pmix_data_array_t)
Array of process data. Starts with rank, then contains more data.

PMIX_NODE_MAP "pmix.nmap" (char*)
Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.

PMIX_PROC_MAP "pmix.pmap" (char*)
Regular expression describing processes on each node - see 11.1.3.1 for an explanation of its generation.

PMIX_ANL_MAP "pmix.anlmap" (char*)
Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.

PMIX_APP_MAP_TYPE "pmix.apmap.type" (char*)
Type of mapping used to layout the application (e.g., **cyclic**).

PMIX_APP_MAP_REGEX "pmix.apmap.regex" (char*)
Regular expression describing the result of the process mapping.

3.4.18 Server-to-Client attributes

Attributes used internally to communicate data from the PMIx server to the PMIx client - they do not represent values accessed using the **PMIx_Get** API.

PMIX_PROC_BLOB "pmix.pblob" (pmix_byte_object_t)
Packed blob of process data.

PMIX_MAP_BLOB "pmix.mblob" (pmix_byte_object_t)
Packed blob of process location.

3.4.19 Event handler registration and notification attributes

Attributes to support event registration and notification - they are values passed to the event registration and notification APIs and therefore are not accessed using the **PMIx_Get** API.

PMIX_EVENT_HDLR_NAME "pmix.evname" (char*)
String name identifying this handler.

PMIX_EVENT_HDLR_FIRST "pmix.evfirst" (bool)
Invoke this event handler before any other handlers.

PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool)
Invoke this event handler after all other handlers have been called.

PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool)
Invoke this event handler before any other handlers in this category.

PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)
Invoke this event handler after all other handlers in this category have been called.

PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)
Put this event handler immediately before the one specified in the **(char*)** value.

PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)
Put this event handler immediately after the one specified in the **(char*)** value.

PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)
Prepend this handler to the precedence list within its category.
PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool)
Append this handler to the precedence list within its category.
PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)
Array of **pmix_proc_t** defining range of event notification.
PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)
The single process that was affected.
PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)
Array of **pmix_proc_t** defining affected processes.
PMIX_EVENT_NON_DEFAULT "pmix.evnondef" (bool)
Event is not to be delivered to default event handlers.
PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
Object to be returned whenever the registered callback function **cbfunc** is invoked. The object will only be returned to the process that registered it.
PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)
Instruct the PMIx server not to cache the event.
PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
Do not generate an event when this job normally terminates.
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

3.4.20 Fault tolerance attributes

Attributes to support fault tolerance behaviors - they are values passed to the event notification API and therefore are not accessed using the **PMIx_Get** API.

PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)
The RM intends to terminate this session.
PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)
The RM intends to terminate this job.
PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)
The RM intends to terminate all processes on this node.
PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool)
The RM intends to terminate just this process.
PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int)
The time in seconds before the RM will execute error response.
PMIX_EVENT_NO_TERMINATION "pmix.evnoterm" (bool)
Indicates that the handler has satisfactorily handled the event and believes termination of the application is not required.
PMIX_EVENT_WANT_TERMINATION "pmix.evterm" (bool)
Indicates that the handler has determined that the application should be terminated

3.4.21 Spawn attributes

Attributes used to describe **PMIx_Spawn** behavior - they are values passed to the **PMIx_Spawn** API and therefore are not accessed using the **PMIx_Get** API when used in that context. However, some of the attributes defined in this section can be provided by the host environment for other purposes - e.g., the environment might provide the **PMIX_MAPPER** attribute in the job-related information so that an application can use **PMIx_Get** to discover the layout algorithm used for determining process locations. Multi-use attributes and their respective access reference rank are denoted below.

PMIX_PERSONALITY "pmix.pers" (char*)
Name of personality to use.

PMIX_HOST "pmix.host" (char*)
Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE "pmix.hostfile" (char*)
Hostfile to use for spawned processes.

PMIX_ADD_HOST "pmix.addhost" (char*)
Comma-delimited list of hosts to add to the allocation.

PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)
Hostfile listing hosts to add to existing allocation.

PMIX_PREFIX "pmix.prefix" (char*)
Prefix to use for starting spawned processes.

PMIX_WDIR "pmix.wdir" (char*)
Working directory for spawned processes.

PMIX_MAPPER "pmix.mapper" (char*)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
Display process mapping upon spawn.

PMIX_PPR "pmix.ppr" (char*)
Number of processes to spawn on each identified resource.

PMIX_MAPBY "pmix.mapby" (char*)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY "pmix.rankby" (char*)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO "pmix.bindto" (char*)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
Preload binaries onto nodes.
PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
Comma-delimited list of files to pre-position on nodes.
PMIX_NON_PMI "pmix.nonpmi" (bool)
Spawned processes will not call **PMIx_Init** .
PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
Spawned process rank that is to receive **stdin**.
PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)
Forward this process's **stdin** to the designated process.
PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
Forward **stdout** from spawned processes to this process.
PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
Forward **stderr** from spawned processes to this process.
PMIX_FWD_STDDIAG "pmix.fwd.stddiag" (bool)
If a diagnostic channel exists, forward any output on it from the spawned processes to this process (typically used by a tool)
PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)
Spawned application consists of debugger daemons.
PMIX_COSPAWN_APP "pmix.cospawn" (bool)
Designated application is to be spawned as a disconnected job. Meaning that it is not part of the "comm_world" of the parent process.
PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace
PMIX_TAG_OUTPUT "pmix.tagout" (bool)
Tag application output with the identity of the source process.
PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
Timestamp output from applications.
PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
Merge **stdout** and **stderr** streams from application processes.
PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
Output application output to the specified file.
PMIX_INDEX_ARGV "pmix.indxargv" (bool)
Mark the **argv** with the rank of the process.
PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace
PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
Do not place processes on the head node.
PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)

Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
Report bindings of the individual processes.

PMIX_CPU_LIST "pmix.cpulist" (char*)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

PMIX_SPAWN_TOOL "pmix.spwn.tool" (bool)
Indicate that the job being spawned is a tool

3.4.22 Query attributes

Attributes used to describe **PMIx_Query_info_nb** behavior - these are values passed to the **PMIx_Query_info_nb** API and therefore are not passed to the **PMIx_Get** API.

PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)
Retrieve updated information from server.

PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*)
Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t)
Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*)
Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD)
Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_AUTHORIZATIONS "pmix.qry.auths" (bool)
Return operations the PMIx tool is authorized to perform.

PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
Return a comma-delimited list of supported debug attributes.
PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
Return information on memory usage for the processes indicated in the qualifiers.
PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
Constrain the query to local information only.
PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
Report only average values for sampled information.
PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
Report minimum and maximum values.
PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
String identifier of the allocation whose status is being requested.
PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.
PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
Query list of supported attributes for specified APIs
PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
Return the number of psets defined in the specified range (defaults to session).
PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (char*)
Return a comma-delimited list of the names of the psets defined in the specified range (defaults to session).

3.4.23 Log attributes

Attributes used to describe **PMIx_Log_nb** behavior - these are values passed to the **PMIx_Log_nb** API and therefore are not accessed using the **PMIx_Get** API.

PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
ID of source of the log request
PMIX_LOG_STDERR "pmix.log.stderr" (char*)
Log string to **stderr**.
PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
Log string to **stdout**.
PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog
PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)
Log data to local syslog. Defaults to **ERROR** priority.
PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.
PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)
Syslog priority level
PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
Timestamp for log report

PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)
Generate timestamp for log

PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
Label the output stream with the channel name (e.g., "stdout")

PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
Print timestamp in output string

PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
Print the output stream in XML format

PMIX_LOG_ONCE "pmix.log.once" (bool)
Only log this once with whichever channel can first support it, taking the channels in priority order

PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
Message blob to be sent somewhere.

PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SENDER_ADDR "pmix.log.emfaddr" (char*)
Return email address of sender

PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
Subject line for email.

PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
Message to be included in email.

PMIX_LOG_EMAIL_SERVER "pmix.log.esrvr" (char*)
Hostname (or IP address) of estmp server

PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (int32_t)
Port the email server is listening to

PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
Store the log data in a global data store (e.g., database)

PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
Log the provided information to the host environment's job record

3.4.24 Debugger attributes

Attributes used to assist debuggers - these are values that can be passed to the **PMIx_Spawn** or **PMIx_Init** APIs. Some may be accessed using the **PMIx_Get** API with the **PMIX_RANK_WILDCARD** rank.

PMIX_DEBUG_STOP_ON_EXEC "pmix.dbg.exec" (bool)
Passed to **PMIx_Spawn** to indicate that the specified application is being spawned under debugger, and that the launcher is to pause the resulting application processes on first instruction for debugger attach.

PMIX_DEBUG_STOP_IN_INIT "pmix.dbg.init" (bool)

Passed to **PMIx_Spawn** to indicate that the specified application is being spawned under debugger, and that the PMIx client library is to pause the resulting application processes during **PMIx_Init** until debugger attach and release.

PMIX_DEBUG_WAIT_FOR_NOTIFY "pmix.dbg.notify" (bool)
Passed to **PMIx_Spawn** to indicate that the specified application is being spawned under debugger, and that the resulting application processes are to pause at some application-determined location until debugger attach and release.

PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
Namespace of the job to be debugged - provided to the debugger upon launch.

PMIX_DEBUG_WAITING_FOR_NOTIFY "pmix.dbg.waiting" (bool)
Job to be debugged is waiting for a release - this is not a value accessed using the **PMIx_Get** API.

PMIX_DEBUG_JOB_DIRECTIVES "pmix.dbg.jdirs" (pmix_data_array_t*)
Array of job-level directives

PMIX_DEBUG_APP_DIRECTIVES "pmix.dbg.adirs" (pmix_data_array_t*)
Array of app-level directives

3.4.25 Resource manager attributes

Attributes used to describe the RM - these are values assigned by the host environment and accessed using the **PMIx_Get** API. The value of the provided namespace is unimportant but should be given as the namespace of the requesting process and a rank of **PMIX_RANK_WILDCARD** used to indicate that the information will be found with the job-level information.

PMIX_RM_NAME "pmix.rm.name" (char*)
String name of the RM.

PMIX_RM_VERSION "pmix.rm.version" (char*)
RM version string.

3.4.26 Environment variable attributes

Attributes used to adjust environment variables - these are values passed to the **PMIx_Spawn** API and are not accessed using the **PMIx_Get** API.

PMIX_SET_ENVAR "pmix.envar.set" (pmix_envar_t*)
Set the envar to the given value, overwriting any pre-existing one

PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)
Unset the environment variable specified in the string.

PMIX_ADD_ENVAR "pmix.envar.add" (pmix_envar_t*)
Add the environment variable, but do not overwrite any pre-existing one

PMIX_PREPEND_ENVAR "pmix.envar.prepnd" (pmix_envar_t*)
Prepend the given value to the specified environmental value using the given separator character, creating the variable if it doesn't already exist

PMIX_APPEND_ENVAR "pmix.envar.appnd" (pmix_envar_t*)
Append the given value to the specified environmental value using the given separator character, creating the variable if it doesn't already exist

3.4.27 Job Allocation attributes

Attributes used to describe the job allocation - these are values passed to and/or returned by the **PMIx_Allocation_request_nb** and **PMIx_Allocation_request** APIs and are not accessed using the **PMIx_Get** API

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_ID "**pmix.alloc.id**" (**char***)
: A string 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** .

PMIX_ALLOC_NUM_NODES "**pmix.alloc.nnodes**" (**uint64_t**)
: The number of nodes.

PMIX_ALLOC_NODE_LIST "**pmix.alloc.nlist**" (**char***)
: Regular expression of the specific nodes.

PMIX_ALLOC_NUM_CPUS "**pmix.alloc.ncpus**" (**uint64_t**)
: Number of cpus.

PMIX_ALLOC_NUM_CPU_LIST "**pmix.alloc.ncpulist**" (**char***)
: Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST "**pmix.alloc.cpulist**" (**char***)
: Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE "**pmix.alloc.msize**" (**float**)
: Number of Megabytes.

PMIX_ALLOC_NETWORK "**pmix.alloc.net**" (**array**)
: Array of **pmix_info_t** describing requested network resources. This must include at least: **PMIX_ALLOC_NETWORK_ID** , **PMIX_ALLOC_NETWORK_TYPE** , and **PMIX_ALLOC_NETWORK_ENDPTS** , plus whatever other descriptors are desired.

PMIX_ALLOC_NETWORK_ID "**pmix.alloc.netid**" (**char***)
: The key to be used when accessing this requested network allocation. The allocation will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the network support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: **PMIX_ALLOC_NETWORK_TYPE** - the type of resources provided; **PMIX_ALLOC_NETWORK_PLANE** - if applicable, what plane the resources were assigned from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the requested network allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

PMIX_ALLOC_BANDWIDTH "**pmix.alloc.bw**" (**float**)
: Mbits/sec.

PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
Quality of service level.
PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
Time in seconds.
PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)
Type of desired transport (e.g., *"tcp"*, *"udp"*)
PMIX_ALLOC_NETWORK_PLANE "pmix.alloc.netplane" (char*)
ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)
PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)
Number of endpoints to allocate per process
PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
Number of endpoints to allocate per node
PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
Network security key

3.4.28 Job control attributes

Attributes used to request control operations on an executing application - these are values passed to the **PMIx_Job_control_nb** API and are not accessed using the **PMIx_Get** API.

PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.
PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
Pause the specified processes.
PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
Resume ("un-pause") the specified processes.
PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
Cancel the specified request - the provided request ID must match the **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of **NULL** implies cancel all requests from this requestor.
PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
Forcibly terminate the specified processes and cleanup.
PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
Restart the specified processes using the given checkpoint ID.
PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
Checkpoint the specified processes and assign the given ID to it.
PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
Use event notification to trigger a process checkpoint.
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
Use the given signal to trigger a process checkpoint.
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.
PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (**int**)
Send given signal to specified processes.
PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (**char***)
Regular expression identifying nodes that are to be provisioned.
PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (**char***)
Name of the image that is to be provisioned.
PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (**bool**)
Indicate that the job can be pre-empted.
PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (**bool**)
Politely terminate the specified processes.
PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (**char***)
Comma-delimited list of files to be removed upon process termination
PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (**char***)
Comma-delimited list of directories to be removed upon process termination
PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (**bool**)
Recursively cleanup all subdirectories under the specified one(s)
PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (**bool**)
Only remove empty subdirectories
PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (**char***)
Comma-delimited list of filenames that are not to be removed
PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (**bool**)
When recursively cleaning subdirectories, do not remove the top-level directory (the one given in the cleanup request)

3.4.29 Monitoring attributes

Attributes used to control monitoring of an executing application- these are values passed to the **PMIx_Process_monitor_nb** API and are not accessed using the **PMIx_Get** API.

PMIX_MONITOR_ID "pmix.monitor.id" (**char***)
Provide a string identifier for this request.
PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (**char***)
Identifier to be canceled (**NULL** means cancel all monitoring for this process).
PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (**bool**)
The application desires to control the response to a monitoring event.
PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (**void**)
Register to have the PMIx server monitor the requestor for heartbeats.
PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (**void**)
Send heartbeat to local PMIx server.
PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (**uint32_t**)
Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
Number of heartbeats that can be missed before generating the event.
PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
Register to monitor file for signs of life.
PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
Monitor size of given file is growing to determine if the application is running.
PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
Monitor time since last access of given file to determine if the application is running.
PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
Monitor time since last modified of given file to determine if the application is running.
PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
Time in seconds between checking the file.
PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
Number of file checks that can be missed before generating the event.

3.4.30 Security attributes

PMIx v3.0 Attributes for managing security credentials

PMIX_CRED_TYPE "pmix.sec.ctype" (char*)
When passed in **PMIx_Get_credential** , a prioritized, comma-delimited list of desired credential types for use in environments where multiple authentication mechanisms may be available. When returned in a callback function, a string identifier of the credential type.
PMIX_CRYPTO_KEY "pmix.sec.key" (pmix_byte_object_t)
Blob containing crypto key

3.4.31 IO Forwarding attributes

PMIx v3.0 Attributes used to control IO forwarding behavior

PMIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t)
The requested size of the 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, drop the oldest bytes to make room in the cache.
PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)
In an overflow situation, drop any new bytes received until room becomes available in the cache (default).
PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)
Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon call to deregister the respective channel.
PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)

Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.

PMIX_IOF_COMPLETE "pmix.iof.cmp" (bool)
Indicates whether or not the specified IO channel has been closed by the source.

PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool)
Tag output with the channel it comes from.

PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts" (bool)
Timestamp output

PMIX_IOF_XML_OUTPUT "pmix.iof.xml" (bool)
Format output in XML

3.4.32 Application setup attributes

PMIx v3.0 Attributes 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

3.4.33 Attribute support level attributes

PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (bool)
Request a list of functions supported by the PMIx client library

PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
Request attributes supported by the PMIx client library

PMIX_SERVER_FUNCTIONS "pmix.srvr.fns" (bool)
Request a list of functions supported by the PMIx server library

PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
Request attributes supported by the PMIx server library

PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (bool)
Request a list of functions supported by the host environment

PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
Request attributes supported by the host environment

PMIX_TOOL_FUNCTIONS "pmix.tool.fns" (bool)
Request a list of functions supported by the PMIx tool library

PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
Request attributes supported by the PMIx tool library functions

3.4.34 Descriptive attributes

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.

3.4.35 Process group attributes

PMIx v4.0 Attributes for controlling the PMIx Group APIs

PMIX_GROUP_ID **"pmix.grp.id"** (**char***)
User-provided group identifier

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_NOTIFY_TERMINATION **"pmix.grp.notterm"** (**bool**)
Notify remaining members when another member terminates without first leaving the group. The default is false

PMIX_GROUP_INVITE_DECLINE **"pmix.grp.decline"** (**bool**)
Notify the inviting process that this process does not wish to participate in the proposed group The default is true

PMIX_GROUP_MEMBERSHIP **"pmix.grp.mbrs"** (**pmix_data_array_t***)
Array of group member ID's

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_CONTEXT_ID **"pmix.grp.ctxid"** (**size_t**)
Context identifier assigned to the group by the host RM.

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. The default is false

PMIX_GROUP_ENDPT_DATA **"pmix.grp.endpt"** (**pmix_byte_object_t**)
Data collected to be shared during group construction

3.5 Callback Functions

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

3.5.1 Release Callback Function

Summary

The **pmix_release_cbfunc_t** is used by the **pmix_modex_cbfunc_t** and **pmix_info_cbfunc_t** operations to indicate that the callback data may be reclaimed/freed by the caller.

Format

PMIx v1.0

C

```
typedef void (*pmix_release_cbfunc_t)
    (void *cbdata)
```

C

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

Description

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

3.5.2 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

```
typedef void (*pmix_modex_cbfunc_t)
     (pmix_status_t status,
      const char *data, size_t ndata,
      void *cbdata,
      pmix_release_cbfunc_t release_fn,
      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.

3.5.3 Spawn Callback Function

Summary

The **pmix_spawn_cbfunc_t** is used on the PMIx client side by **PMIx_Spawn_nb** and on the PMIx server side by **pmix_server_spawn_fn_t** .

PMIx v1.0

C

```
typedef void (*pmix_spawn_cbfunc_t)
     (pmix_status_t status,
      pmix_nspace_t nspace, void *cbdata);
```

C

IN **`status`**
Status associated with the operation (handle)
IN **`nspace`**
Namespace string (**`pmix_nspace_t`**)
IN **`cbdata`**
Callback data passed to original API call (memory reference)

Description

The callback will be executed upon launch of the specified applications in **`PMIx_Spawn_nb`** , or upon failure to launch any of them.

The *status* of the callback will indicate whether or not the spawn succeeded. The *nspace* of the spawned processes will be returned, along with any provided callback data. Note that the returned *nspace* value will not be protected by the PRI upon return from the callback function, so the receiver must copy it if it needs to be retained.

3.5.4 Op Callback Function

Summary

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

PMIx v1.0

C

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

C

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_connected_fn_t`** , **`PMIx_server_register_nspace`** . This callback function is used to return a status to an often nonblocking operation.

3.5.5 Lookup Callback Function

Summary

The **pmix_lookup_cbfunc_t** is used by **PMIx_Lookup_nb** to return data.

PMIx v1.0

C

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

C

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

IN **data**
Array of data returned (**pmix_pdata_t**)

IN **ndata**
Number of elements in the *data* array (**size_t**)

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

Description

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

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

3.5.6 Value Callback Function

Summary

The **pmix_value_cbfunc_t** is used by **PMIx_Get_nb** to return data.

PMIx v1.0

C

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

C

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.

3.5.7 Info Callback Function

Summary

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

PMIx v2.0

C

```
typedef void (*pmix_info_cbfunc_t)
    (pmix_status_t status,
     pmix_info_t info[], size_t ninfo,
     void *cbdata,
     pmix_release_cbfunc_t release_fn,
     void *release_cbdata);
```

C

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

IN **info**
Array of **pmix_info_t** returned by the operation (pointer)

IN **ninfo**
Number of elements in the *info* array (**size_t**)

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

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

IN **release_cbdata**
Callback data to be passed to *release_fn* (memory reference)

Description

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

3.5.8 Event Handler Registration Callback Function

The **pmix_evhdlr_reg_cbfunc_t** callback function.

Advice to users

The PMIx *ad hoc* v1.0 Standard defined an error handler registration callback function with a compatible signature, but with a different type definition function name (pmix_errhandler_reg_cbfunc_t). It was removed from the v2.0 Standard and is not included in this document to avoid confusion.

PMIx v2.0

C

```
typedef void (*pmix_evhdlr_reg_cbfunc_t)
     (pmix_status_t status,
      size_t evhdlr_ref,
      void *cbdata)
```

C

IN **status**
Status indicates if the request was successful or not (**pmix_status_t**)

IN **evhdlr_ref**
Reference assigned to the event handler by PMIx — this reference * must be used to deregister the err handler (**size_t**)

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

Description

Define a callback function for calls to **PMIx_Register_event_handler**

3.5.9 Notification Handler Completion Callback Function

Summary

The **pmix_event_notification_cbfunc_fn_t** is called by event handlers to indicate completion of their operations.

PMIx v2.0

C

```
typedef void (*pmix_event_notification_cbfunc_fn_t)
     (pmix_status_t status,
      pmix_info_t *results, size_t nresults,
      pmix_op_cbfunc_t cbfunc, void *thiscbdata,
      void *notification_cbdata);
```

C

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.

3.5.10 Notification Function

Summary

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

Advice to users

The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but 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.

PMIx v2.0

C

```
typedef void (*pmix_notification_fn_t)
     (size_t evhdlr_registration_id,
      pmix_status_t status,
      const pmix_proc_t *source,
      pmix_info_t info[], size_t ninfo,
      pmix_info_t results[], size_t nresults,
      pmix_event_notification_cbfunc_fn_t cbfunc,
      void *cbdata);
```

C

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)

Description

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

Advice to users

Possible uses of the *info* array include:

- for the host RM to alert the process as to planned actions, such as aborting the session, in response to the reported event
- provide a timeout for alternative action to occur, such as for the application to request an alternate response to the event

For example, the RM might alert the application to the failure of a node that resulted in termination of several processes, and indicate that the overall session will be aborted unless the application requests an alternative behavior in the next 5 seconds. The application then has time to respond with a checkpoint request, or a request to recover from the failure by obtaining replacement nodes and restarting from some earlier checkpoint.

Support for these options is left to the discretion of the host RM. Info keys are included in the common definitions above but may be augmented by environment vendors.

Advice to PMIx server hosts

On the server side, the notification function is used to inform the PMIx server library's host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the **pmix_server_notify_event_fn_t** function pointer, if provided.

3.5.11 Server Setup Application Callback Function

The **PMIx_server_setup_application** callback function.

Summary

Provide a function by which the resource manager can receive application-specific environmental variables and other setup data prior to launch of an application.

Format

PMIx v2.0

C

```
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)
```

C

IN **status**
returned status of the request (**pmix_status_t**)

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

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

IN **provided_cbdata**
Data originally passed to call to **PMIx_server_setup_application** (memory reference)

IN **cbfunc**
pmix_op_cbfunc_t function to be called when processing completed (function reference)

IN **cbdata**
Data to be passed to the *cbfunc* 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.

3.5.12 Server Direct Modex Response Callback Function

The **PMIx_server_dmodex_request** callback function.

Summary

Provide a function by which the local PMIx server library can return connection and other data posted by local application processes to the host resource manager.

Format

PMIx v1.0

C

```
typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,
                          char *data, size_t sz,
                          void *cbdata);
```

C

IN **status**
Returned status of the request (**pmix_status_t**)

IN **data**
Pointer to a data "blob" containing the requested information (handle)

IN **sz**
Number of bytes in the *data* blob (integer)

IN **cbdata**
Data passed into the initial call to **PMIx_server_dmodex_request** (memory reference)

Description

Define a function to be called by the PMIx server library for return of information posted by a local application process (via **PMIx_Put** with subsequent **PMIx_Commit**) in response to a request from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd upon return from the function.

3.5.13 PMIx Client Connection Callback Function

Summary

Callback function for incoming connection request from a local client

Format

PMIx v1.0

C

```
typedef void (*pmix_connection_cbfunc_t)(
                                int incoming_sd, void *cbdata)
```

C

IN **incoming_sd**
(integer)

IN **cbdata**
(memory reference)

Description

Callback function for incoming connection requests from local clients - only used by host environments that wish to directly handle socket connection requests.

3.5.14 PMIx Tool Connection Callback Function

Summary

Callback function for incoming tool connections.

Format

PMIx v2.0

C

```
typedef void (*pmix_tool_connection_cbfunc_t)(
                                pmix_status_t status,
                                pmix_proc_t *proc, void *cbdata)
```

C

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.

3.5.15 Credential callback function

Summary

Callback function to return a requested security credential

Format

PMIx v3.0

C

```
typedef void (*pmix_credential_cbfunc_t)(
                                pmix_status_t status,
                                pmix_byte_object_t *credential,
                                pmix_info_t info[], size_t ninfo,
                                void *cbdata)
```

C

IN **status**
pmix_status_t value (handle)

IN **credential**
pmix_byte_object_t structure containing the security credential (handle)

IN **info**
Array of provided by the system to pass any additional information about the credential - e.g., the identity of the issuing agent. (handle)

IN **ninfo**
Number of elements in *info* (**size_t**)

IN **cbdata**
Object passed in original request (memory reference)

Description

Define a callback function to return a requested security credential. Information provided by the issuing agent can subsequently be used by the application for a variety of purposes. Examples include:

- checking identified authorizations to determine what requests/operations are feasible as a means to steering **workflows**
- compare the credential type to that of the local SMS for compatibility

Advice to users

The credential is opaque and therefore understandable only by a service compatible with the issuer. The *info* array is owned by the PMIx library and is not to be released or altered by the receiving party.

3.5.16 Credential validation callback function

Summary

Callback function for security credential validation

Format

PMIx v3.0

C

```
typedef void (*pmix_validation_cbfunc_t)(
                                pmix_status_t status,
                                pmix_info_t info[], size_t ninfo,
                                void *cbdata);
```

C

IN **status**
pmix_status_t value (handle)

IN **info**
Array of **pmix_info_t** provided by the system to pass any additional information about the authentication - e.g., the effective userid and group id of the certificate holder, and any related authorizations (handle)

IN **ninfo**
Number of elements in *info* (**size_t**)

IN **cbdata**
Object passed in original request (memory reference)

Description

Define a validation callback function to indicate if a provided credential is valid, and any corresponding information regarding authorizations and other security matters.

Advice to users

The precise contents of the array will depend on the host environment and its associated security system. At the minimum, it is expected (but not required) that the array will contain entries for the **PMIX_USERID** and **PMIX_GRPID** of the client described in the credential. The *info* array is owned by the PMIx library and is not to be released or altered by the receiving party.

3.5.17 IOF delivery function

Summary

Callback function for delivering forwarded IO to a process

Format

PMIx v3.0

C

```
typedef void (*pmix_iof_cbfunc_t)(
                                size_t iofhdlr, pmix_iof_channel_t channel,
                                pmix_proc_t *source, char *payload,
                                pmix_info_t info[], size_t ninfo);
```

C

IN **iofhdlr**
Registration number of the handler being invoked (**size_t**)

IN **channel**
bitmask identifying the channel the data arrived on (**pmix_iof_channel_t**)

IN **source**
Pointer to a **pmix_proc_t** identifying the namespace/rank of the process that generated the data (**char***)

IN **payload**
Pointer to character array containing the data.

IN **info**
Array of **pmix_info_t** provided by the source containing metadata about the payload. This could include **PMIX_IOF_COMPLETE** (handle)

IN **ninfo**
Number of elements in *info* (**size_t**)

Description

Define a callback function for delivering forwarded IO to a process. This function will be called whenever data becomes available, or a specified buffering size and/or time has been met.

Advice to users

Multiple strings may be included in a given *payload*, and the *payload* may *not* be **NULL** terminated. The user is responsible for releasing the *payload* memory. The *info* array is owned by the PMIx library and is not to be released or altered by the receiving party.

3.5.18 IOF and Event registration function

Summary

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

Format

PMIx v3.0

C

```
typedef void (*pmix_hdlr_reg_cbfunc_t)(pmix_status_t status,
                                       size_t refid,
                                       void *cbdata);
```

C

IN **status**
PMIX_SUCCESS or an appropriate error constant (**pmix_status_t**)

IN **refid**
reference identifier assigned to the handler by PMIx, used to deregister the handler (**size_t**)

IN **cbdata**
object provided to the registration call (pointer)

Description

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

3.6 Constant String Functions

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

C

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

C

Summary

String representation of a **pmix_proc_state_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_scope_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_persistence_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_data_range_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_info_directives_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_data_type_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_alloc_directive_t** .

PMIx v2.0

C

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

C

Summary

String representation of a **pmix_iof_channel_t** .

PMIx v3.0

C

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

C

CHAPTER 4

Initialization and Finalization

The PMIx library is required to be initialized and finalized around the usage of most of the APIs. 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 universe. Each of these functional sets are described in this chapter. Note that a process can only call *one* of the init/finalize functional pairs - 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 finalize.

Advice to users

Processes that initialize as a server or tool automatically are given access to all client APIs. Server initialization includes setting up the infrastructure to support local clients - thus, it necessarily includes overhead and an increased memory footprint. Tool initialization automatically searches for a server to which it can connect — if declared as a *launcher*, the PMIx library sets up the required "hooks" for other tools (e.g., debuggers) to attach to it.

4.1 Query

The API defined in this section can be used by any PMIx process, regardless of their role in the PMIx universe.

4.1.1 PMIx_Initialized

PMIx v1.0

Format

C

```
int PMIx_Initialized(void)
```

C

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

Rationale

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

Description

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

4.1.2 PMIx_Get_version

Summary

Get the PMIx version information.

Format

PMIx v1.0

C

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

C

Description

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

4.2 Client Initialization and Finalization

Initialization and finalization routines for PMIx clients.

Advice to users

The PMIx *ad hoc* v1.0 Standard defined the **PMIx_Init** function, but modified the function signature in the v1.2 version. The *ad hoc* v1.0 version is not included in this document to avoid confusion.

4.2.1 PMIx_Init

Summary

Initialize the PMIx client library

Format

PMIx v1.2

C

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

C

INOUT `proc`
: proc structure (handle)

IN `info`
: Array of `pmix_info_t` structures (array of handles)

IN `ninfo`
: Number of element in the *info* array (`size_t`)

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

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

`PMIX_USOCK_DISABLE` `"pmix.usock.disable"` (`bool`)
: Disable legacy UNIX socket (usock) support If the library supports Unix socket connections, this attribute may be supported for disabling it.

`PMIX_SOCKET_MODE` `"pmix.sockmode"` (`uint32_t`)
: POSIX *mode_t* (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.

`PMIX_SINGLE_LISTENER` `"pmix.sing.listnr"` (`bool`)
: Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport. If the library supports multiple methods for clients to connect to servers, this attribute may be supported for disabling all but one of them.

`PMIX_TCP_REPORT_URI` `"pmix.tcp.repuri"` (`char*`)
: If provided, directs that the TCP URI be reported and indicates the desired method of reporting: `'-'` for stdout, `'+'` for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

`PMIX_TCP_IF_INCLUDE` `"pmix.tcp.ifinclude"` (`char*`)
: Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

`PMIX_TCP_IF_EXCLUDE` `"pmix.tcp.ifexclude"` (`char*`)
: Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are *not* to be used.

PMIX_TCP_IPV4_PORT **"pmix.tcp.ipv4"** (**int**)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT **"pmix.tcp.ipv6"** (**int**)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 **"pmix.tcp.disipv4"** (**bool**)
Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 **"pmix.tcp.disipv6"** (**bool**)
Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_EVENT_BASE **"pmix.evbase"** (**struct event_base ***)
Pointer to libevent[1] **event_base** to use in place of the internal progress thread.

PMIX_GDS_MODULE **"pmix.gds.mod"** (**char***)
Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

Description

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

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

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

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

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

[1] http://libevent.org/

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

4.2.2 PMIx_Finalize

Summary

Finalize the PMIx client library.

Format

PMIx v1.0

C

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

C

IN **info**
Array of **pmix_info_t** structures (array of handles)

IN **ninfo**
Number of element in the *info* array (**size_t**)

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

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_EMBED_BARRIER **"pmix.embed.barrier"** (**bool**)
Execute a blocking fence operation before executing the specified operation. For example, **PMIx_Finalize** does not include an internal barrier operation by default. This attribute would direct **PMIx_Finalize** to execute a barrier as part of the finalize operation.

Description

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

4.3 Tool Initialization and Finalization

Initialization and finalization routines for PMIx tools.

4.3.1 PMIx_tool_init

Summary

Initialize the PMIx library for operating as a tool.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_tool_init(pmix_proc_t *proc,
               pmix_info_t info[], size_t ninfo)
```

C

INOUT `proc`
: `pmix_proc_t` structure (handle)

IN `info`
: Array of `pmix_info_t` structures (array of handles)

IN `ninfo`
: Number of element in the *info* array (`size_t`)

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:

`PMIX_TOOL_NSPACE` `"pmix.tool.nspace"` (`char*`)
: Name of the namespace to use for this tool.

`PMIX_TOOL_RANK` `"pmix.tool.rank"` (`uint32_t`)
: Rank of this tool.

`PMIX_TOOL_DO_NOT_CONNECT` `"pmix.tool.nocon"` (`bool`)
: The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

`PMIX_SERVER_URI` `"pmix.srvr.uri"` (`char*`)
: URI of the PMIx server to be contacted.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

`PMIX_CONNECT_TO_SYSTEM` `"pmix.cnct.sys"` (`bool`)
: The requestor requires that a connection be made only to a local, system-level PMIx server.

`PMIX_CONNECT_SYSTEM_FIRST` `"pmix.cnct.sys.first"` (`bool`)
: Preferentially, look for a system-level PMIx server first.

`PMIX_SERVER_PIDINFO` `"pmix.srvr.pidinfo"` (`pid_t`)
: PID of the target PMIx server for a tool.

`PMIX_TCP_URI` `"pmix.tcp.uri"` (`char*`)
: The URI of the PMIx server to connect to, or a file name containing it in the form of `file:<name of file containing it>`.

`PMIX_CONNECT_RETRY_DELAY` `"pmix.tool.retry"` (`uint32_t`)

Time in seconds between connection attempts to a PMIx server.

PMIX_CONNECT_MAX_RETRIES **"pmix.tool.mretries"** (**uint32_t**)
Maximum number of times to try to connect to PMIx server.

PMIX_SOCKET_MODE **"pmix.sockmode"** (**uint32_t**)
POSIX *mode_t* (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.

PMIX_TCP_REPORT_URI **"pmix.tcp.repuri"** (**char***)
If provided, directs that the TCP URI be reported and indicates the desired method of reporting: **'-'** for stdout, **'+'** for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

PMIX_TCP_IF_INCLUDE **"pmix.tcp.ifinclude"** (**char***)
Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

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

PMIX_TCP_IPV4_PORT **"pmix.tcp.ipv4"** (**int**)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT **"pmix.tcp.ipv6"** (**int**)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 **"pmix.tcp.disipv4"** (**bool**)
Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 **"pmix.tcp.disipv6"** (**bool**)
Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_EVENT_BASE **"pmix.evbase"** (**struct event_base ***)
Pointer to libevent[2] **event_base** to use in place of the internal progress thread.

PMIX_GDS_MODULE **"pmix.gds.mod"** (**char***)
Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

[2] http://libevent.org/

Description

Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided **pmix_proc_t** struct. The *info* array is used to pass user requests pertaining to the init and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives are desired.

If called with the **PMIX_TOOL_DO_NOT_CONNECT** attribute, the PMIx tool library will fully initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later point in time, if desired. In all other cases, the PMIx tool library will attempt to connect to according to the following precedence chain:

- if **PMIX_SERVER_URI** or **PMIX_TCP_URI** is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. **PMIX_SERVER_URI** is the preferred method as it is more generalized — **PMIX_TCP_URI** is provided for those cases where the user specifically wants to use a TCP transport for the connection and wants to error out if it isn't available or cannot succeed. The PMIx library will return an error if connection fails — it will not proceed to check for other connection options as the user specified a particular one to use
- if **PMIX_SERVER_PIDINFO** was provided, then the tool will search under the directory provided by the PMIX_SERVER_TMPDIR environmental variable for a rendezvous file created by the process corresponding to that PID. The PMIx library will return an error if the rendezvous file cannot be found, or the connection is refused by the server
- 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 directory specified by the PMIX_SYSTEM_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. An error is returned if the rendezvous file cannot be found or the connection is refused.
- if **PMIX_CONNECT_SYSTEM_FIRST** is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX_SYSTEM_TMPDIR environmental variable. 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 directory provided by the PMIX_SERVER_TMPDIR environmental variable 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.

If successful, the function will return **PMIX_SUCCESS** and will fill the provided structure (if provided) with the server-assigned namespace and rank of the tool. Note that each connection attempt in the above precedence chain will retry (with delay between each retry) a number of times according to the values of the corresponding attributes. Default is no retries.

Note that the PMIx tool library is referenced counted, and so multiple calls to **PMIx_tool_init** are allowed. Thus, one way to obtain the namespace and rank of the process is to simply call **PMIx_tool_init** with a non-NULL parameter.

4.3.2 PMIx_tool_finalize

Summary

Finalize the PMIx library for a tool connection.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_tool_finalize(void)
```

C

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

Description

Finalize the PMIx tool library, closing the connection to the server. An error code will be returned if, for some reason, the connection cannot be cleanly terminated — in this case, the connection is dropped.

4.3.3 PMIx_tool_connect_to_server

Summary

Switch connection from the current PMIx server to another one, or initialize a connection to a specified server.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_tool_connect_to_server(pmix_proc_t *proc,
                            pmix_info_t info[], size_t ninfo)
```

C

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:

PMIX_CONNECT_TO_SYSTEM **"pmix.cnct.sys"** (**bool**)
: The requestor requires that a connection be made only to a local, system-level PMIx server.

PMIX_CONNECT_SYSTEM_FIRST **"pmix.cnct.sys.first"** (**bool**)
: Preferentially, look for a system-level PMIx server first.

PMIX_SERVER_URI **"pmix.srvr.uri"** (**char***)
: URI of the PMIx server to be contacted.

PMIX_SERVER_NSPACE **"pmix.srv.nspace"** (**char***)
: Name of the namespace to use for this PMIx server.

PMIX_SERVER_PIDINFO **"pmix.srvr.pidinfo"** (**pid_t**)
: PID of the target PMIx server for a tool.

Description

Switch connection from the current PMIx server to another one, or initialize a connection to a specified server. Closes the connection, if existing, to a server and establishes a connection to the specified server. This function can be called at any time by a PMIx tool to shift connections between servers. The process identifier assigned to this tool is returned in the provided **pmix_proc_t** struct. Passing a value of **NULL** for this parameter is allowed if the user wishes solely to connect to the PMIx server and does not require return of the identifier at that time.

Advice to PMIx library implementers

PMIx tools and clients are prohibited from being connected to more than one server at a time to avoid confusion in subsystems such as event notification.

When a tool connects to a server that is under a different namespace manager (e.g., host RM) as the prior server, the identifier of the tool must remain unique in the namespaces. This may require the identifier of the tool to be changed on-the-fly, that is, the *proc* parameter would be filled (if non-NULL) with a different nspace/rank from the current tool identifier.

Advice to users

Passing a **NULL** value for the *info* pointer is not allowed and will result in returning an error.

Some PMIx implementations (for example, the current PRI) may not support connecting to a server that is not under the same namespace manager (e.g., host RM) as the tool.

4.4 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers.

4.4.1 PMIx_server_init

Summary

Initialize the PMIx server.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_init(pmix_server_module_t *module,
                 pmix_info_t info[], size_t ninfo)
```

C

INOUT `module`
: `pmix_server_module_t` structure (handle)

IN `info`
: Array of `pmix_info_t` structures (array of handles)

IN `ninfo`
: Number of elements in the *info* array (`size_t`)

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

Required Attributes

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

`PMIX_SERVER_NSPACE` `"pmix.srv.nspace"` (`char*`)
: Name of the namespace to use for this PMIx server.

`PMIX_SERVER_RANK` `"pmix.srv.rank"` (`pmix_rank_t`)
: Rank of this PMIx server

`PMIX_SERVER_TMPDIR` `"pmix.srvr.tmpdir"` (`char*`)
: Top-level temporary directory for all client processes connected to this server, and where the PMIx server will place its tool rendezvous point and contact information.

`PMIX_SYSTEM_TMPDIR` `"pmix.sys.tmpdir"` (`char*`)
: Temporary directory for this system, and where a PMIx server that declares itself to be a system-level server will place a tool rendezvous point and contact information.

`PMIX_SERVER_TOOL_SUPPORT` `"pmix.srvr.tool"` (`bool`)
: The host RM wants to declare itself as willing to accept tool connection requests.

`PMIX_SERVER_SYSTEM_SUPPORT` `"pmix.srvr.sys"` (`bool`)
: The host RM wants to declare itself as being the local system server for PMIx connection requests.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

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

PMIX_SOCKET_MODE **"pmix.sockmode"** (**uint32_t**)
POSIX *mode_t* (9 bits valid) If the library supports socket connections, this attribute may be supported for setting the socket mode.

PMIX_TCP_REPORT_URI **"pmix.tcp.repuri"** (**char***)
If provided, directs that the TCP URI be reported and indicates the desired method of reporting: **'-'** for stdout, **'+'** for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

PMIX_TCP_IF_INCLUDE **"pmix.tcp.ifinclude"** (**char***)
Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

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

PMIX_TCP_IPV4_PORT **"pmix.tcp.ipv4"** (**int**)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT **"pmix.tcp.ipv6"** (**int**)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 **"pmix.tcp.disipv4"** (**bool**)
Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 **"pmix.tcp.disipv6"** (**bool**)
Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_SERVER_REMOTE_CONNECTIONS **"pmix.srvr.remote"** (**bool**)
Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device. If the library supports connections from remote tools, this attribute may be supported for enabling or disabling it.

PMIX_EVENT_BASE **"pmix.evbase"** (**struct event_base ***)
Pointer to libevent[3] **event_base** to use in place of the internal progress thread.

PMIX_GDS_MODULE **"pmix.gds.mod"** (**char***)
Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.

Description

Initialize the PMIx server support library, and provide a pointer to a **pmix_server_module_t** structure containing the caller's callback functions. The array of **pmix_info_t** structs is used to pass additional info that may be required by the server when initializing. For example, it may include the **PMIX_SERVER_TOOL_SUPPORT** attribute, thereby indicating that the daemon is willing to accept connection requests from tools.

Advice to PMIx server hosts

Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module* structure. Doing so indicates that the host environment will not provide support for multi-node operations such as **PMIx_Fence** , but does intend to support local clients access to information.

4.4.2 PMIx_server_finalize

Summary

Finalize the PMIx server library.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_finalize(void)
```

C

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.

[3] http://libevent.org/

CHAPTER 5

Key/Value Management

Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of the PMIx community is to eliminate collective operations, it is recognized that the traditional method of posting/exchanging data must be supported until that objective can be met. This method relies on processes to discover and post their local information which is collected by the local PMIx server library. Global exchange of the posted information is then executed via a collective operation performed by the host SMS servers. The **PMIx_Put** and **PMIx_Commit** APIs, plus an attribute directing **PMIx_Fence** to globally collect the data posted by processes, are provided for this purpose.

5.1 Setting and Accessing Key/Value Pairs

5.1.1 PMIx_Put

Summary

Push a key/value pair into the client's namespace.

Format

PMIx v1.0

C

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

C

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.

Description

Push a value into the client's namespace. The client's PMIx library will cache the information locally until **PMIx_Commit** is called.

The provided *scope* is passed to the local PMIx server, which will distribute the data to other processes according to the provided scope. The **pmix_scope_t** values are defined in Section 3.2.9 on page 34. 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 will support heterogeneous environments by properly converting binary values between host architectures, and will copy the provided *value* into internal memory.

Advice to PMIx library implementers

The PMIx server library will properly pack/unpack data to accommodate heterogeneous environments. The host SMS is not involved in this action. The *value* argument must be copied - the caller is free to release it following return from the function.

Advice to users

The value is copied by the PMIx client library. Thus, the application is free to release and/or modify the value once the call to **PMIx_Put** has completed.

Note that keys starting with a string of "**pmix**" are exclusively reserved for the PMIx standard and 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** .

5.1.2 PMIx_Get

Summary

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

Format

PMIx v1.0

C

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

C

IN **proc**
process reference (handle)

IN **key**
key to retrieve (**pmix_key_t**)

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

IN **ninfo**
Number of element in the *info* array (integer)

OUT **val**
value (handle)

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:

PMIX_OPTIONAL **"pmix.optional"** (**bool**)
Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.

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

PMIX_DATA_SCOPE **"pmix.scope"** (**pmix_scope_t**)
Scope of the data to be found in a **PMIx_Get** call.

PMIX_SESSION_INFO **"pmix.ssn.info"** (**bool**)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

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

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO **"pmix.app.info"** (**bool**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO **"pmix.node.info"** (**bool**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_GET_STATIC_VALUES **"pmix.get.static"** (**bool**)
Request that any pointers in the returned value point directly to values in the key-value store and indicate that the address provided for the return value points to a statically defined memory location. Returned non-pointer values should therefore be copied directly into the provided memory. Pointers in the returned value should point directly to values in the key-value store. User is responsible for *not* releasing memory on any returned pointer value. Note that a return status of **PMIX_ERR_GET_MALLOC_REQD** indicates that direct pointers could not be supported - thus, the returned data contains allocated memory that the user must release.

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Retrieve information for the specified *key* as published by the process identified in the given **pmix_proc_t** , returning a pointer to the value in the given address.

This is a blocking operation - the caller will block until either the specified data becomes available from the specified rank in the *proc* structure or the operation times out should the **PMIX_TIMEOUT** attribute have been given. The caller is responsible for freeing all memory associated with the returned *value* when no longer required.

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

Advice to users

Information provided by the PMIx server at time of process start is accessed by providing the namespace of the job with the rank set to **PMIX_RANK_WILDCARD** . The list of data referenced in this way is maintained on the PMIx web site at https://pmix.org/support/faq/wildcard-rank-access/ but includes items such as the number of processes in the namespace (**PMIX_JOB_SIZE**), total available slots in the allocation (**PMIX_UNIV_SIZE**), and the number of nodes in the allocation (**PMIX_NUM_NODES**).

Data posted by a process via **PMIx_Put** needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of **PMIX_RANK_WILDCARD** when the information being retrieved refers to something non-rank specific (e.g., number of processes on a node, number of processes in a job), and using the rank of the relevant process when requesting information that is rank-specific (e.g., the URI of the process, or the node upon which it is executing). Each subsection of Section 3.4 indicates the appropriate rank value for referencing the defined attribute.

5.1.3 PMIx_Get_nb

Summary

Nonblocking **PMIx_Get** operation.

Format

PMIx v1.0

C

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

C

IN **proc**
process reference (handle)

IN **key**
key to retrieve (string)

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

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

IN **cbfunc**
Callback function (function reference)

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

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

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

- **PMIX_SUCCESS** The requested data has been returned
- **PMIX_ERR_NOT_FOUND** The requested data was not available
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

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

PMIX_OPTIONAL **"pmix.optional"** (**bool**)
Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.

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

PMIX_DATA_SCOPE **"pmix.scope"** (**pmix_scope_t**)
Scope of the data to be found in a **PMIx_Get** call.

PMIX_SESSION_INFO **"pmix.ssn.info"** (**bool**)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO **"pmix.job.info"** (**bool**)
Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO **"pmix.app.info"** (**bool**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO **"pmix.node.info"** (**bool**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_GET_STATIC_VALUES **"pmix.get.static"** (**bool**)
Request that any pointers in the returned value point directly to values in the key-value store and indicate that user takes responsibility for properly releasing memory on the returned value (i.e., free'ing the value structure but not the pointer fields). Note that a return status of **PMIX_ERR_GET_MALLOC_REQD** indicates that direct pointers could not be supported - thus, the returned data contains allocated memory that the user must release.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

The callback function will be executed once the specified data becomes available from the identified process and retrieved by the local server. The *info* array is used as described by the **PMIx_Get** routine.

Advice to users

Information provided by the PMIx server at time of process start is accessed by providing the namespace of the job with the rank set to **PMIX_RANK_WILDCARD** . Attributes referenced in this way are identified in 3.4 but includes items such as the number of processes in the namespace (**PMIX_JOB_SIZE**), total available slots in the allocation (**PMIX_UNIV_SIZE**), and the number of nodes in the allocation (**PMIX_NUM_NODES**).

In general, data posted by a process via **PMIx_Put** and data that refers directly to a process-related value needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of **PMIX_RANK_WILDCARD** , as illustrated in 5.1.5. See 3.4.11 for an explanation regarding use of the *level* attributes.

5.1.4 PMIx_Store_internal

Summary

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

Format

PMIx v1.0

C

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

C

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.

Description

Store some data locally for retrieval by other areas of the proc. This is data that has only internal scope - it will never be "pushed" externally.

5.1.5 Accessing information: examples

This section provides examples illustrating methods for accessing information at various levels. The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate how **PMIx_Get** can be used to obtain information on a **session** , **job** , **application** , process, and node.

5.1.5.1 Session-level information

The **PMIx_Get** API does not include an argument for specifying the **session** associated with the information being requested. Information regarding the session containing the requestor can be obtained by the following methods:

- for session-level attributes (e.g., **PMIX_UNIV_SIZE**), specifying the requestor's namespace and a rank of **PMIX_RANK_WILDCARD** ; or
- for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_SESSION_INFO** attribute to indicate that the session-level information for that attribute is being requested

Example requests are shown below:

C

```
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, wildcard;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #slots in our session */
PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);

/* get the #nodes in our session */
PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

C

Information regarding a different session can be requested by either specifying the namespace and a rank of **PMIX_RANK_WILDCARD** for a process in the target session, or adding the **PMIX_SESSION_ID** attribute identifying the target session. In the latter case, the *proc* argument to **PMIx_Get** will be ignored:

C

```
pmix_info_t info[2];
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc;
uint32_t sid;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #nodes in a different session */
sid = 12345;
PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
rc = PMIx_Get(&myproc, PMIX_NUM_NODES, info, 2, &value);
```

C

5.1.5.2 Job-level information

Information regarding a job can be obtained by the following methods:

- for job-level attributes (e.g., **PMIX_JOB_SIZE** or **PMIX_JOB_NUM_APPS**), specifying the namespace of the job and a rank of **PMIX_RANK_WILDCARD** for the *proc* argument to **PMIx_Get** ; or
- for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_JOB_INFO** attribute to indicate that the job-level information for that attribute is being requested

Example requests are shown below:

C

```
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, wildcard;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #apps in our job */
PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);

/* get the #nodes in our job */
PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

C

5.1.5.3 Application-level information

Information regarding an application can be obtained by the following methods:

- for application-level attributes (e.g., **PMIX_APP_SIZE**), specifying the namespace and rank of a process within that application;
- for application-level attributes (e.g., **PMIX_APP_SIZE**), including the **PMIX_APPNUM** attribute specifying the application whose information is being requested. In this case, the namespace field of the *proc* argument is used to reference the **job** containing the application - the **rank** field is ignored;
- or application-level attributes (e.g., **PMIX_APP_SIZE**), including the **PMIX_APPNUM** and **PMIX_NSPACE** or **PMIX_JOBID** attributes specifying the job/application whose information is being requested. In this case, the *proc* argument is ignored;
- for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_APP_INFO** attribute to indicate that the application-level information for that attribute is being requested

Example requests are shown below:

C

```
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t appsize, appnum;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #processes in our application */
rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
appsize = value->data.uint32;

/* get the #nodes in an application containing "otherproc".
 * Note that the rank of a process in the other application
 * must be obtained first - a simple method is shown here */

/* assume for this example that we are in the first application
 * and we want the #nodes in the second application - use the
 * rank of the first process in that application, remembering
 * that ranks start at zero */
PMIX_PROC_LOAD(&otherproc, myproc.nspace, appsize);

PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);

/* alternatively, we can directly ask for the #nodes in
 * the second application in our job, again remembering that
 * application numbers start with zero */
appnum = 1;
PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
```

C

5.1.5.4 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.

5.1.5.5 Node-level information

Information regarding a node within the system can be obtained by the following methods:

- for node-level attributes (e.g., **PMIX_NODE_SIZE**), specifying the namespace and rank of a process executing on the target node;
- for node-level attributes (e.g., **PMIX_NODE_SIZE**), including the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute specifying the node whose information is being requested. In this case, the *proc* argument's values are ignored; or
- for non-specific attributes (e.g., **PMIX_MAX_PROCS**), including the **PMIX_NODE_INFO** attribute to indicate that the node-level information for that attribute is being requested

Example requests are shown below:

C

```
pmix_info_t info[2];
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t nodeid;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #procs on our node */
rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);

/* get the #slots on another node */
PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);
```

C

Advice to users

An explanation of the use of **PMIx_Get** versus **PMIx_Query_info_nb** is provided in 7.1.4.1.

5.2 Exchanging Key/Value Pairs

The APIs defined in this section push key/value pairs from the client to the local PMIx server, and circulate the data between PMIx servers for subsequent retrieval by the local clients.

5.2.1 PMIx_Commit

Summary

Push all previously **PMIx_Put** values to the local PMIx server.

Format

PMIx v1.0

C

```
pmix_status_t PMIx_Commit(void)
```

C

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

Description

This is an asynchronous operation. The PRI will immediately return to the caller while the data is transmitted to the local server in the background.

Advice to users

The local PMIx server will cache the information locally - i.e., the committed data will not be circulated during **PMIx_Commit** . Availability of the data upon completion of **PMIx_Commit** is therefore implementation-dependent.

5.2.2 PMIx_Fence

Summary

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

Format

PMIx v1.0

C

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

C

IN **procs**
Array of **pmix_proc_t** structures (array of handles)

IN **nprocs**
Number of element in the *procs* array (integer)

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

IN **ninfo**
Number of element in the *info* array (integer)

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:

PMIX_COLLECT_DATA **"pmix.collect"** (**bool**)
Collect data and return it at the end of the operation.

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

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

The *info* array is used to pass user requests regarding the fence operation.

Note that for scalability reasons, the default behavior for **PMIx_Fence** is to not collect the data.

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

5.2.3 PMIx_Fence_nb

Summary

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

Format

PMIx v1.0

C

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

C

IN **procs**
Array of **pmix_proc_t** structures (array of handles)

IN **nprocs**
Number of element in the *procs* array (integer)

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

IN **ninfo**
Number of element in the *info* array (integer)

IN **cbfunc**
Callback function (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. This can occur if the collective involved only processes on the local node.
- 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:

PMIX_COLLECT_DATA **"pmix.collect"** (**bool**)
Collect data and return it at the end of the operation.

Optional Attributes

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

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
: Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
: Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
: If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Note that PMIx libraries may choose to implement an optimization for the case where only the calling process is involved in the fence operation by immediately returning **PMIX_OPERATION_SUCCEEDED** from the client's call in lieu of passing the fence operation to a PMIx server. Fence operations involving more than just the calling process must be communicated to the PMIx server for proper execution of the included barrier behavior.

Similarly, fence operations that involve only processes that are clients of the same PMIx server may be resolved by that server without referral to its host environment as no inter-node coordination is required.

Description

Nonblocking **PMIx_Fence** routine. Note that the function will return an error if a **NULL** callback function is given.

Note that for scalability reasons, the default behavior for **PMIx_Fence_nb** is to not collect the data.

See the **PMIx_Fence** description for further details.

5.3 Publish and Lookup Data

The APIs defined in this section publish data from one client that can be later exchanged and looked up by another client.

Advice to PMIx library implementers

PMIx libraries that support any of the functions in this section are required to support *all* of them.

Advice to PMIx server hosts

Host environments that support any of the functions in this section are required to support *all* of them.

5.3.1 PMIx_Publish

Summary

Publish data for later access via **PMIx_Lookup** .

Format

PMIx v1.0

C

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

C

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

IN **ninfo**
Number of element 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, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that published 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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
: Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_PERSISTENCE **"pmix.persist"** (**pmix_persistence_t**)
: Value for calls to **PMIx_Publish** .

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

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 will be rejected. The persistence parameter instructs the server as to how long the data is to be retained.

The blocking form will block until the server confirms that the data has been sent to the PMIx server and that it has obtained confirmation from its host SMS daemon that the data is ready to be looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be released upon return from the blocking function call.

Advice to users

Publishing duplicate keys is permitted provided they are published to different ranges.

Advice to PMIx library implementers

Implementations should, to the best of their ability, detect duplicate keys being posted on the same data range and protect the user from unexpected behavior by returning the **PMIX_ERR_DUPLICATE_KEY** error.

5.3.2 PMIx_Publish_nb

Summary

Nonblocking **PMIx_Publish** routine.

Format

PMIx v1.0

C

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

C

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

IN **ninfo**
Number of element 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, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that published 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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
: Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_PERSISTENCE **"pmix.persist"** (**pmix_persistence_t**)
: Value for calls to **PMIx_Publish** .

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking **PMIx_Publish** routine. The non-blocking form will return immediately, executing the callback when the PMIx server receives confirmation from its host SMS daemon.

Note that the function will return an error if a **NULL** callback function is given, and that the *info* array must be maintained until the callback is provided.

5.3.3 PMIx_Lookup

Summary

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

Format

PMIx v1.0

C

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

C

INOUT `data`
: Array of publishable data structures (array of handles)

IN `ndata`
: Number of elements in the *data* 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, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is requesting the info.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (**int**)
: Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE "pmix.range" (**pmix_data_range_t**)
: Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_WAIT "pmix.wait" (**int**)
: Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Lookup information published by this or another process. By default, the search will be conducted across the **PMIX_RANGE_SESSION** range. Changes to the range, and any additional directives, can be provided in the **pmix_info_t** array. Data is returned provided the following conditions are met:

- the requesting process resides within the range specified by the publisher. For example, data published to **PMIX_RANGE_LOCAL** can only be discovered by a process executing on the same node
- the provided key matches the published key within that data range
- the data was published by a process with corresponding user and/or group IDs as the one looking up the data. There currently is no option to override this behavior - such an option may become available later via an appropriate **pmix_info_t** directive.

The *data* parameter consists of an array of **pmix_pdata_t** struct with the keys specifying the requested information. Data will be returned for each key in the associated *value* struct. Any key that cannot be found will return with a data type of **PMIX_UNDEF** . The function will return **PMIX_SUCCESS** if any values can be found, so the caller must check each data element to ensure it was returned.

The proc field in each **pmix_pdata_t** struct will contain the namespace/rank of the process that published the data.

Advice to users

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

5.3.4 PMIx_Lookup_nb

Summary

Nonblocking version of **PMIx_Lookup** .

Format

PMIx v1.0

C

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

C

IN **keys**
Array to be provided to the callback (array of strings)

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

IN **ninfo**
Number of element in the *info* array (integer)

IN **cbfunc**
Callback function (handle)

IN **cbdata**
Callback data to be provided to the callback function (pointer)

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

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 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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_WAIT **"pmix.wait"** (**int**)
Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking form of the **PMIx_Lookup** function. Data for the provided NULL-terminated *keys* array will be returned in the provided callback function. As with **PMIx_Lookup**, the default behavior is to not wait for data to be published. The *info* array can be used to modify the behavior as previously described by **PMIx_Lookup**. Both the *info* and *keys* arrays must be maintained until the callback is provided.

5.3.5 PMIx_Unpublish

Summary

Unpublish data posted by this process using the given keys.

Format

PMIx v1.0

C

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

C

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

IN **ninfo**
Number of element 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, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is requesting the operation.

Optional Attributes

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

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

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

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

5.3.6 PMIx_Unpublish_nb

Summary

Nonblocking version of **PMIx_Unpublish** .

Format

PMIx v1.0

C

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

C

IN **keys**
(array of strings)

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

IN **ninfo**
Number of element 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, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is requesting the operation.

Optional Attributes

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

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
: Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
: Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

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

CHAPTER 6

Process Management

This chapter defines functionality used by clients to create and destroy/abort processes in the PMIx universe.

6.1 Abort

PMIx provides a dedicated API by which an application can request that specified processes be aborted by the system.

6.1.1 PMIx_Abort

Summary

Abort the specified processes

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Abort(int status, const char msg[],
           pmix_proc_t procs[], size_t nprocs)
```

C

IN `status`
Error code to return to invoking environment (integer)

IN `msg`
String message to be returned to user (string)

IN `procs`
Array of `pmix_proc_t` structures (array of handles)

IN `nprocs`
Number of elements in the *procs* array (integer)

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

Description

Request that the host resource manager print the provided message and abort the provided array of *procs*. A Unix or POSIX environment should handle the provided status as a return error code from the main program that launched the application. A **NULL** for the *procs* array indicates that all processes in the caller's namespace are to be aborted, including itself. Passing a **NULL** *msg* parameter is allowed.

Advice to users

The response to this request is somewhat dependent on the specific resource manager and its configuration (e.g., some resource managers will not abort the application if the provided status is zero unless specifically configured to do so, and some cannot abort subsets of processes in an application), and thus lies outside the control of PMIx itself. However, the PMIx client library shall inform the RM of the request that the specified *procs* be aborted, regardless of the value of the provided status.

Note that race conditions caused by multiple processes calling **PMIx_Abort** are left to the server implementation to resolve with regard to which status is returned and what messages (if any) are printed.

6.2 Process Creation

The **PMIx_Spawn** commands spawn new processes and/or applications in the PMIx universe. This may include requests to extend the existing resource allocation or obtain a new one, depending upon provided and supported attributes.

6.2.1 PMIx_Spawn

Summary

Spawn a new job.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
           const pmix_app_t apps[], size_t napps,
           char nspace[])
```

C

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 `napps`
Number of elements in the *apps* array (integer)

OUT `nspace`
Namespace of the new job (string)

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, and the PMIx library is required to add the following attributes to those provided before passing the request to the host:

`PMIX_SPAWNED` `"pmix.spawned"` (`bool`)
`true` if this process resulted from a call to `PMIx_Spawn` .

`PMIX_PARENT_ID` `"pmix.parent"` (`pmix_proc_t`)
Process identifier of the parent process of the calling process.

`PMIX_REQUESTOR_IS_CLIENT` `"pmix.req.client"` (`bool`)
The requesting process is a PMIx client.

`PMIX_REQUESTOR_IS_TOOL` `"pmix.req.tool"` (`bool`)
The requesting process is a PMIx tool.

Host environments that implement support for `PMIx_Spawn` are required to pass the `PMIX_SPAWNED` and `PMIX_PARENT_ID` attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

`PMIX_WDIR` `"pmix.wdir"` (`char*`)

Working directory for spawned processes.

PMIX_SET_SESSION_CWD **"pmix.ssncwd"** (**bool**)
Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX **"pmix.prefix"** (**char***)
Prefix to use for starting spawned processes.

PMIX_HOST **"pmix.host"** (**char***)
Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE **"pmix.hostfile"** (**char***)
Hostfile to use for spawned processes.

Optional Attributes

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

PMIX_ADD_HOSTFILE **"pmix.addhostfile"** (**char***)
Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST **"pmix.addhost"** (**char***)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN **"pmix.preloadbin"** (**bool**)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES **"pmix.preloadfiles"** (**char***)
Comma-delimited list of files to pre-position on nodes.

PMIX_PERSONALITY **"pmix.pers"** (**char***)
Name of personality to use.

PMIX_MAPPER **"pmix.mapper"** (**char***)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP **"pmix.dispmap"** (**bool**)
Display process mapping upon spawn.

PMIX_PPR **"pmix.ppr"** (**char***)
Number of processes to spawn on each identified resource.

PMIX_MAPBY **"pmix.mapby"** (**char***)

Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY **"pmix.rankby"** (**char***)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO **"pmix.bindto"** (**char***)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI **"pmix.nonpmi"** (**bool**)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT **"pmix.stdin"** (**uint32_t**)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS **"pmix.debugger"** (**bool**)
Spawned application consists of debugger daemons.

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)
Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)

Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)
Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS **"pmix.maxrestarts"** (**uint32_t**)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

PMIX_NOTIFY_COMPLETION **"pmix.notecomp"** (**bool**)
Notify the parent process upon termination of child job.

Description

Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspace* parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the namespace returned. The *nspace* array must be at least of size one more than **PMIX_MAX_NSLEN** .

By default, the spawned processes will be PMIx "connected" to the parent process upon successful launch (see **PMIx_Connect** description for details). Note that this only means that (a) the parent process will be given a copy of the new job's information so it can query job-level info without incurring any communication penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level info, and (c) both the parent process and members of the child job will receive notification of errors from processes in their combined assemblage.

Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

6.2.2 PMIx_Spawn_nb

Summary

Nonblocking version of the **PMIx_Spawn** routine.

Format

PMIx v1.0

C

```
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)
```

C

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

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 following attributes to those provided before passing the request to the host:

PMIX_SPAWNED **"pmix.spawned"** (**bool**)
: **true** if this process resulted from a call to **PMIx_Spawn** .

PMIX_PARENT_ID **"pmix.parent"** (**pmix_proc_t**)
: Process identifier of the parent process of the calling process.

PMIX_REQUESTOR_IS_CLIENT **"pmix.req.client"** (**bool**)
: The requesting process is a PMIx client.

PMIX_REQUESTOR_IS_TOOL **"pmix.req.tool"** (**bool**)
: The requesting process is a PMIx tool.

Host environments that implement support for **PMIx_Spawn** are required to pass the **PMIX_SPAWNED** and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX_WDIR **"pmix.wdir"** (**char***)
: Working directory for spawned processes.

PMIX_SET_SESSION_CWD **"pmix.ssncwd"** (**bool**)
: Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX **"pmix.prefix"** (**char***)
: Prefix to use for starting spawned processes.

PMIX_HOST **"pmix.host"** (**char***)
: Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE **"pmix.hostfile"** (**char***)
: Hostfile to use for spawned processes.

Optional Attributes

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

PMIX_ADD_HOSTFILE **"pmix.addhostfile"** (**char***)
Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST **"pmix.addhost"** (**char***)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN **"pmix.preloadbin"** (**bool**)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES **"pmix.preloadfiles"** (**char***)
Comma-delimited list of files to pre-position on nodes.

PMIX_PERSONALITY **"pmix.pers"** (**char***)
Name of personality to use.

PMIX_MAPPER **"pmix.mapper"** (**char***)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP **"pmix.dispmap"** (**bool**)
Display process mapping upon spawn.

PMIX_PPR **"pmix.ppr"** (**char***)
Number of processes to spawn on each identified resource.

PMIX_MAPBY **"pmix.mapby"** (**char***)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY **"pmix.rankby"** (**char***)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO **"pmix.bindto"** (**char***)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI **"pmix.nonpmi"** (**bool**)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT **"pmix.stdin"** (**uint32_t**)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS **"pmix.debugger"** (**bool**)
Spawned application consists of debugger daemons.

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)
Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)
Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

Description

Nonblocking version of the **PMIx_Spawn** routine. The provided callback function will be executed upon successful start of *all* specified application processes.

Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

6.3 Connecting and Disconnecting Processes

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of *connected* solely implies that the host environment should treat the failure of any process in the assemblage as a reportable event, taking action on the assemblage as if it were a single application. For example, if the environment defaults (in the absence of any application directives) to terminating an application upon failure of any process in that application, then the environment should terminate all processes in the connected assemblage upon failure of any member.

Advice to PMIx server hosts

The host environment may choose to assign a new namespace to the connected assemblage and/or assign new ranks for its members for its own internal tracking purposes. However, it is not required to communicate such assignments to the participants (e.g., in response to an appropriate call to **PMIx_Query_info_nb**). The host environment is required to generate a **PMIX_ERR_INVALID_TERMINATION** event should any process in the assemblage terminate or call **PMIx_Finalize** without first *disconnecting* from the assemblage.

The *connect* operation does not require the exchange of job-level information nor the inclusion of information posted by participating processes via **PMIx_Put** . Indeed, the callback function utilized in **pmix_server_connect_fn_t** cannot pass information back into the PMIx server library. However, host environments are advised that collecting such information at the participating daemons represents an optimization opportunity as participating processes are likely to request such information after the connect operation completes.

Advice to users

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation. While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

Neither the PMIx implementation nor host environment are required to provide any tracking support for the assemblage. Thus, the application is responsible for maintaining the membership list of the assemblage.

6.3.1 PMIx_Connect

Summary

Connect namespaces.

Format

PMIx v1.0

C

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

C

IN **procs**
Array of proc structures (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

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

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

Returns **PMIX_SUCCESS** or a negative value 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

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
: Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
: If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The function will return once all processes identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

Advice to users

All processes engaged in a given **PMIx_Connect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

Processes that combine via **PMIx_Connect** must call **PMIx_Disconnect** prior to finalizing and/or terminating - any process in the assemblage failing to meet this requirement will cause a **PMIX_ERR_INVALID_TERMINATION** event to be generated.

A process can only engage in one connect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple connect operations, each involving a different *procs* array.

As in the case of the **PMIx_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

6.3.2 PMIx_Connect_nb

Summary

Nonblocking **PMIx_Connect_nb** routine.

Format

PMIx v1.0

C

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

C

IN **procs**
Array of proc structures (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

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

IN **ninfo**
Number of element 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

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking version of **PMIx_Connect** . The callback function is called once all processes identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes. See the advice provided in the description for **PMIx_Connect** for more information.

6.3.3 PMIx_Disconnect

Summary

Disconnect a previously connected set of processes.

Format

PMIx v1.0

C

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

C

IN **procs**
Array of proc structures (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

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

IN **ninfo**
Number of element 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

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Disconnect a previously connected set of processes. A **PMIX_ERR_INVALID_OPERATION** error will be returned if the specified set of *procs* was not previously *connected* via a call to **PMIx_Connect** or its non-blocking form. 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.

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.

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.

6.3.4 PMIx_Disconnect_nb

Summary

Nonblocking **PMIx_Disconnect** routine.

Format

PMIx v1.0

C

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

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 element 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

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Nonblocking **PMIx_Disconnect** routine. The callback function is called once all processes identified in *procs* have called either **PMIx_Disconnect_nb** or its blocking version, *and* the host environment has completed any required supporting operations. See the advice provided in the description for **PMIx_Disconnect** for more information.

6.4 IO Forwarding

This section defines functions by which tools (e.g., debuggers) can request forwarding of input/output to/from other processes. The term "tool" widely refers to non-computational programs executed by the user or system administrator to monitor or control a principal computational program. Tools almost always interact with either the host environment, 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.

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 for display on the user's console. Historically, each tool developer was responsible for creating their own IO forwarding subsystem. However, with the introduction of PMIx as a standard mechanism for interacting between applications and the host environment, it has become possible to relieve tool developers of this burden.

Advice to PMIx server hosts

The responsibility of the host environment in forwarding of IO falls into the following areas:

- Capturing output from specified child processes
- Forwarding that output to the host of the PMIx server library that requested it
- Delivering that payload to the PMIx server library via the **PMIx_server_IOF_deliver** API for final dispatch

It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the requesting client.

Advice to users

The forwarding of IO via PMIx requires that both the host environment and the tool support PMIx, but does not impose any similar requirements on the application itself.

6.4.1 PMIx_IOF_pull

Summary

Register to receive output forwarded from a set of remote processes.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,
              const pmix_info_t directives[], size_t ndirs,
              pmix_iof_channel_t channel, pmix_iof_cbfunc_t cbfunc,
              pmix_hdlr_reg_cbfunc_t regcbfunc, void *regcbdata)
```

C

IN **procs**
Array of proc structures identifying desired source processes (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (integer)

IN **channel**
Bitmask of IO channels included in the request (**pmix_iof_channel_t**)

IN **cbfunc**
Callback function for delivering relevant output (**pmix_iof_cbfunc_t** function reference)

IN **regcbfunc**
Function to be called when registration is completed (**pmix_hdlr_reg_cbfunc_t** function reference)

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

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

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

PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API. The IOF handler identifier will be returned in the callback

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.

Required Attributes

The following attributes are required for PMIx libraries that support IO forwarding:

PMIX_IOF_CACHE_SIZE **"pmix.iof.csize"** (**uint32_t**)
The requested size of the 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, drop the oldest bytes to make room in the cache.

PMIX_IOF_DROP_NEWEST **"pmix.iof.new"** (**bool**)
In an overflow situation, 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**)
Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon 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**)
Tag output with the channel it comes from.

PMIX_IOF_TIMESTAMP_OUTPUT **"pmix.iof.ts"** (**bool**)

Timestamp output

PMIX_IOF_XML_OUTPUT **"pmix.iof.xml"** (**bool**)
Format output in XML

Description

Register to receive output forwarded from a set of remote processes.

Advice to users

Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated channels to be written to their corresponding stdout/stderr file descriptors. Use of **PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but should be used carefully due to bandwidth considerations.

6.4.2 PMIx_IOF_deregister

Summary

Deregister from output forwarded from a set of remote processes.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_IOF_deregister(size_t iofhdlr,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **iofhdlr**
Registration number returned from the **pmix_hdlr_reg_cbfunc_t** callback from the call to **PMIx_IOF_pull** (**size_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (integer)

IN **cbfunc**
Callback function to be called when deregistration has been completed. (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 IOF handler was successfully deregistered.
PMIX_ERR_BAD_PARAM The provided *iofhdlr* was unrecognized.
PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.

Description

Deregister from output forwarded from a set of remote processes.

Advice to PMIx library implementers

Any currently buffered IO should be flushed upon receipt of a deregistration request. All received IO after receipt of the request shall be discarded.

6.4.3 PMIx_IOF_push

Summary

Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,
              pmix_byte_object_t *bo,
              const pmix_info_t directives[], size_t ndirs,
              pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **targets**
Array of proc structures identifying desired target processes (array of handles)

IN **ntargets**
Number of elements in the *targets* array (integer)

IN **bo**
Pointer to **pmix_byte_object_t** containing the payload to be delivered (handle)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (integer)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **cbfunc**
Callback function to be called when operation has been completed. (**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 provided data has been accepted for transmission - it is not indicative of the payload being delivered to any member of the provided *targets*

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
a PMIx error constant indicating the nature of the error

Required Attributes

The following attributes are required for PMIx libraries that support IO forwarding:

PMIX_IOF_CACHE_SIZE **"pmix.iof.csize"** (**uint32_t**)
The requested size of the 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, drop the oldest bytes to make room in the cache.

PMIX_IOF_DROP_NEWEST **"pmix.iof.new"** (**bool**)
In an overflow situation, 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**)
Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon 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.

Description

Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

Advice to users

Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets. Use of **PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but should be used carefully due to bandwidth considerations.

CHAPTER 7

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_nb** , **PMIx_Job_control_nb** , and **PMIx_Process_monitor_nb** APIs.

7.1 Query

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

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

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

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

7.1.1 PMIx_Resolve_peers

Summary

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

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Resolve_peers(const char *nodename,
                   const pmix_nspace_t nspace,
                   pmix_proc_t **procs, size_t *nprocs)
```

C

IN **nodename**
Name of the node to query (string)

IN **nspace**
namespace (string)

OUT **procs**
Array of process structures (array of handles)

OUT **nprocs**
Number of elements in the *procs* array (integer)

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

Description

Given a *nodename*, return the array of processes within the specified *nspace* that are executing on that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will be **0**. The caller is responsible for releasing the *procs* array when done with it. The **PMIX_PROC_FREE** macro is provided for this purpose.

7.1.2 PMIx_Resolve_nodes

Summary

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

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Resolve_nodes(const char *nspace, char **nodelist)
```

C

IN **nspace**
Namespace (string)

OUT **nodelist**
Comma-delimited list of nodenames (string)

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

Description

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

7.1.3 PMIx_Query_info

Summary

Query information about the system in general.

Format

PMIx v4.0

C

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

C

IN `queries`
Array of query structures (array of handles)

IN `nqueries`
Number of elements in the *queries* array (integer)

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

INOUT `ninfo`
Address where the number of elements in *info* can be returned (handle)

Returns one of the following:

- `PMIX_SUCCESS` All data has been returned
- `PMIX_ERR_NOT_FOUND` None of the requested data was available
- `PMIX_ERR_PARTIAL_SUCCESS` Some of the data has been returned
- `PMIX_ERR_NOT_SUPPORTED` The host RM does not support this function
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

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

`PMIX_QUERY_REFRESH_CACHE` `"pmix.qry.rfsh"` (`bool`)
Retrieve updated information from server.

`PMIX_SESSION_INFO` `"pmix.ssn.info"` (`bool`)

Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO **"pmix.job.info"** (**bool**)
Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO **"pmix.app.info"** (**bool**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO **"pmix.node.info"** (**bool**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_PROCID **"pmix.procid"** (**pmix_proc_t**)
Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

PMIX_QUERY_ATTRIBUTE_SUPPORT **"pmix.qry.attrs"** (**bool**)
Query list of supported attributes for specified APIs

PMIX_CLIENT_ATTRIBUTES **"pmix.client.attrs"** (**bool**)
Request attributes supported by the PMIx client library

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

Request attributes supported by the PMIx server library

PMIX_HOST_ATTRIBUTES **"pmix.host.attrs"** (**bool**)
Request attributes supported by the host environment

PMIX_TOOL_ATTRIBUTES **"pmix.setup.env"** (**bool**)
Request attributes supported by the PMIx tool library functions

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

PMIx libraries are not required to directly support any other 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 support this operation are required to support the following attributes as qualifiers to the request:

PMIX_PROCID **"pmix.procid"** (**pmix_proc_t**)
Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

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

Optional Attributes

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

PMIX_QUERY_NAMESPACES **"pmix.qry.ns"** (**char***)
Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS **"pmix.qry.jst"** (**pmix_status_t**)
Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST **"pmix.qry.qlst"** (**char***)
Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS **"pmix.qry.qst"** (**TBD**)
Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE **"pmix.qry.ptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE **"pmix.qry.lptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_SPAWN_SUPPORT **"pmix.qry.spawn"** (**bool**)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT **"pmix.qry.debug"** (**bool**)
Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE **"pmix.qry.mem"** (**bool**)
Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_REPORT_AVG **"pmix.qry.avg"** (**bool**)
Report only average values for sampled information.

PMIX_QUERY_REPORT_MINMAX **"pmix.qry.minmax"** (**bool**)
Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS **"pmix.query.alloc"** (**char***)
String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING **"pmix.time.remaining"** (**char***)
Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

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

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

Description

Query information about the system in general. This can include a list of active namespaces, network topology, etc. Also can be used to query node-specific info such as the list of peers executing on a given node. We assume that the host RM will exercise appropriate access control on the information.

The returned *status* indicates if requested data was found or not. The returned array of **pmix_info_t** will contain each key that was provided and the corresponding value that was found. Requests for keys that are not found will return the key paired with a value of type **PMIX_UNDEF** . 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.

7.1.4 PMIx_Query_info_nb

Summary

Query information about the system in general.

Format

PMIx v2.0

C

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

C

IN **queries**
Array of query structures (array of handles)

IN **nqueries**
Number of elements in the *queries* array (integer)

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

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 has been returned
- **PMIX_ERR_NOT_FOUND** None of the requested data was available
- **PMIX_ERR_PARTIAL_SUCCESS** Some of the data has been returned
- **PMIX_ERR_NOT_SUPPORTED** The host RM does not support this function
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

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

PMIX_QUERY_REFRESH_CACHE **"pmix.qry.rfsh"** (**bool**)
Retrieve updated information from server.

PMIX_SESSION_INFO **"pmix.ssn.info"** (**bool**)
Return information about the specified session. If information about a session other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_SESSION_ID** attribute identifying the desired target.

PMIX_JOB_INFO **"pmix.job.info"** (**bool**)
Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_JOBID** or **PMIX_NSPACE** attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

PMIX_APP_INFO **"pmix.app.info"** (**bool**)
Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

PMIX_NODE_INFO **"pmix.node.info"** (**bool**)
Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute identifying the desired target.

PMIX_PROCID **"pmix.procid"** (**pmix_proc_t**)
Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char*****)**
Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

PMIX_QUERY_ATTRIBUTE_SUPPORT **"pmix.qry.attrs"** (**bool**)
Query list of supported attributes for specified APIs

PMIX_CLIENT_ATTRIBUTES **"pmix.client.attrs"** (**bool**)
Request attributes supported by the PMIx client library

PMIX_SERVER_ATTRIBUTES **"pmix.srvr.attrs"** (**bool**)
Request attributes supported by the PMIx server library

PMIX_HOST_ATTRIBUTES **"pmix.host.attrs"** (**bool**)
Request attributes supported by the host environment

PMIX_TOOL_ATTRIBUTES **"pmix.setup.env"** (**bool**)
Request attributes supported by the PMIx tool library functions

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

PMIx libraries are not required to directly support any other 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 support this operation are required to support the following attributes as qualifiers to the request:

PMIX_PROCID **"pmix.procid"** (**pmix_proc_t**)
Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the request is for information on a specific process.

PMIX_NSPACE **"pmix.nspace"** (**char***)
Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_RANK** attribute. Only required when the request is for information on a specific process.

PMIX_RANK **"pmix.rank"** (**pmix_rank_t**)
Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must be accompanied by the **PMIX_NSPACE** attribute. Only required when the request is for information on a specific process.

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

Optional Attributes

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

PMIX_QUERY_NAMESPACES **"pmix.qry.ns"** (**char***)
Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS **"pmix.qry.jst"** (**pmix_status_t**)
Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST **"pmix.qry.qlst"** (**char***)
Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS **"pmix.qry.qst"** (**TBD**)
Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE **"pmix.qry.ptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE **"pmix.qry.lptable"** (**char***)
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_SPAWN_SUPPORT **"pmix.qry.spawn"** (**bool**)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT **"pmix.qry.debug"** (**bool**)
Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE **"pmix.qry.mem"** (**bool**)
Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_REPORT_AVG **"pmix.qry.avg"** (**bool**)
Report only average values for sampled information.

PMIX_QUERY_REPORT_MINMAX **"pmix.qry.minmax"** (**bool**)
Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS **"pmix.query.alloc"** (**char***)
String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING **"pmix.time.remaining"** (**char***)
Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

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

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

Description

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

7.1.4.1 Using PMIx_Get vs PMIx_Query_info

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

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

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

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

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

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

7.1.4.2 Accessing attribute support information

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

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

The desired levels (see 3.4.33) of attribute support are provided as qualifiers. Multiple levels can be requested simultaneously by simply adding elements to the *qualifiers* array. Each qualifier should contain the desired level attribute with the boolean value set to indicate whether or not that level is to be included in the returned information. Failure to provide any levels is equivalent to a request for all levels.

Unlike other queries, queries for attribute support can result in the number of returned **`pmix_info_t`** structures being different from the number of queries. Each element in the returned array will correspond to a pair of specified attribute level and function in the query, where the *key* is the function and the *value* contains a **`pmix_data_array_t`** of **`pmix_info_t`** . Each element of the array is marked by a *key* indicating the requested attribute *level* with a *value* composed of a **`pmix_data_array_t`** of **`pmix_regattr_t`** , each describing a supported attribute for that function, as illustrated in Fig. 7.1 below where the requestor asked for supported attributes of **`PMIx_Get`** at the *client* and *server* levels, plus attributes of **`PMIx_Allocation_request`** at all levels:

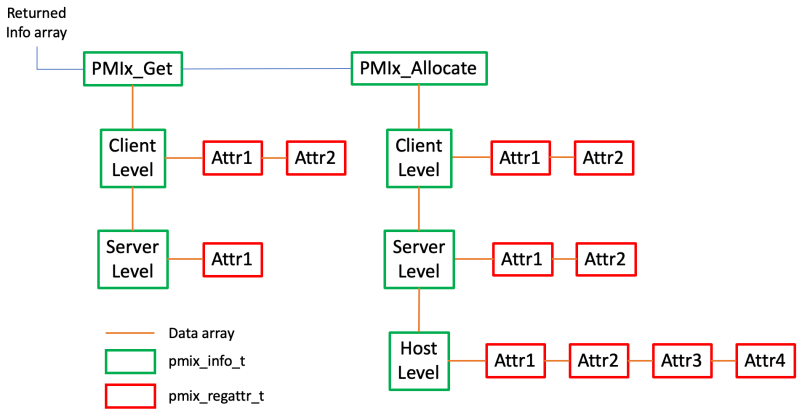

Figure 7.1.: Returned information hierarchy for attribute support request

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

7.2 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.

7.2.1 PMIx_Allocation_request

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_Allocation_request(pmix_alloc_directive_t directive,
                        pmix_info_t info[], size_t ninfo,
                        pmix_info_t *results[], size_t *nresults);
```

C

IN `directive`
Allocation directive (handle)

IN `info`
Array of `pmix_info_t` structures (array of handles)

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

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*
- a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

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.

PMIX_ALLOC_NUM_CPUS **"pmix.alloc.ncpus"** (**uint64_t**)
: Number of cpus.

PMIX_ALLOC_TIME **"pmix.alloc.time"** (**uint32_t**)
: Time in seconds.

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.

PMIX_ALLOC_NUM_CPU_LIST **"pmix.alloc.ncpulist"** (**char***)
: Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST **"pmix.alloc.cpulist"** (**char***)
: Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE **"pmix.alloc.msize"** (**float**)
: Number of Megabytes.

PMIX_ALLOC_NETWORK **"pmix.alloc.net"** (**array**)
: Array of **pmix_info_t** describing requested network resources. This must include at least: **PMIX_ALLOC_NETWORK_ID** , **PMIX_ALLOC_NETWORK_TYPE** , and **PMIX_ALLOC_NETWORK_ENDPTS** , plus whatever other descriptors are desired.

PMIX_ALLOC_NETWORK_ID **"pmix.alloc.netid"** (**char***)

The key to be used when accessing this requested network allocation. The allocation will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the network support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: **PMIX_ALLOC_NETWORK_TYPE** - the type of resources provided; **PMIX_ALLOC_NETWORK_PLANE** - if applicable, what plane the resources were assigned from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the requested network allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw"** (**float**)
Mbits/sec.

PMIX_ALLOC_NETWORK_QOS **"pmix.alloc.netqos"** (**char***)
Quality of service level.

PMIX_ALLOC_NETWORK_TYPE **"pmix.alloc.nettype"** (**char***)
Type of desired transport (e.g., *"tcp"*, *"udp"*)

PMIX_ALLOC_NETWORK_PLANE **"pmix.alloc.netplane"** (**char***)
ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

PMIX_ALLOC_NETWORK_ENDPTS **"pmix.alloc.endpts"** (**size_t**)
Number of endpoints to allocate per process

PMIX_ALLOC_NETWORK_ENDPTS_NODE **"pmix.alloc.endpts.nd"** (**size_t**)
Number of endpoints to allocate per node

PMIX_ALLOC_NETWORK_SEC_KEY **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
Network security key

Description

Request an allocation operation from the host resource manager. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute. This should be accomplished in a non-blocking manner so that the application can continue to progress while waiting for resources to become available. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.

- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities. This includes extending the time limit on current resources, and/or requesting additional resources be allocated to the requesting job. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Return no-longer-required resources to the scheduler. This includes the "loan" of resources back to the scheduler with a promise to return them upon subsequent request.

If successful, the returned results for a request for additional resources must include the host resource manager's identifier (**PMIX_ALLOC_ID**) that the requester can use to specify the resources in, for example, a call to **PMIx_Spawn** .

7.2.2 PMIx_Allocation_request_nb

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
                           pmix_info_t info[], size_t ninfo,
                           pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

IN **directive**
Allocation directive (handle)

IN **info**
Array of **pmix_info_t** structures (array of handles)

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

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Returns one of the following:

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

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

PMIX_ALLOC_REQ_ID **"pmix.alloc.reqid"** (**char***)
: User-provided string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_NUM_NODES **"pmix.alloc.nnodes"** (**uint64_t**)
: The number of nodes.

PMIX_ALLOC_NUM_CPUS **"pmix.alloc.ncpus"** (**uint64_t**)
: Number of cpus.

PMIX_ALLOC_TIME **"pmix.alloc.time"** (**uint32_t**)
: Time in seconds.

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.

PMIX_ALLOC_NUM_CPU_LIST **"pmix.alloc.ncpulist"** (**char***)
: Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST **"pmix.alloc.cpulist"** (**char***)
: Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE **"pmix.alloc.msize"** (**float**)
: Number of Megabytes.

PMIX_ALLOC_NETWORK **"pmix.alloc.net"** (**array**)
: Array of **pmix_info_t** describing requested network resources. This must include at least: **PMIX_ALLOC_NETWORK_ID** , **PMIX_ALLOC_NETWORK_TYPE** , and **PMIX_ALLOC_NETWORK_ENDPTS** , plus whatever other descriptors are desired.

PMIX_ALLOC_NETWORK_ID **"pmix.alloc.netid"** (**char***)

The key to be used when accessing this requested network allocation. The allocation will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the network support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: **PMIX_ALLOC_NETWORK_TYPE** - the type of resources provided; **PMIX_ALLOC_NETWORK_PLANE** - if applicable, what plane the resources were assigned from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the requested network allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw"** (**float**)
Mbits/sec.

PMIX_ALLOC_NETWORK_QOS **"pmix.alloc.netqos"** (**char*****)**
Quality of service level.

PMIX_ALLOC_NETWORK_TYPE **"pmix.alloc.nettype"** (**char*****)**
Type of desired transport (e.g., *"tcp"*, *"udp"*)

PMIX_ALLOC_NETWORK_PLANE **"pmix.alloc.netplane"** (**char*****)**
ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

PMIX_ALLOC_NETWORK_ENDPTS **"pmix.alloc.endpts"** (**size_t**)
Number of endpoints to allocate per process

PMIX_ALLOC_NETWORK_ENDPTS_NODE **"pmix.alloc.endpts.nd"** (**size_t**)
Number of endpoints to allocate per node

PMIX_ALLOC_NETWORK_SEC_KEY **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
Network security key

Description

Non-blocking form of the **PMIx_Allocation_request** API.

7.3 Job Control

This section defines APIs that enable the application and host environment to coordinate the response to failures and other events. This can include requesting termination of the entire job or a subset of processes within a job, but can also be used in combination with other PMIx capabilities (e.g., allocation support and event notification) for more nuanced responses. For example, an application notified of an incipient over-temperature condition on a node could use the **PMIx_Allocation_request_nb** interface to request replacement nodes while simultaneously using the **PMIx_Job_control_nb** interface to direct that a checkpoint event be delivered to all processes in the application. If replacement resources are not available, the application might use the **PMIx_Job_control_nb** interface to request that the job continue at a lower power setting, perhaps sufficient to avoid the over-temperature failure.

The job control APIs can also be used by an application to register itself as available for preemption when operating in an environment such as a cloud or where incentives, financial or otherwise, are provided to jobs willing to be preempted. Registration can include attributes indicating how many resources are being offered for preemption (e.g., all or only some portion), whether the application will require time to prepare for preemption, etc. Jobs that request a warning will receive an event notifying them of an impending preemption (possibly including information as to the resources that will be taken away, how much time the application will be given prior to being preempted, whether the preemption will be a suspension or full termination, etc.) so they have an opportunity to save their work. Once the application is ready, it calls the provided event completion callback function to indicate that the SMS is free to suspend or terminate it, and can include directives regarding any desired restart.

7.3.1 PMIx_Job_control

Summary

Request a job control action.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
                 const pmix_info_t directives[], size_t ndirs,
                 pmix_info_t *results[], size_t *nresults)
```

C

IN **targets**
Array of proc structures (array of handles)

IN **ntargets**
Number of element in the *targets* array (integer)

IN **directives**
Array of info structures (array of handles)

IN `ndirs`
Number of element in the *directives* array (integer)

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 by the host environment and returned *success*. Details of the result will be returned in the *results* array
- a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

`PMIX_JOB_CTRL_ID` `"pmix.jctrl.id"` (`char*`)
Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

`PMIX_JOB_CTRL_PAUSE` `"pmix.jctrl.pause"` (`bool`)
Pause the specified processes.

`PMIX_JOB_CTRL_RESUME` `"pmix.jctrl.resume"` (`bool`)
Resume ("un-pause") the specified processes.

`PMIX_JOB_CTRL_KILL` `"pmix.jctrl.kill"` (`bool`)
Forcibly terminate the specified processes and cleanup.

`PMIX_JOB_CTRL_SIGNAL` `"pmix.jctrl.sig"` (`int`)
Send given signal to specified processes.

`PMIX_JOB_CTRL_TERMINATE` `"pmix.jctrl.term"` (`bool`)
Politely terminate the specified processes.

`PMIX_REGISTER_CLEANUP` `"pmix.reg.cleanup"` (`char*`)
Comma-delimited list of files to be removed upon process termination

`PMIX_REGISTER_CLEANUP_DIR` `"pmix.reg.cleanupdir"` (`char*`)

Comma-delimited list of directories to be removed upon process termination

PMIX_CLEANUP_RECURSIVE **"pmix.clnup.recurse"** (**bool**)
Recursively cleanup all subdirectories under the specified one(s)

PMIX_CLEANUP_EMPTY **"pmix.clnup.empty"** (**bool**)
Only remove empty subdirectories

PMIX_CLEANUP_IGNORE **"pmix.clnup.ignore"** (**char****)
Comma-delimited list of filenames that are not to be removed

PMIX_CLEANUP_LEAVE_TOPDIR **"pmix.clnup.lvtop"** (**bool**)
When recursively cleaning subdirectories, do not remove the top-level directory (the one given in the cleanup request)

Optional Attributes

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

PMIX_JOB_CTRL_CANCEL **"pmix.jctrl.cancel"** (**char****)
Cancel the specified request - the provided request ID must match the **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of **NULL** implies cancel all requests from this requestor.

PMIX_JOB_CTRL_RESTART **"pmix.jctrl.restart"** (**char****)
Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT **"pmix.jctrl.ckpt"** (**char****)
Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT **"pmix.jctrl.ckptev"** (**bool**)
Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL **"pmix.jctrl.ckptsig"** (**int**)
Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT **"pmix.jctrl.ckptsig"** (**int**)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.

PMIX_JOB_CTRL_PROVISION **"pmix.jctrl.pvn"** (**char****)
Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE **"pmix.jctrl.pvnimg"** (**char****)
Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE **"pmix.jctrl.preempt"** (**bool**)

Indicate that the job can be pre-empted.

Description

Request a job control action. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX_RANK_WILDCARD** can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix_info_t** structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

7.3.2 PMIx_Job_control_nb

Summary

Request a job control action.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **targets**
Array of proc structures (array of handles)

IN **ntargets**
Number of element in the *targets* array (integer)

IN **directives**
Array of info structures (array of handles)

IN **ndirs**
Number of element in the *directives* array (integer)

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

PMIX_JOB_CTRL_ID **"pmix.jctrl.id"** (**char***)
: Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

PMIX_JOB_CTRL_PAUSE **"pmix.jctrl.pause"** (**bool**)
: Pause the specified processes.

PMIX_JOB_CTRL_RESUME **"pmix.jctrl.resume"** (**bool**)
: Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_KILL **"pmix.jctrl.kill"** (**bool**)
: Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_SIGNAL **"pmix.jctrl.sig"** (**int**)
: Send given signal to specified processes.

PMIX_JOB_CTRL_TERMINATE **"pmix.jctrl.term"** (**bool**)
: Politely terminate the specified processes.

PMIX_REGISTER_CLEANUP **"pmix.reg.cleanup"** (**char***)
: Comma-delimited list of files to be removed upon process termination

PMIX_REGISTER_CLEANUP_DIR **"pmix.reg.cleanupdir"** (**char***)
: Comma-delimited list of directories to be removed upon process termination

PMIX_CLEANUP_RECURSIVE **"pmix.clnup.recurse"** (**bool**)
: Recursively cleanup all subdirectories under the specified one(s)

PMIX_CLEANUP_EMPTY **"pmix.clnup.empty"** (**bool**)
: Only remove empty subdirectories

PMIX_CLEANUP_IGNORE **"pmix.clnup.ignore"** (**char***)

Comma-delimited list of filenames that are not to be removed

PMIX_CLEANUP_LEAVE_TOPDIR **"pmix.clnup.lvtop"** (**bool**)
When recursively cleaning subdirectories, do not remove the top-level directory (the one given in the cleanup request)

Optional Attributes

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

PMIX_JOB_CTRL_CANCEL **"pmix.jctrl.cancel"** (**char***)
Cancel the specified request - the provided request ID must match the **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of **NULL** implies cancel all requests from this requestor.

PMIX_JOB_CTRL_RESTART **"pmix.jctrl.restart"** (**char***)
Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT **"pmix.jctrl.ckpt"** (**char***)
Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT **"pmix.jctrl.ckptev"** (**bool**)
Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL **"pmix.jctrl.ckptsig"** (**int**)
Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT **"pmix.jctrl.ckptsig"** (**int**)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.

PMIX_JOB_CTRL_PROVISION **"pmix.jctrl.pvn"** (**char***)
Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE **"pmix.jctrl.pvnimg"** (**char***)
Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE **"pmix.jctrl.preempt"** (**bool**)
Indicate that the job can be pre-empted.

Description

Non-blocking form of the **PMIx_Job_control** API. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX_RANK_WILDCARD** can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix_info_t** structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

7.4 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.

At the request of SMS vendors and members, a monitoring support interface has been included in the PMIx v2 standard. The defined API allows 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. In addition, heartbeat and file monitoring methods have been included in the PRI but are active only when requested.

7.4.1 PMIx_Process_monitor

Summary

Request that application processes be monitored.

Format

PMIx v3.0

C

```
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)
```

C

IN **monitor**
info (handle)

IN **error**
status (integer)

IN `directives`
Array of info structures (array of handles)

IN `ndirs`
Number of elements in the *directives* array (integer)

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*. Details of the result will be returned in the *results* array
- 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:

`PMIX_MONITOR_ID` `"pmix.monitor.id"` (`char*`)
Provide a string identifier for this request.

`PMIX_MONITOR_CANCEL` `"pmix.monitor.cancel"` (`char*`)
Identifier to be canceled (`NULL` means cancel all monitoring for this process).

`PMIX_MONITOR_APP_CONTROL` `"pmix.monitor.appctrl"` (`bool`)
The application desires to control the response to a monitoring event.

`PMIX_MONITOR_HEARTBEAT` `"pmix.monitor.mbeat"` (`void`)
Register to have the PMIx server monitor the requestor for heartbeats.

`PMIX_MONITOR_HEARTBEAT_TIME` `"pmix.monitor.btime"` (`uint32_t`)
Time in seconds before declaring heartbeat missed.

`PMIX_MONITOR_HEARTBEAT_DROPS` `"pmix.monitor.bdrop"` (`uint32_t`)
Number of heartbeats that can be missed before generating the event.

`PMIX_MONITOR_FILE` `"pmix.monitor.fmon"` (`char*`)
Register to monitor file for signs of life.

`PMIX_MONITOR_FILE_SIZE` `"pmix.monitor.fsize"` (`bool`)
Monitor size of given file is growing to determine if the application is running.

`PMIX_MONITOR_FILE_ACCESS` `"pmix.monitor.faccess"` (`char*`)

Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY **"pmix.monitor.fmod"** (**char***)
Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME **"pmix.monitor.ftime"** (**uint32_t**)
Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS **"pmix.monitor.fdrop"** (**uint32_t**)
Number of file checks that can be missed before generating the event.

Description

Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not become "wedged". When a monitor detects the specified alarm condition, it will generate an event notification using the provided error code and passing along any available relevant information. It is up to the caller to register a corresponding event handler.

The *monitor* argument is an attribute indicating the type of monitor being requested. For example, **PMIX_MONITOR_FILE** to indicate that the requestor is asking that a file be monitored.

The *error* argument is the status code to be used when generating an event notification alerting that the monitor has been triggered. The range of the notification defaults to **PMIX_RANGE_NAMESPACE** . This can be changed by providing a **PMIX_RANGE** directive.

The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency of checking to be done

7.4.2 PMIx_Process_monitor_nb

Summary

Request that application processes be monitored.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,
                        const pmix_info_t directives[], size_t ndirs,
                        pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **monitor**
info (handle)
IN **error**
status (integer)
IN **directives**
Array of info structures (array of handles)
IN **ndirs**
Number of elements in the *directives* array (integer)
IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)
IN **cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **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

Optional Attributes

The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

PMIX_MONITOR_ID **"pmix.monitor.id"** (**char***)
Provide a string identifier for this request.

PMIX_MONITOR_CANCEL **"pmix.monitor.cancel"** (**char***)
Identifier to be canceled (**NULL** means cancel all monitoring for this process).

PMIX_MONITOR_APP_CONTROL **"pmix.monitor.appctrl"** (**bool**)
The application desires to control the response to a monitoring event.

PMIX_MONITOR_HEARTBEAT **"pmix.monitor.mbeat"** (**void**)
Register to have the PMIx server monitor the requestor for heartbeats.

PMIX_MONITOR_HEARTBEAT_TIME **"pmix.monitor.btime"** (**uint32_t**)
Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
Number of file checks that can be missed before generating the event.

Description

Non-blocking form of the **PMIx_Process_monitor** API. The *cbfunc* function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

7.4.3 PMIx_Heartbeat

Summary

Send a heartbeat to the PMIx server library

Format

PMIx v2.0

C

```
PMIx_Heartbeat(void)
```

C

Description

A simplified macro wrapping **PMIx_Process_monitor_nb** that sends a heartbeat to the PMIx server library.

7.5 Logging

The logging interface supports posting information by applications and SMS elements to persistent storage. This function is *not* intended for output of computational results, but rather for reporting status and saving state information such as inserting computation progress reports into the application's SMS job log or error reports to the local syslog.

7.5.1 PMIx_Log

Summary

Log data to a data service.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_Log(const pmix_info_t data[], size_t ndata,
         const pmix_info_t directives[], size_t ndirs)
```

C

IN **data**
Array of info structures (array of handles)

IN **ndata**
Number of elements in the *data* array (**size_t**)

IN **directives**
Array of info structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

Return codes are one of the following:

PMIX_SUCCESS The logging request was successful.

PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry.

PMIX_ERR_NOT_SUPPORTED The PMIx implementation or host environment does not support this function.

Required Attributes

If the PMIx library does not itself perform this operation, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

PMIX_LOG_STDERR **"pmix.log.stderr"** (**char***)
Log string to **stderr**.

PMIX_LOG_STDOUT **"pmix.log.stdout"** (**char***)
Log string to **stdout**.

PMIX_LOG_SYSLOG **"pmix.log.syslog"** (**char***)
Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog

PMIX_LOG_LOCAL_SYSLOG **"pmix.log.lsys"** (**char***)
Log data to local syslog. Defaults to **ERROR** priority.

PMIX_LOG_GLOBAL_SYSLOG **"pmix.log.gsys"** (**char***)
Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.

PMIX_LOG_SYSLOG_PRI **"pmix.log.syspri"** (**int**)
Syslog priority level

PMIX_LOG_ONCE **"pmix.log.once"** (**bool**)
Only log this once with whichever channel can first support it, taking the channels in priority order

Optional Attributes

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

PMIX_LOG_SOURCE **"pmix.log.source"** (**pmix_proc_t***)
ID of source of the log request

PMIX_LOG_TIMESTAMP **"pmix.log.tstmp"** (**time_t**)
Timestamp for log report

PMIX_LOG_GENERATE_TIMESTAMP **"pmix.log.gtstmp"** (**bool**)
Generate timestamp for log

PMIX_LOG_TAG_OUTPUT **"pmix.log.tag"** (**bool**)
Label the output stream with the channel name (e.g., "stdout")

PMIX_LOG_TIMESTAMP_OUTPUT **"pmix.log.tsout"** (**bool**)
Print timestamp in output string

PMIX_LOG_XML_OUTPUT **"pmix.log.xml"** (**bool**)
Print the output stream in XML format

PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
Subject line for email.

PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
Message to be included in email.

PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
Log the provided information to the host environment's job record

PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
Store the log data in a global data store (e.g., database)

Description

Log data subject to the services offered by the host environment. The data to be logged is provided in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel.

Advice to users

It is strongly recommended that the **PMIx_Log** API not be used by applications for streaming data as it is not a "performant" transport and can perturb the application since it involves the local PMIx server and host SMS daemon. Note that a return of **PMIX_SUCCESS** only denotes that the data was successfully handed to the appropriate system call (for local channels) or the host environment and does not indicate receipt at the final destination.

7.5.2 PMIx_Log_nb

Summary

Log data to a data service.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Log_nb(const pmix_info_t data[], size_t ndata,
            const pmix_info_t directives[], size_t ndirs,
            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **data**
Array of info structures (array of handles)

IN **ndata**
Number of elements in the *data* array (**size_t**)

IN **directives**
Array of info structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Return codes are one of the following:

PMIX_SUCCESS The logging request is valid and is being processed. The resulting status from the operation will be provided in the callback function. Note that the library must not invoke the callback function prior to returning from the API.

PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry that prevents it from being processed. The callback function will not be called.

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function. The callback function will not be called.

Required Attributes

If the PMIx library does not itself perform this operation, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

PMIX_LOG_STDERR **"pmix.log.stderr"** (**char***)
: Log string to **stderr**.

PMIX_LOG_STDOUT **"pmix.log.stdout"** (**char***)
: Log string to **stdout**.

PMIX_LOG_SYSLOG **"pmix.log.syslog"** (**char***)
: Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog

PMIX_LOG_LOCAL_SYSLOG **"pmix.log.lsys"** (**char***)
: Log data to local syslog. Defaults to **ERROR** priority.

PMIX_LOG_GLOBAL_SYSLOG **"pmix.log.gsys"** (**char***)
: Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.

PMIX_LOG_SYSLOG_PRI **"pmix.log.syspri"** (**int**)
: Syslog priority level

PMIX_LOG_ONCE **"pmix.log.once"** (**bool**)
: Only log this once with whichever channel can first support it, taking the channels in priority order

Optional Attributes

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

PMIX_LOG_SOURCE **"pmix.log.source"** (**pmix_proc_t***)
: ID of source of the log request

PMIX_LOG_TIMESTAMP **"pmix.log.tstmp"** (**time_t**)
: Timestamp for log report

PMIX_LOG_GENERATE_TIMESTAMP **"pmix.log.gtstmp"** (**bool**)
: Generate timestamp for log

PMIX_LOG_TAG_OUTPUT **"pmix.log.tag"** (**bool**)
: Label the output stream with the channel name (e.g., "stdout")

PMIX_LOG_TIMESTAMP_OUTPUT **"pmix.log.tsout"** (**bool**)
: Print timestamp in output string

PMIX_LOG_XML_OUTPUT **"pmix.log.xml"** (**bool**)
: Print the output stream in XML format

PMIX_LOG_EMAIL **"pmix.log.email"** (**pmix_data_array_t**)
Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR **"pmix.log.emaddr"** (**char***)
Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SUBJECT **"pmix.log.emsub"** (**char***)
Subject line for email.

PMIX_LOG_EMAIL_MSG **"pmix.log.emmsg"** (**char***)
Message to be included in email.

PMIX_LOG_JOB_RECORD **"pmix.log.jrec"** (**bool**)
Log the provided information to the host environment's job record

PMIX_LOG_GLOBAL_DATASTORE **"pmix.log.gstore"** (**bool**)
Store the log data in a global data store (e.g., database)

Description

Log data subject to the services offered by the host environment. The data to be logged is provided in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The callback function will be executed when the log operation has been completed. The *data* and *directives* arrays must be maintained until the callback is provided.

Advice to users

It is strongly recommended that the **PMIx_Log_nb** API not be used by applications for streaming data as it is not a "performant" transport and can perturb the application since it involves the local PMIx server and host SMS daemon. Note that a return of **PMIX_SUCCESS** only denotes that the data was successfully handed to the appropriate system call (for local channels) or the host environment and does not indicate receipt at the final destination.

CHAPTER 8

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.

8.1 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 that includes 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

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

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

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

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

Users can specify the callback order of a handler within its category at the time of registration. Ordering can be specified either by providing the relevant returned event handler registration ID or using event handler names, if the user specified an event handler name when registering the corresponding event. Thus, users can specify that a given handler be executed before or after another handler should both handlers appear in an event chain (the ordering is ignored if the other handler isn't included). Note that ordering does not imply immediate relationships. For example, multiple handlers registered to be serviced after event handler *A* will all be executed after *A*, but are not guaranteed to be executed in any particular order amongst themselves.

In addition, one event handler can be declared as the *first* handler to be executed in the chain. This handler will *always* be called prior to any other handler, regardless of category, provided the incoming event matches both the specified range and event code. Only one handler can be so designated — attempts to designate additional handlers as *first* will return an error. Deregistration of the declared *first* handler will re-open the position for subsequent assignment.

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

Advice to users

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

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

8.1.1 PMIx_Register_event_handler

Summary

Register an event handler

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
                            pmix_info_t info[], size_t ninfo,
                            pmix_notification_fn_t evhdlr,
                            pmix_evhdlr_reg_cbfunc_t cbfunc,
                            void *cbdata);
```

C

IN **codes**
Array of status codes (array of **pmix_status_t**)

IN **ncodes**
Number of elements in the *codes* array (**size_t**)

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

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **evhdlr**
Event handler to be called **pmix_notification_fn_t** (function reference)

IN **cbfunc**
Callback function **pmix_evhdlr_reg_cbfunc_t** (function reference)

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

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

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

PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API. The event handler identifier will be returned in the callback

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.

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

Required Attributes

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

PMIX_EVENT_HDLR_NAME **"pmix.evname"** (**char***)
String name identifying this handler.

PMIX_EVENT_HDLR_FIRST **"pmix.evfirst"** (**bool**)
Invoke this event handler before any other handlers.

PMIX_EVENT_HDLR_LAST **"pmix.evlast"** (**bool**)
Invoke this event handler after all other handlers have been called.

PMIX_EVENT_HDLR_FIRST_IN_CATEGORY **"pmix.evfirstcat"** (**bool**)
Invoke this event handler before any other handlers in this category.

PMIX_EVENT_HDLR_LAST_IN_CATEGORY **"pmix.evlastcat"** (**bool**)
Invoke this event handler after all other handlers in this category have been called.

PMIX_EVENT_HDLR_BEFORE **"pmix.evbefore"** (**char***)
Put this event handler immediately before the one specified in the **(char*)** value.

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

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

PMIX_EVENT_HDLR_PREPEND **"pmix.evprepend"** (**bool**)
Prepend this handler to the precedence list within its category.

PMIX_EVENT_HDLR_APPEND **"pmix.evappend"** (**bool**)
Append this handler to the precedence list within its category.

PMIX_EVENT_CUSTOM_RANGE **"pmix.evrange"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining range of event notification.

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_EVENT_RETURN_OBJECT **"pmix.evobject"** (**void ***)
Object to be returned whenever the registered callback function **cbfunc** is invoked. The object will only be returned to the process that registered it.

Host environments that implement support for PMIx event notification are required to support the following attributes:

PMIX_EVENT_AFFECTED_PROC **"pmix.evproc"** (**pmix_proc_t**)
The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS **"pmix.evaffected"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining affected processes.

Optional Attributes

Host environments that support PMIx event notification *may* offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes are optional for host environments that suppport this operation:

PMIX_EVENT_TERMINATE_SESSION **"pmix.evterm.sess"** (**bool**)
The RM intends to terminate this session.

PMIX_EVENT_TERMINATE_JOB **"pmix.evterm.job"** (**bool**)
The RM intends to terminate this job.

PMIX_EVENT_TERMINATE_NODE **"pmix.evterm.node"** (**bool**)
The RM intends to terminate all processes on this node.

PMIX_EVENT_TERMINATE_PROC **"pmix.evterm.proc"** (**bool**)
The RM intends to terminate just this process.

PMIX_EVENT_ACTION_TIMEOUT **"pmix.evtimeout"** (**int**)
The time in seconds before the RM will execute error response.

PMIX_EVENT_SILENT_TERMINATION **"pmix.evsilentterm"** (**bool**)

Do not generate an event when this job normally terminates.

Description

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

Advice to users

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

Advice to users

As previously stated, upon completing its work, and prior to returning, each handler *must* call the event handler completion function provided when it was invoked (including a status code plus any information to be passed to later handlers) so that the chain can continue being progressed. An event handler can terminate all further progress along the chain by passing the **PMIX_EVENT_ACTION_COMPLETE** status to the completion callback function. Note that the parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once 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.

8.1.2 PMIx_Deregister_event_handler

Summary

Deregister an event handler.

Format

PMIx v2.0

C

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

C

IN **evhdlr_ref**
Event handler ID returned by registration (**size_t**)

IN **cbfunc**
Callback function to be executed upon completion of operation **pmix_op_cbfunc_t** (function reference)

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

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

If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call 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*).

8.1.3 PMIx_Notify_event

Summary

Report an event for notification via any registered event handler.

Format

PMIx v2.0

C

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

C

IN **status**
Status code of the event (**pmix_status_t**)

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

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

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

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **cbfunc**
Callback function to be executed upon completion of operation **pmix_op_cbfunc_t** (function reference)

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

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

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

PMIX_SUCCESS The notification request is valid and is being processed. The callback function will be called when the process-local operation is complete and will provide the resulting status of that operation. Note that this does *not* reflect the success or failure of delivering the event to any recipients. The callback function must not be executed prior to returning from the API.

PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called

PMIX_ERR_BAD_PARAM The request contains at least one incorrect entry that prevents it from being processed. The callback function will *not* be called.

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

Required Attributes

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

PMIX_EVENT_NON_DEFAULT **"pmix.evnondef"** (**bool**)
Event is not to be delivered to default event handlers.

PMIX_EVENT_CUSTOM_RANGE **"pmix.evrange"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining range of event notification.

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

PMIX_EVENT_AFFECTED_PROC **"pmix.evproc"** (**pmix_proc_t**)
The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS **"pmix.evaffected"** (**pmix_data_array_t***)
Array of **pmix_proc_t** defining affected processes.

Description

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

Host SMS daemons call the API to pass events down to its embedded PMIx server both for transmittal to local client processes and for the server's own internal processing.

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

Advice to users

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

CHAPTER 9

Data Packing and Unpacking

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

9.1 Data Buffer Type

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

PMIx v2.0

C

```
typedef struct pmix_data_buffer {
    /** Start of my memory */
    char *base_ptr;
    /** Where the next data will be packed to
        (within the allocated memory starting
        at base_ptr) */
    char *pack_ptr;
    /** Where the next data will be unpacked
        from (within the allocated memory
        starting as base_ptr) */
    char *unpack_ptr;
    /** Number of bytes allocated (starting
        at base_ptr) */
    size_t bytes_allocated;
    /** Number of bytes used by the buffer
        (i.e., amount of data -- including
        overhead -- packed in the buffer) */
    size_t bytes_used;
} pmix_data_buffer_t;
```

C

9.2 Support Macros

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

9.2.1 PMIX_DATA_BUFFER_CREATE

Summary

Allocate memory for a **pmix_data_buffer_t** object and initialize it

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_CREATE(buffer);
```

C

OUT **buffer**

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

Description

This macro uses *calloc* to allocate memory for the buffer and initialize all fields in it

9.2.2 PMIX_DATA_BUFFER_RELEASE

Summary

Free a **pmix_data_buffer_t** object and the data it contains

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_RELEASE(buffer);
```

C

IN **buffer**

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

Description

Free's the data contained in the buffer, and then free's the buffer itself

9.2.3 PMIX_DATA_BUFFER_CONSTRUCT

Summary

Initialize a statically declared **pmix_data_buffer_t** object

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_CONSTRUCT(buffer);
```

C

IN **buffer**
Pointer to the allocated **pmix_data_buffer_t** that is to be initialized (handle)

Description

Initialize a pre-allocated buffer object

9.2.4 PMIX_DATA_BUFFER_DESTRUCT

Summary

Release the data contained in a **pmix_data_buffer_t** object

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_DESTRUCT(buffer);
```

C

IN **buffer**
Pointer to the **pmix_data_buffer_t** whose data is to be released (handle)

Description

Free's the data contained in a **pmix_data_buffer_t** object

9.2.5 PMIX_DATA_BUFFER_LOAD

Summary

Load a blob into a **pmix_data_buffer_t** object

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_LOAD(buffer, data, size);
```

C

IN **buffer**
Pointer to a pre-allocated **pmix_data_buffer_t** (handle)

IN **data**
Pointer to a blob (**char***)

IN **size**
Number of bytes in the blob **size_t**

Description

Load the given data into the provided **pmix_data_buffer_t** object, usually done in preparation for unpacking the provided data. Note that the data is *not* copied into the buffer - thus, the blob must not be released until after operations on the buffer have completed.

9.2.6 PMIX_DATA_BUFFER_UNLOAD

Summary

Unload the data from a **pmix_data_buffer_t** object

Format

PMIx v2.0

C

```
PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

C

IN **buffer**
: Pointer to the **pmix_data_buffer_t** whose data is to be extracted (handle)

OUT **data**
: Variable to be assigned the pointer to the extracted blob (**void***)

OUT **size**
: Variable to be assigned the number of bytes in the blob **size_t**

Description

Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to the provided variables, usually done to transmit the blob to a remote process for unpacking. The buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release - thus, the user is responsible for releasing the blob when done with it.

9.3 General Routines

The following routines are provided to support internode transfers in heterogeneous environments.

9.3.1 PMIx_Data_pack

Summary

Pack one or more values of a specified type into a buffer, usually for transmission to another process

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Data_pack(const pmix_proc_t *target,
               pmix_data_buffer_t *buffer,
               void *src, int32_t num_vals,
               pmix_data_type_t type);
```

C

IN **target**
Pointer to a **pmix_proc_t** containing the nspace/rank of the process that will be unpacking the final buffer. A NULL value may be used to indicate that the target is based on the same PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

IN **buffer**
Pointer to a **pmix_data_buffer_t** where the packed data is to be stored (handle)

IN **src**
Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the caller must pass the address of the pointer to the string as the (void*). This allows the caller to pass multiple strings in a single call. (memory reference)

IN **num_vals**
Number of elements pointed to by the *src* pointer. A string value is counted as a single value regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g., string arrays) should be contiguous, although the data pointed to need not be contiguous across array entries.(**int32_t**)

IN **type**
The type of the data to be packed (**pmix_data_type_t**)

Returns one of the following:

PMIX_SUCCESS The data has been packed as requested
PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
PMIX_ERR_BAD_PARAM The provided buffer or src is **NULL**
PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this implementation
PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation
PMIX_ERROR General error

Description

The pack function packs one or more values of a specified type into the specified buffer. The buffer must have already been initialized via the **PMIX_DATA_BUFFER_CREATE** or **PMIX_DATA_BUFFER_CONSTRUCT** macros — otherwise, **PMIx_Data_pack** will return an error. Providing an unsupported type flag will likewise be reported as an error.

Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may lose precision when unpacked by a non-homogeneous recipient. The **PMIx_Data_pack** function

will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code (generated upon unpacking) — the error cannot be detected during packing.

The namespace of the intended recipient of the packed buffer (i.e., the process that will be unpacking it) is used solely to resolve any data type differences between PMIx versions. The recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the recipient is using. Note that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the target's namespace.

9.3.2 PMIx_Data_unpack

Summary

Unpack values from a **pmix_data_buffer_t**

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Data_unpack(const pmix_proc_t *source,
                 pmix_data_buffer_t *buffer, void *dest,
                 int32_t *max_num_values,
                 pmix_data_type_t type);
```

C

IN **source**
Pointer to a **pmix_proc_t** structure containing the nspace/rank of the process that packed the provided buffer. A NULL value may be used to indicate that the source is based on the same PMIx version as the caller. Note that only the source's nspace is relevant. (handle)

IN **buffer**
A pointer to the buffer from which the value will be extracted. (handle)

INOUT **dest**
A pointer to the memory location into which the data is to be stored. Note that these values will be stored contiguously in memory. For strings, this pointer must be to (char**) to provide a means of supporting multiple string operations. The unpack function will allocate memory for each string in the array - the caller must only provide adequate memory for the array of pointers. (**void***)

INOUT **max_num_values**
The number of values to be unpacked — upon completion, the parameter will be set to the actual number of values unpacked. In most cases, this should match the maximum number provided in the parameters — but in no case will it exceed the value of this parameter. Note that unpacking fewer values than are actually available will leave the buffer in an unpackable state — the function will return an error code to warn of this condition.(**int32_t**)

IN **`type`**
The type of the data to be unpacked — must be one of the PMIx defined data types (**`pmix_data_type_t`**)

Returns one of the following:

`PMIX_SUCCESS` The data has been unpacked as requested
`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.
`PMIX_ERR_BAD_PARAM` The provided buffer or dest is **`NULL`**
`PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this implementation
`PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation
`PMIX_ERROR` General error

Description

The unpack function unpacks the next value (or values) of a specified type from the given buffer. The buffer must have already been initialized via an **`PMIX_DATA_BUFFER_CREATE`** or **`PMIX_DATA_BUFFER_CONSTRUCT`** call (and assumedly filled with some data) — otherwise, the unpack_value function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte field in a string array that so happens to have a value that matches the specified data type flag). Therefore, the data type error check is *not* completely safe.

Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer. It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the unpack_ptr.

Warning: The caller is responsible for providing adequate memory storage for the requested data. The user must provide a parameter indicating the maximum number of values that can be unpacked into the allocated memory. If more values exist in the buffer than can fit into the memory storage, then the function will unpack what it can fit into that location and return an error code indicating that the buffer was only partially unpacked.

Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code generated upon unpacking — these errors cannot be detected during packing.

The namespace of the process that packed the buffer is used solely to resolve any data type differences between PMIx versions. The packer must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the packer is using. Note

that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the packer's namespace.

9.3.3 PMIx_Data_copy

Summary

Copy a data value from one location to another.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Data_copy(void **dest, void *src,
               pmix_data_type_t type);
```

C

IN **dest**
The address of a pointer into which the address of the resulting data is to be stored. (**void****)

IN **src**
A pointer to the memory location from which the data is to be copied (handle)

IN **type**
The type of the data to be copied — must be one of the PMIx defined data types. (**pmix_data_type_t**)

Returns one of the following:

PMIX_SUCCESS The data has been copied as requested
PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
PMIX_ERR_BAD_PARAM The provided src or dest is **NULL**
PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this implementation
PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation
PMIX_ERROR General error

Description

Since registered data types can be complex structures, the system needs some way to know how to copy the data from one location to another (e.g., for storage in the registry). This function, which can call other copy functions to build up complex data types, defines the method for making a copy of the specified data type.

9.3.4 PMIx_Data_print

Summary

Pretty-print a data value.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Data_print(char **output, char *prefix,
                void *src, pmix_data_type_t type);
```

C

IN **output**
The address of a pointer into which the address of the resulting output is to be stored. (**char****)

IN **prefix**
String to be prepended to the resulting output (**char***)

IN **src**
A pointer to the memory location of the data value to be printed (handle)

IN **type**
The type of the data value to be printed — must be one of the PMIx defined data types. (**pmix_data_type_t**)

Returns one of the following:

PMIX_SUCCESS The data has been printed as requested
PMIX_ERR_BAD_PARAM The provided data type is not recognized.
PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.

Description

Since registered data types can be complex structures, the system needs some way to know how to print them (i.e., convert them to a string representation). Primarily for debug purposes.

9.3.5 PMIx_Data_copy_payload

Summary

Copy a payload from one buffer to another

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
                       pmix_data_buffer_t *src);
```

C

IN `dest`
Pointer to the destination `pmix_data_buffer_t` (handle)

IN `src`
Pointer to the source `pmix_data_buffer_t` (handle)

Returns one of the following:

`PMIX_SUCCESS` The data has been copied as requested
`PMIX_ERR_BAD_PARAM` The src and dest `pmix_data_buffer_t` types do not match
`PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

Description

This function will append a copy of the payload in one buffer into another buffer. Note that this is *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

CHAPTER 10

Security

PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes. *Client* processes (i.e., processes started by the host environment) must be preregistered with the PMIx server library via the **PMIx_server_register_client** API before they are spawned. This API requires that you pass the expected uid/gid of the client process.

When the client attempts to connect to the PMIx server, the server uses available standard Operating System (OS) methods to determine the effective uid/gid of the process requesting the connection. PMIx implementations shall not rely on any values reported by the client process itself as that would be unsafe. The effective uid/gid reported by the OS is compared to the values provided by the host during registration - if they don't match, the PMIx server is required to drop the connection request. This ensures that the PMIx server does not allow connection from a client that doesn't at least meet some minimal security requirement.

Once the requesting client passes the initial test, the PMIx server can, at the choice of the implementor, perform additional security checks. This may involve a variety of methods such as exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server reports the client connection request to the host via the **pmix_server_client_connected_fn_t** interface. The host may then perform any additional checks and operations before responding with either **PMIX_SUCCESS** to indicate that the connection is approved, or a PMIx error constant indicating that the connection request is refused. In this latter case, the PMIx server is required to drop the connection.

Tools started by the host environment are classed as a subgroup of client processes and follow the client process procedure. However, tools that are not started by the host environment must be handled differently as registration information is not available prior to the connection request. In these cases, the PMIx server library is required to use available standard OS methods to get the effective uid/gid and report them upwards as part of invoking the **pmix_server_tool_connection_fn_t** interface, deferring initial security screening to the host. It is recognized that this may represent a security risk - for this reason, PMIx server libraries must not enable tool connections by default. Instead, the host has to explicitly enable them via the **PMIX_SERVER_TOOL_SUPPORT** attribute, thus recognizing the associated risk. Once the host has completed its authentication procedure, it again informs the PMIx server of the result.

Applications and tools often interact with the host environment in ways that require security beyond just verifying the user's identity - e.g., access to that user's relevant authorizations. This is particularly important when tools connect directly to a system-level PMIx server that may be operating at a privileged level. A variety of system management software packages provide authorization services, but the lack of standardized interfaces makes portability problematic.

This section defines two PMIx client-side APIs for this purpose. These are most likely to be used by user-space applications/tools, but are not restricted to that realm.

10.1 Obtaining Credentials

The API for obtaining a credential is a non-blocking operation since the host environment may have to contact a remote credential service. The definition takes into account the potential that the returned credential could be sent via some mechanism to another application that resides in an environment using a different security mechanism. Thus, provision is made for the system to return additional information (e.g., the identity of the issuing agent) outside of the credential itself and visible to the application.

10.1.1 PMIx_Get_credential

Summary

Request a credential from the PMIx server library or the host environment

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,
                    pmix_byte_object_t *credential)
```

C

IN **info**
Array of **pmix_info_t** structures (array of handles)

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **credential**
Address of a **pmix_byte_object_t** within which to return credential (handle)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the credential has been returned in the provided **pmix_byte_object_t**
- a PMIx error constant indicating either an error in the input or that the request is unsupported

Required Attributes

PMIx libraries that choose not to support this operation *must* return **PMIX_ERR_NOT_SUPPORTED** when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Request a credential from the PMIx server library or the host environment

10.1.2 PMIx_Get_credential_nb

Summary

Request a credential from the PMIx server library or the host environment

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,
                       pmix_credential_cbfunc_t cbfunc, void *cbdata)
```

C

IN **info**
Array of **pmix_info_t** structures (array of handles)

IN **ninfo**
Number of elements in the *info* array (**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

PMIx libraries that choose not to support this operation *must* return **PMIX_ERR_NOT_SUPPORTED** when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Request a credential from the PMIx server library or the host environment

10.2 Validating Credentials

The API for validating a credential is a non-blocking operation since the host environment may have to contact a remote credential service. Provision is made for the system to return additional information regarding possible authorization limitations beyond simple authentication.

10.2.1 PMIx_Validate_credential

Summary

Request validation of a credential by the PMIx server library or the host environment

Format

PMIx v3.0

C

```
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)
```

C

IN `cred`
Pointer to `pmix_byte_object_t` containing the credential (handle)

IN `info`
Array of `pmix_info_t` structures (array of handles)

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

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*. Details of the result will be returned in the *results* array
- a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

PMIx libraries that choose not to support this operation *must* return `PMIX_ERR_NOT_SUPPORTED` when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

`PMIX_USERID` `"pmix.euid"` (`uint32_t`)
Effective user id.

`PMIX_GRPID` `"pmix.egid"` (`uint32_t`)
Effective group id.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Request validation of a credential by the PMIx server library or the host environment.

10.2.2 PMIx_Validate_credential_nb

Summary

Request validation of a credential by the PMIx server library or the host environment

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
                            const pmix_info_t info[], size_t ninfo,
                            pmix_validation_cbfunc_t cbfunc,
                            void *cbdata)
```

C

IN **cred**
Pointer to **pmix_byte_object_t** containing the credential (handle)

IN **info**
Array of **pmix_info_t** structures (array of handles)

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **cbfunc**
Callback function to return result (**pmix_validation_cbfunc_t** function reference)

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

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

PMIx libraries that choose not to support this operation *must* return **PMIX_ERR_NOT_SUPPORTED** when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Request validation of a credential by the PMIx server library or the host environment.

CHAPTER 11

Server-Specific Interfaces

The RM daemon 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 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.

11.1 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

11.1.1 PMIx_generate_regex

Summary

Generate a compressed representation of the input string.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_generate_regex(const char *input, char **output)
```

C

IN `input`
String to process (string)

OUT `output`
Compressed representation of *input* (array of bytes)

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

Description

Given a comma-separated list of *input* values, generate a reduced size representation of the input that can be passed down to the PMIx server library's **PMIx_server_register_nspace** API for parsing. The order of the individual values in the *input* string is preserved across the operation. The caller is responsible for releasing the returned data.

The precise compressed representations will be implementation specific. However, all PMIx implementations are required to include a **NULL**-terminated string in the output representation that can be printed for diagnostic purposes.

Advice to PMIx server hosts

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with **"pmix:\0"** might indicate that the representation is a PMIx-defined regular expression represented as a **NULL**-terminated string following the **"pmix:\0"** prefix. In contrast, an output starting with **"blob:\0"** might indicate a compressed binary array follows the prefix.

Communicating the resulting output should be done by first packing the returned expression using the **PMIx_Data_pack** , declaring the input to be of type **PMIX_REGEX** , and then obtaining the resulting blob to be communicated using the **PMIX_DATA_BUFFER_UNLOAD** macro. The reciprocal method can be used on the remote end prior to passing the regex into **PMIx_server_register_nspace** . The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

11.1.2 PMIx_generate_ppn

Summary

Generate a compressed representation of the input identifying the processes on each node.

Format

PMIx v1.0

C

```
pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)
```

C

IN **input**
String to process (string)

OUT **ppn**
Compressed representation of *input* (array of bytes)

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

Description

The input shall consist of a semicolon-separated list of ranges representing the ranks of processes 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 correspond to the node name provided at that position in the input to **PMIx_generate_regex**. Thus, in the example, ranks 1-4 would be located on the first node of the comma-separated list of names provided to **PMIx_generate_regex**, and ranks 2-5 would be on the second name in the list.

Advice to PMIx server hosts

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with **"pmix:"** indicates that the representation is a PMIx-defined regular expression represented as a NULL-terminated string. In contrast, an output starting with **"blob:\0size=1234:"** is a compressed binary array.

Communicating the resulting output should be done by first packing the returned expression using the **PMIx_Data_pack**, declaring the input to be of type **PMIX_REGEX**, and then obtaining the blob to be communicated using the **PMIX_DATA_BUFFER_UNLOAD** macro. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

11.1.3 PMIx_server_register_nspace

Summary

Setup the data about a particular namespace.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_register_nspace(const pmix_nspace_t nspace,
                            int nlocalprocs,
                            pmix_info_t info[], size_t ninfo,
                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **nspace**
Character array of maximum size **PMIX_MAX_NSLEN** containing the namespace identifier (string)

IN **nlocalprocs**
number of local processes (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

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

PMIX_REGISTER_NODATA **"pmix.reg.nodata"** (**bool**)
Registration is for this namespace only, do not copy job data - this attribute is not accessed using the **PMIx_Get**

Host environments are required to provide the following attributes:

- for the session containing the given namespace:
  - **PMIX_UNIV_SIZE** **"pmix.univ.size"** (**uint32_t**)
    Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is the equivalent to the combination of **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it is included in the Standard for historical reasons.
- for the given namespace:
  - **PMIX_JOBID** **"pmix.jobid"** (**char***)
    Job identifier assigned by the scheduler.
  - **PMIX_JOB_SIZE** **"pmix.job.size"** (**uint32_t**)
    Total number of processes in this job across all contained applications. Note that this value can be different from **PMIX_MAX_PROCS** . For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running **job**

on-they-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.

- **PMIX_MAX_PROCS** "**pmix.max.size**" (**uint32_t**)
  Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description.

- **PMIX_NODE_MAP** "**pmix.nmap**" (**char***)
  Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.

- **PMIX_PROC_MAP** "**pmix.pmap**" (**char***)
  Regular expression describing processes on each node - see 11.1.3.1 for an explanation of its generation.

- for its own node:

  - **PMIX_LOCAL_SIZE** "**pmix.local.size**" (**uint32_t**)
    Number of processes in this job or application on this node.

  - **PMIX_LOCAL_PEERS** "**pmix.lpeers**" (**char***)
    Comma-delimited list of ranks on this node within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

  - **PMIX_LOCAL_CPUSETS** "**pmix.lcpus**" (**char***)
    Colon-delimited cpusets of local peers within the specified namespace - referenced using **PMIX_RANK_WILDCARD** .

- for each process in the given namespace:

  - **PMIX_RANK** "**pmix.rank**" (**pmix_rank_t**)
    Process rank within the job.

  - **PMIX_LOCAL_RANK** "**pmix.lrank**" (**uint16_t**)
    Local rank on this node within this job.

  - **PMIX_NODE_RANK** "**pmix.nrank**" (**uint16_t**)
    Process rank on this node spanning all jobs.

  - **PMIX_NODEID** "**pmix.nodeid**" (**uint32_t**)
    Node identifier where the specified process is located, expressed as the node's index (beginning at zero) in the array resulting from expansion of the **PMIX_NODE_MAP** regular expression for the **job**

If more than one application is included in the namespace, then the host environment is also required to provide the following attributes:

- for each application:

  - **PMIX_APPNUM** "**pmix.appnum**" (**uint32_t**)
    Application number within the job.

- **PMIX_APPLDR** **"pmix.aldr"** (**pmix_rank_t**)
  Lowest rank in this application within this job - referenced using **PMIX_RANK_WILDCARD** .
- **PMIX_APP_SIZE** **"pmix.app.size"** (**uint32_t**)
  Number of processes in this application.

- for each process:
  - **PMIX_APP_RANK** **"pmix.apprank"** (**pmix_rank_t**)
    Process rank within this application.
  - **PMIX_APPNUM** **"pmix.appnum"** (**uint32_t**)
    Application number within the job.

Optional Attributes

The following attributes may be provided by host environments:

- for the session containing the given namespace:
  - **PMIX_SESSION_ID** **"pmix.session.id"** (**uint32_t**)
    Session identifier - referenced using **PMIX_RANK_WILDCARD** .
- for the given namespace:
  - **PMIX_SERVER_NSPACE** **"pmix.srv.nspace"** (**char***)
    Name of the namespace to use for this PMIx server.
  - **PMIX_SERVER_RANK** **"pmix.srv.rank"** (**pmix_rank_t**)
    Rank of this PMIx server
  - **PMIX_NPROC_OFFSET** **"pmix.offset"** (**pmix_rank_t**)
    Starting global rank of this job - referenced using **PMIX_RANK_WILDCARD** .
  - **PMIX_ALLOCATED_NODELIST** **"pmix.alist"** (**char***)
    Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using **PMIX_RANK_WILDCARD** .
  - **PMIX_JOB_NUM_APPS** **"pmix.job.napps"** (**uint32_t**)
    Number of applications in this job.
  - **PMIX_MAPBY** **"pmix.mapby"** (**char***)
    Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace
  - **PMIX_RANKBY** **"pmix.rankby"** (**char***)

Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

- **PMIX_BINDTO** "**pmix.bindto**" (**char***)
  Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

- **PMIX_ANL_MAP** "**pmix.anlmap**" (**char***)
  Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.

- for its own node:

  - **PMIX_AVAIL_PHYS_MEMORY** "**pmix.pmem**" (**uint64_t**)
    Total available physical memory on this node.

  - **PMIX_HWLOC_XML_V1** "**pmix.hwlocxml1**" (**char***)
    XML representation of local topology using HWLOC's v1.x format.

  - **PMIX_HWLOC_XML_V2** "**pmix.hwlocxml2**" (**char***)
    XML representation of local topology using HWLOC's v2.x format.

  - **PMIX_LOCALLDR** "**pmix.lldr**" (**pmix_rank_t**)
    Lowest rank on this node within this job - referenced using **PMIX_RANK_WILDCARD** .

  - **PMIX_NODE_SIZE** "**pmix.node.size**" (**uint32_t**)
    Number of processes across all jobs on this node.

  - **PMIX_LOCAL_PROCS** "**pmix.lprocs**" (**pmix_proc_t array**)
    Array of **pmix_proc_t** of all processes on the specified node - referenced using **PMIX_RANK_WILDCARD** .

- for each process in the given namespace:

  - **PMIX_PROCID** "**pmix.procid**" (**pmix_proc_t**)
    Process identifier

  - **PMIX_GLOBAL_RANK** "**pmix.grank**" (**pmix_rank_t**)
    Process rank spanning across all jobs in this session.

  - **PMIX_HOSTNAME** "**pmix.hname**" (**char***)
    Name of the host where the specified process is running.

Attributes not directly provided by the host environment may be derived by the PMIx server library from other required information and included in the data made available to the server library's clients.

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 namespace. The PMIx

server library may utilize this information to customize the environment to fit that model (e.g., adding environmental variables specified by the corresponding standard for that model):

- **PMIX_PROGRAMMING_MODEL** **"pmix.pgm.model"** (**char***)
  Programming model being initialized (e.g., "MPI" or "OpenMP")
- **PMIX_MODEL_LIBRARY_NAME** **"pmix.mdl.name"** (**char***)
  Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
- **PMIX_MODEL_LIBRARY_VERSION** **"pmix.mld.vrs"** (**char***)
  Programming model version string (e.g., "2.1.1")

Description

Pass job-related information to the PMIx server library for distribution to local client processes.

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting any local application process within the given namespace.

The PMIx server must register all namespaces that will participate in collective operations with local processes. This means that the server must register a namespace even if it will not host any local processes from within that namespace if any local process of another namespace might at some point perform an operation involving one or more processes from the new namespace. This is necessary so that the collective operation can identify the participants and know when it is locally complete.

The caller must also provide the number of local processes that will be launched within this namespace. This is required for the PMIx server library to correctly handle collectives as a collective operation call can occur before all the local processes have been started.

Advice to users

The number of local processes for any given namespace is generally fixed at the time of application launch. Calls to **PMIx_Spawn** result in processes launched in their own namespace, not that of their parent. However, it is possible for processes to *migrate* to another node via a call to **PMIx_Job_control_nb** , thus resulting in a change to the number of local processes on both the initial node and the node to which the process moved. It is therefore critical that applications not migrate processes without first ensuring that PMIx-based collective operations are not in progress, and that no such operations be initiated until process migration has completed.

11.1.3.1 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it is describes how information provided in the *info* parameter of the **PMIx_server_register_nspace** shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

Advice to PMIx server hosts

Creating the *info* array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single **pmix_info_t** structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final *info* array can be allocated based on the number of elements on the list, and then the values in the list object **pmix_info_t** structures transferred to the corresponding array element utilizing the **PMIX_INFO_XFER** macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a **session** or **job** . PMIx provides several tools to facilitate this operation, beginning by constructing an argv-like array of node names. This array is then passed to the **PMIx_generate_regex** function to create a regular expression parseable by the PMIx server library, as shown below:

C

```
char **nodes = NULL;
char *nodelist;
char *regex;
size_t n;
pmix_status_t rc;
pmix_info_t info;

/* loop over an array of nodes, adding each
 * name to the array */
for (n=0; n < num_nodes; n++)
    /* filter the nodes to ignore those not included
     * in the target range (session, job, etc.). In
     * this example, all nodes are accepted */
    PMIX_ARGV_APPEND(&nodes, node[n]->name);


/* join into a comma-delimited string */
nodelist = PMIX_ARGV_JOIN(nodes, ',');
```

```c
/* release the array */
PMIX_ARGV_FREE(nodes);

/* generate regex */
rc = PMIx_generate_regex(nodelist, &regex);

/* release list */
free(nodelist);

/* pass the regex as the value to the PMIX_NODE_MAP key */
PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
/* release the regex */
free(regex);
```

C

Changing the filter criteria allows the construction of node maps for any level of information.

A similar method is used to construct the map of processes on each node from the namespace being registered. This may be done for each information level of interest (e.g., to identify the process map for the entire **job** or for each **application** in the job) by changing the search criteria. An example is shown below for the case of creating the process map for a **job** :

C

```c
char **ndppn;
char rank[30];
char **ppnarray = NULL;
char *ppn;
char *localranks;
char *regex;
size_t n, m;
pmix_status_t rc;
pmix_info_t info;

/* loop over an array of nodes */
for (n=0; n < num_nodes; n++)
    /* for each node, construct an array of ranks on that node */
    ndppn = NULL;
    for (m=0; m < node[n]->num_procs; m++)
        /* ignore processes that are not part of the target job */
        if (!PMIX_CHECK_NSPACE(targetjob,node[n]->proc[m].nspace))
            continue;

        snprintf(rank, 30, "%d", node[n]->proc[m].rank);
        PMIX_ARGV_APPEND(&ndppn, rank);
```

```
        /* convert the array into a comma-delimited string of ranks */
        localranks = PMIX_ARGV_JOIN(ndppn, ',');
        /* release the local array */
        PMIX_ARGV_FREE(ndppn);
        /* add this node's contribution to the overall array */
        PMIX_ARGV_APPEND(&ppnarray, localranks);
        /* release the local list */
        free(localranks);


    /* join into a semicolon-delimited string */
    ppn = PMIX_ARGV_JOIN(ppnarray, ';');

    /* release the array */
    PMIX_ARGV_FREE(ppnarray);

    /* generate ppn regex */
    rc = PMIx_generate_ppn(ppn, &regex);

    /* release list */
    free(ppn);

    /* pass the regex as the value to the PMIX_PROC_MAP key */
    PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
    /* release the regex */
    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 identifed 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):

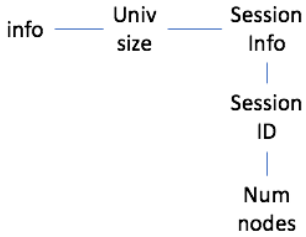

Figure 11.1.: Session-level information elements

- 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:

  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** and **PMIX_APPLDR** . If the **job** contains only a single **application** , then the application-specific values can be specified independently - i.e., in their own **pmix_info_t**

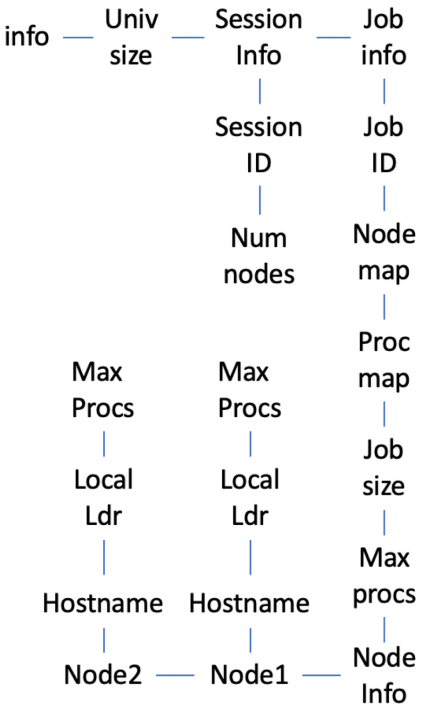

Figure 11.2.: Job-level information elements

elements of the *info* array - or as a **pmix_data_array_t** array of **pmix_info_t** using the **PMIX_APP_INFO_ARRAY** attribute and identifed by including the **PMIX_APPNUM** attribute in the array. Use of the array format is must in cases where non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the **PMIX_APP_INFO_ARRAY** format, each identified by its **PMIX_APPNUM** value.

Upon conclusion of this step, the *info* array might look like that shown in 11.3, assuming there are two applications in the job being registered:

- Process-level information includes an entry for each process in the job being registered, each entry marked with the **PMIX_PROC_DATA** attribute. The **rank** of the process must be the first entry in the array - this provides efficiency when storing the data. Upon conclusion of this step, the *info* array might look like the diagram in 11.4:
- For purposes of this example, node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases,

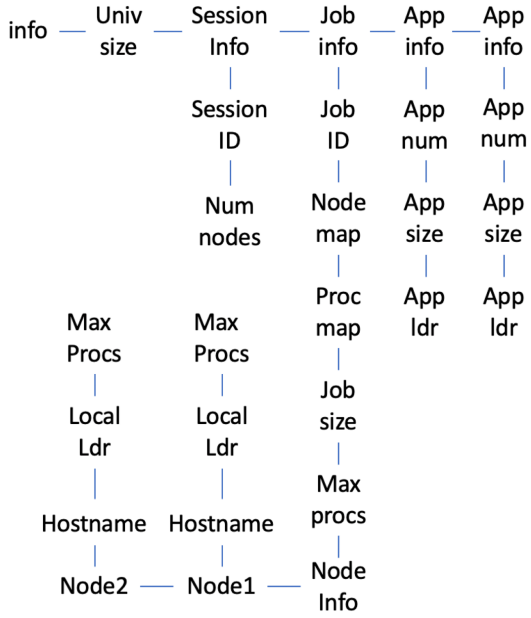

Figure 11.3.: Application-level information elements

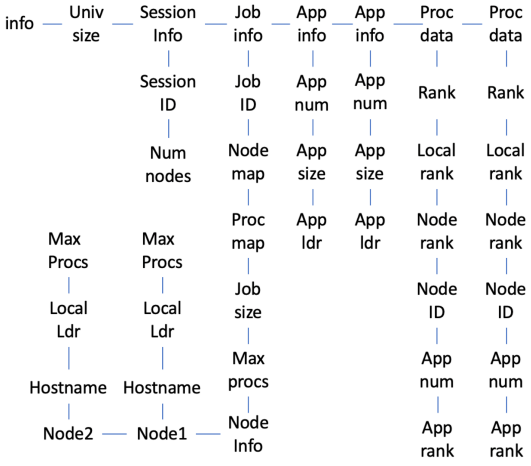

Figure 11.4.: Process-level information elements

the values included in this level are unique to it and can be specified independently - i.e., in their own **pmix_info_t** elements of the *info* array. Alternatively, they can be provided as a **pmix_data_array_t** array of **pmix_info_t** using the **PMIX_NODE_INFO_ARRAY**

attribute - this is required in cases where non-specific attributes are passed to describe aspects of the node, or where values for multiple nodes are being provided.

The node-level information requires two elements that must be constructed in a manner similar to that used for the node map. The **PMIX_LOCAL_PEERS** value is computed based on the processes on the local node, filtered to select those from the job being registered, as shown below using the tools provided by PMIx:

C

```
char **ndppn = NULL;
char rank[30];
char *localranks;
size_t m;
pmix_info_t info;

for (m=0; m < mynode->num_procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace))
        continue;

    snprintf(rank, 30, "%d", mynode->proc[m].rank);
    PMIX_ARGV_APPEND(&ndppn, rank);

/* convert the array into a comma-delimited string of ranks */
localranks = PMIX_ARGV_JOIN(ndppn, ',');
/* release the local array */
PMIX_ARGV_FREE(ndppn);

/* pass the string as the value to the PMIX_LOCAL_PEERS key */
PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
/* release the list */
free(localranks);
```

C

The **PMIX_LOCAL_CPUSETS** value is constructed in a similar manner. In the provided example, it is assumed that the 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:

C

```
char **ndcpus = NULL;
char *localcpus;
size_t m;
pmix_info_t info;

for (m=0; m < mynode->num_procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace))
        continue;

    PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);

/* convert the array into a colon-delimited string */
localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
/* release the local array */
PMIX_ARGV_FREE(ndcpus);

/* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);
/* release the list */
free(localcpus);
```

C

Note that for efficiency, these two values can be computed at the same time.

The final *info* array might therefore look like the diagram in 11.5:

11.1.4 PMIx_server_deregister_nspace

Summary

Deregister a namespace.

Format

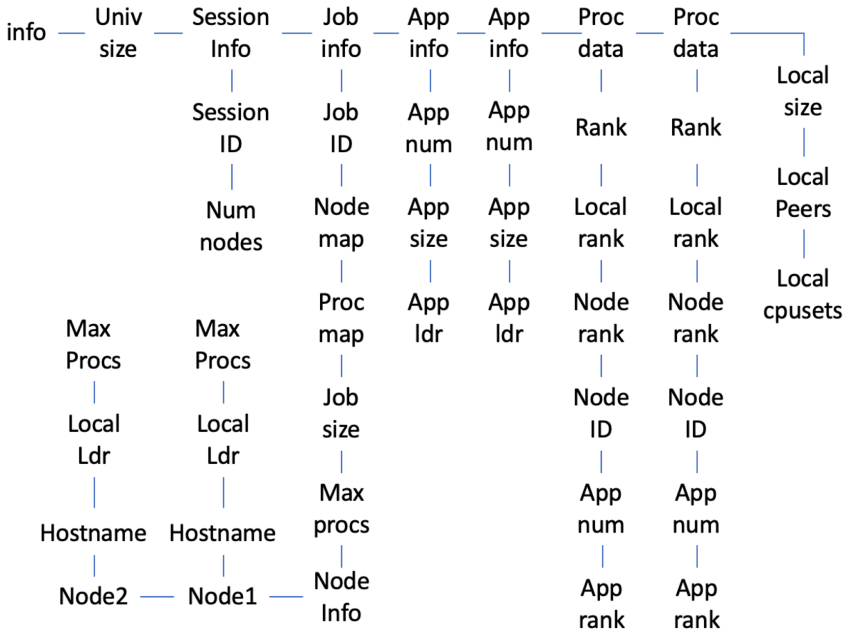

Figure 11.5.: Final information array

C

```
void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
                                   pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **`nspace`**
Namespace (string)

IN **`cbfunc`**
Callback function **`pmix_op_cbfunc_t`** (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.

11.1.5 PMIx_server_register_client

Summary

Register a client process with the PMIx server library.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_register_client(const pmix_proc_t *proc,
                            uid_t uid, gid_t gid,
                            void *server_object,
                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **proc**
pmix_proc_t structure (handle)

IN **uid**
user id (integer)

IN **gid**
group id (integer)

IN **server_object**
(memory reference)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (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

Register a client process with the PMIx server library.

The host server can also, if it desires, provide an object it wishes to be returned when a server function is called that relates to a specific process. For example, the host server may have an object that tracks the specific client. Passing the object to the library allows the library to provide that object to the host server during subsequent calls related to that client, such as a

`pmix_server_client_connected_fn_t` function. This allows the host server to access the object without performing a lookup based on the client's namespace and rank.

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process. The expected user ID and group ID of the child process allows the server library to properly authenticate clients as they connect by requiring the two values to match. Accordingly, the detected user and group ID's of the connecting process are not included in the `pmix_server_client_connected_fn_t` server module function.

Advice to PMIx library implementers

For security purposes, the PMIx server library should check the user and group ID's of a connecting process against those provided for the declared client process identifier via the `PMIx_server_register_client` prior to completing the connection.

11.1.6 PMIx_server_deregister_client

Summary

Deregister a client and purge all data relating to it.

Format

PMIx v1.0

C

```
void
PMIx_server_deregister_client(const pmix_proc_t *proc,
                              pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `proc`
`pmix_proc_t` structure (handle)

IN `cbfunc`
Callback function `pmix_op_cbfunc_t` (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.

11.1.7 PMIx_server_setup_fork

Summary

Setup the environment of a child process to be forked by the host.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_setup_fork(const pmix_proc_t *proc,
                       char ***env)
```

C

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.

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process.

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).

11.1.8 PMIx_server_dmodex_request

Summary

Define a function by which the host server can request modex data from the local PMIx server.

Format

PMIx v1.0

C

```
pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,
                                         pmix_dmodex_response_fn_t cbfunc,
                                         void *cbdata)
```

C

IN **proc**
pmix_proc_t structure (handle)

IN **cbfunc**
Callback function **pmix_dmodex_response_fn_t** (function reference)

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

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

Description

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the **PMIx_Put** and **PMIx_Commit** functions followed by a **PMIx_Fence** barrier that globally exchanges the posted information. However, the barrier operation represents a signficant time impact at large scale.

PMIx supports an alternative wireup method known as *Direct Modex* that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the **PMIx_server_dmodex_request** function, receiving the response in the provided *cbfunc* once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

Advice to users

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

11.1.9 PMIx_server_setup_application

Summary

Provide a function by which the resource manager can request application-specific setup data prior to launch of a **job** .

Format

PMIx v2.0

C

```
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)
```

C

IN **nspace**
namespace (string)

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

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

IN **cbfunc**
Callback function **pmix_setup_application_cbfunc_t** (function reference)

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

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 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:

PMIX_SETUP_APP_ENVARS **"pmix.setup.env"** (**bool**)
Harvest and include relevant environmental variables

PMIX_SETUP_APP_NONENVARS **""pmix.setup.nenv"** (**bool**)
Include all relevant data other than environmental variables

PMIX_SETUP_APP_ALL **"pmix.setup.all"** (**bool**)

Include all relevant data

PMIX_ALLOC_NETWORK **"pmix.alloc.net"** (**array**)
Array of **pmix_info_t** describing requested network resources. This must include at least: **PMIX_ALLOC_NETWORK_ID** , **PMIX_ALLOC_NETWORK_TYPE** , and **PMIX_ALLOC_NETWORK_ENDPTS** , plus whatever other descriptors are desired.

PMIX_ALLOC_NETWORK_ID **"pmix.alloc.netid"** (**char***)
The key to be used when accessing this requested network allocation. The allocation will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the network support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include: **PMIX_ALLOC_NETWORK_TYPE** - the type of resources provided; **PMIX_ALLOC_NETWORK_PLANE** - if applicable, what plane the resources were assigned from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the requested network allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

PMIX_ALLOC_NETWORK_SEC_KEY **"pmix.alloc.nsec"** (**pmix_byte_object_t**)
Network security key

PMIX_ALLOC_NETWORK_TYPE **"pmix.alloc.nettype"** (**char***)
Type of desired transport (e.g., *"tcp"*, *"udp"*)

PMIX_ALLOC_NETWORK_PLANE **"pmix.alloc.netplane"** (**char***)
ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

PMIX_ALLOC_NETWORK_ENDPTS **"pmix.alloc.endpts"** (**size_t**)
Number of endpoints to allocate per process

PMIX_ALLOC_NETWORK_ENDPTS_NODE **"pmix.alloc.endpts.nd"** (**size_t**)
Number of endpoints to allocate per node

Optional Attributes

PMIx libraries that support this operation may support the following:

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw"** (**float**)
Mbits/sec.

PMIX_ALLOC_NETWORK_QOS **"pmix.alloc.netqos"** (**char***)
Quality of service level.

PMIX_ALLOC_TIME **"pmix.alloc.time"** (**uint32_t**)

Time in seconds.

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_MODE** **"pmix.pgm.model"** (**char***)
  Programming model being initialized (e.g., "MPI" or "OpenMP")
- **PMIX_MODEL_LIBRARY_NAME** **"pmix.mdl.name"** (**char***)
  Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
- **PMIX_MODEL_LIBRARY_VERSION** **"pmix.mld.vrs"** (**char***)
  Programming model version string (e.g., "2.1.1")

Description

Provide a function by which the RM can request application-specific setup data (e.g., environmental variables, fabric configuration and security credentials) from supporting PMIx server library subsystems prior to initiating launch of a job.

Advice to PMIx server hosts

Host environments are required to execute this operation prior to launching a job. In addition to supported directives, the *info* array must include a description of the **job** using the **PMIX_NODE_MAP** and **PMIX_PROC_MAP** attributes.

This is defined as a non-blocking operation in case contributing subsystems need to perform some potentially time consuming action (e.g., query a remote service) before responding. The returned data must be distributed by the RM and subsequently delivered to the local PMIx server on each node where application processes will execute, prior to initiating execution of those processes.

Advice to PMIx library implementers

Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

11.1.10 PMIx_Register_attributes

Summary

Register host environment attribute support for a function.

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Register_attributes(char *function,
                         pmix_regattr_t attrs[],
                         size_t nattrs)
```

C

IN **function**
String name of function (string)

IN **attrs**
Array of **pmix_regattr_t** describing the supported attributes (handle)

IN **nattrs**
Number of elements in *attrs* (**size_t**)

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

Description

The **PMIx_Register_attributes** function is used by the host environment to register with its PMIx server library the attributes it supports for each **pmix_server_module_t** function. The *function* is the string name of the server module function (e.g., "register_events", "validate_credential", or "allocate") whose attributes are being registered. See the **pmix_regattr_t** entry for a description of the *attrs* array elements.

Note that the host environment can also query the library (using the **PMIx_Query_info_nb** API) for its attribute support both at the server, client, and tool levels once the host has executed **PMIx_server_init** since the server will internally register those values.

Advice to PMIx server hosts

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

PMIx implementations are *required* to register all internally supported attributes for each API during initialization of the library (i.e., when the process calls their respective PMIx init function). Specifically, the implementation *must not* register supported attributes upon first call to a given API as this would prevent users from discovering supported attributes prior to first use of an API.

It is the implementation's responsibility to associate registered attributes for a given **pmix_server_module_t** function with their corresponding user-facing API. Supported attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by the level (see 3.4.33) at which the attribute is supported.

Note that attributes can/will be registered on an API for each level. It is *required* that the implementation support user queries for supported attributes on a per-level basis. Duplicate registrations at the *same* level for a function *shall* return an error - however, duplicate registrations at *different* levels *shall* be independently tracked.

11.1.11 PMIx_server_setup_local_support

Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_server_setup_local_support(const pmix_nspace_t nspace,
                                pmix_info_t info[], size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata);
```

C

IN **nspace**
Namespace (string)

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

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (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 network 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.

11.1.12 PMIx_server_IOF_deliver

Summary

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_server_IOF_deliver(const pmix_proc_t *source,
                        pmix_iof_channel_t channel,
                        const pmix_byte_object_t *bo,
                        const pmix_info_t info[], size_t ninfo,
                        pmix_op_cbfunc_t cbfunc, void *cbdata);
```

IN **source**
Pointer to **pmix_proc_t** identifying source of the IO (handle)

IN **channel**
IO channel of the data (**pmix_iof_channel_t**)

IN **bo**
Pointer to **pmix_byte_object_t** containing the payload to be delivered (handle)

IN **info**
Array of **pmix_info_t** metadata describing the data (array of handles)

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

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.

11.1.13 PMIx_server_collect_inventory

Summary

Collect inventory of resources on a node

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_server_collect_inventory(const pmix_info_t directives[],
                              size_t ndirs,
                              pmix_info_cbfunc_t cbfunc,
                              void *cbdata);
```

C

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.

Description

Provide a function by which the host environment can request its PMIx server library collect an inventory of local resources. Supported resources depends upon the PMIx implementation, but may include the local node topology and network interfaces.

Advice to PMIx server hosts

This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested information. Inventory collection is expected to be a rare event – at system startup and upon command from a system administrator. Inventory updates are expected to initiate a smaller operation involving only the changed information. For example, replacement of a node would generate an event to notify the scheduler with an inventory update without invoking a global inventory operation.

11.1.14 PMIx_server_deliver_inventory

Summary

Pass collected inventory to the PMIx server library for storage

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_server_deliver_inventory(const pmix_info_t info[],
                              size_t ninfo,
                              const pmix_info_t directives[],
                              size_t ndirs,
                              pmix_op_cbfunc_t cbfunc,
                              void *cbdata);
```

C

IN **info**
Array of **pmix_info_t** containing the inventory (array of handles)

IN **ninfo**
Number of elements in the *info* array (**size_t**)

IN **directives**
Array of **pmix_info_t** directing the request (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (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 host environment can pass inventory information obtained from a node 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 *cbfunc* callback function will be called once the PMIx server library no longer requires access to the provided data.

11.2 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. Client calls to such functions will return a **PMIX_ERR_NOT_SUPPORTED** status.

The host RM will provide the function pointers in a **pmix_server_module_t** structure passed to **PMIx_server_init** . That module structure and associated function references are defined in this section.

Advice to PMIx server hosts

For performance purposes, the host server is required to return as quickly as possible from all functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx server support library to handle multiple client requests as quickly and scalably as possible.

All data passed to the host server functions is “owned” by the PMIX server support library and must not be free’d. Data returned by the host server via callback function is owned by the host server, which is free to release it upon return from the callback

11.2.1 pmix_server_module_t Module

Summary

List of function pointers that a PMIx server passes to **PMIx_server_init** during startup.

Format

C

```
typedef struct pmix_server_module_3_0_0_t
    /* v1x interfaces */
    pmix_server_client_connected_fn_t   client_connected;
    pmix_server_client_finalized_fn_t   client_finalized;
    pmix_server_abort_fn_t              abort;
    pmix_server_fencenb_fn_t            fence_nb;
    pmix_server_dmodex_req_fn_t         direct_modex;
    pmix_server_publish_fn_t            publish;
    pmix_server_lookup_fn_t             lookup;
    pmix_server_unpublish_fn_t          unpublish;
    pmix_server_spawn_fn_t              spawn;
    pmix_server_connect_fn_t            connect;
    pmix_server_disconnect_fn_t         disconnect;
    pmix_server_register_events_fn_t    register_events;
    pmix_server_deregister_events_fn_t  deregister_events;
    pmix_server_listener_fn_t           listener;
    /* v2x interfaces */
    pmix_server_notify_event_fn_t       notify_event;
    pmix_server_query_fn_t              query;
    pmix_server_tool_connection_fn_t    tool_connected;
    pmix_server_log_fn_t                log;
    pmix_server_alloc_fn_t              allocate;
    pmix_server_job_control_fn_t        job_control;
    pmix_server_monitor_fn_t            monitor;
    /* v3x interfaces */
    pmix_server_get_cred_fn_t           get_credential;
    pmix_server_validate_cred_fn_t      validate_credential;
    pmix_server_iof_fn_t                iof_pull;
    pmix_server_stdin_fn_t              push_stdin;
    /* v4x interfaces */
    pmix_server_grp_fn_t                group;
 pmix_server_module_t;
```

C

11.2.2 pmix_server_client_connected_fn_t

Summary

Notify the host server that a client connected to this server.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_client_connected_fn_t)(
                                const pmix_proc_t *proc,
                                void* server_object,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure (handle)

IN **server_object**
object reference (memory reference)

IN **cbfunc**
Callback function pmix_op_cbfunc_t (function reference)

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

Returns one of the following:

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

Description

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. If provided, an implementation of pmix_server_client_connected_fn_t is only required to call the callback function designated. A host server can choose to not be notified when clients connect by setting pmix_server_client_connected_fn_t to **NULL**.

It is possible that only a subset of the clients in a namespace call PMIx_Init . The server's pmix_server_client_connected_fn_t implementation should not depend on being called once per rank in a namespace or delay calling the callback function until all ranks have connected. However, if a rank makes any PMIx calls, it must first call PMIx_Init and therefore the server's pmix_server_client_connected_fn_t will be called before any other server functions specific to the rank.

Advice to PMIx server hosts

This operation is an opportunity for a host environment to update the status of the ranks it manages. It is also a convenient and well defined time to perform initialization necessary to support further calls into the server related to that rank.

11.2.3 pmix_server_client_finalized_fn_t

Summary

Notify the host environment that a client called **PMIx_Finalize** .

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_client_finalized_fn_t)(
                                const pmix_proc_t *proc,
                                void* server_object,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure (handle)

IN **server_object**
object reference (memory reference)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

Description

Notify the host environment that a client called **PMIx_Finalize** . Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server_object parameter will be the value of the server_object parameter passed to **PMIx_server_register_client** by the host server when registering the connecting client. If provided, an implementation of **pmix_server_client_finalized_fn_t** is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting **pmix_server_client_finalized_fn_t** to **NULL**.

Note that the host server is only being informed that the client has called **PMIx_Finalize** . The client might not have exited. If a client exits without calling **PMIx_Finalize** , the server support library will not call the **pmix_server_client_finalized_fn_t** implementation.

Advice to PMIx server hosts

This operation is an opportunity for a host server to update the status of the tasks it manages. It is also a convenient and well defined time to release resources used to support that client.

11.2.4 pmix_server_abort_fn_t

Summary

Notify the host environment that a local client called **PMIx_Abort** .

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_abort_fn_t)(
                                const pmix_proc_t *proc,
                                void *server_object,
                                int status,
                                const char msg[],
                                pmix_proc_t procs[],
                                size_t nprocs,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

IN **proc**
pmix_proc_t structure identifying the process requesting the abort (handle)

IN **server_object**
object reference (memory reference)

IN **status**
exit status (integer)

IN **msg**
exit status message (string)

IN **procs**
Array of **pmix_proc_t** structures identifying the processes to be terminated (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

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** indicates that all processes in the client's namespace are to be terminated.

11.2.5 pmix_server_fencenb_fn_t

Summary

At least one client called either **PMIx_Fence** or **PMIx_Fence_nb** .

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_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)
```

C

IN **procs**
Array of **pmix_proc_t** structures identifying operation participants(array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

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

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

IN **data**
(string)

IN **ndata**
(integer)

IN **cbfunc**
Callback function **pmix_modex_cbfunc_t** (function reference)

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

Returns one of the following:

- **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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

The following attributes are required to be supported by all host environments:

PMIX_COLLECT_DATA **"pmix.collect"** (**bool**)
Collect data and return it at the end of the operation.

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
If **true**, indicates that the requested choice of algorithm is mandatory.

Advice to PMIx server hosts

Host environment are required to return **PMIX_ERR_NOT_SUPPORTED** if passed an attributed marked as **PMIX_INFO_REQD** that they do not support, even if support for that attribute is optional.

Description

All local clients in the provided array of *procs* called either **PMIx_Fence** or **PMIx_Fence_nb**. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the **PMIx_Fence** / **PMIx_Fence_nb** operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix_server_fencenb_fn_t** function.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

C

```
uint8_t *blob1, *blob2, *total;
size_t sz_blob1, sz_blob2, sz_total;

sz_total = sz_blob1 + sz_blob2;
total = (uint8_t*)malloc(sz_total);
memcpy(total, blob1, sz_blob1);
memcpy(&total[sz_blob1], blob2, sz_blob2);
```

C

Note that the ordering of the data blobs does not matter.

The provided data is to be collectively shared with all PMIx servers involved in the fence operation, and returned in the modex *cbfunc*. A **NULL** data value indicates that the local processes had no data to contribute.

The array of *info* structs is used to pass user-requested options to the server. This can include directives as to the algorithm to be used to execute the fence operation. The directives are optional unless the **PMIX_INFO_REQD** flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

11.2.6 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 proc to obtain and return a direct modex blob for that proc.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
                                const pmix_proc_t *proc,
                                const pmix_info_t info[],
                                size_t ninfo,
                                pmix_modex_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure identifying the process whose data is being requested (handle)

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

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

IN **cbfunc**
Callback function pmix_modex_cbfunc_t (function reference)

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

Returns one of the following:

- PMIX_SUCCESS , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- 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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to **PMIx_Put** and **PMIx_Commit** .

The array of *info* structs is used to pass user-requested options to the server. This can include a timeout to preclude an indefinite wait for data that may never become available. The directives are optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

11.2.7 pmix_server_publish_fn_t

Summary

Publish data per the PMIx API specification.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_publish_fn_t)(
                                const pmix_proc_t *proc,
                                const pmix_info_t info[],
                                size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure of the process publishing the data (handle)

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

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

`PMIX_USERID` **`"pmix.euid"`** (**`uint32_t`**)
Effective user id.

`PMIX_GRPID` **`"pmix.egid"`** (**`uint32_t`**)
Effective group id.

Host environments that implement this entry point are required to support the following attributes:

`PMIX_RANGE` **`"pmix.range"`** (**`pmix_data_range_t`**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

`PMIX_PERSISTENCE` **`"pmix.persist"`** (**`pmix_persistence_t`**)
Value for calls to **`PMIx_Publish`** .

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Publish data per the **PMIx_Publish** specification. The callback is to be executed upon completion of the operation. The default data range is left to the host environment, but expected to be **PMIX_RANGE_SESSION** , and the default persistence **PMIX_PERSIST_SESSION** or their equivalent. These values can be specified by including the respective attributed in the *info* array.

The persistence indicates how long the server should retain the data.

Advice to PMIx server hosts

The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range. However, the server must return an error (a) if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of published info by the original publisher - it is left to the discretion of the host environment to allow info-key-based flags to modify this behavior.

The **PMIX_USERID** and **PMIX_GRPID** of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

11.2.8 pmix_server_lookup_fn_t

Summary

Lookup published data.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_lookup_fn_t)(
                                const pmix_proc_t *proc,
                                char **keys,
                                const pmix_info_t info[],
                                size_t ninfo,
                                pmix_lookup_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure of the process seeking the data (handle)
IN **keys**
(array of strings)
IN **info**
Array of info structures (array of handles)
IN **ninfo**
Number of elements in the *info* array (integer)
IN **cbfunc**
Callback function **pmix_lookup_cbfunc_t** (function reference)
IN **cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments that implement this entry point are required to support the following attributes:

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

PMIX_WAIT **"pmix.wait"** (**int**)
Caller requests that the PMIx server wait until at least the specified number of values are found (*0* indicates all and is the default).

Optional Attributes

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

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Lookup published data. The host server will be passed a **NULL**-terminated array of string keys identifying the data being requested.

The array of *info* structs is used to pass user-requested options to the server. The default data range is left to the host environment, but expected to be **PMIX_RANGE_SESSION** . This can include a wait flag to indicate that the server should wait for all data to become available before executing the callback function, or should immediately callback with whatever data is available. In addition, a timeout can be specified on the wait to preclude an indefinite wait for data that may never be published.

Advice to PMIx server hosts

The **PMIX_USERID** and **PMIX_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

11.2.9 pmix_server_unpublish_fn_t

Summary

Delete data from the data store.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_unpublish_fn_t)(
                                const pmix_proc_t *proc,
                                char **keys,
                                const pmix_info_t info[],
                                size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure identifying the process making the request (handle)

IN **keys**
(array of strings)

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

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments that implement this entry point are required to support the following attributes:

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
: Value for calls to publish/lookup/unpublish or for monitoring event notifications.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Delete data from the data store. The host server will be passed a **NULL**-terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The default data range is left to the host environment, but expected to be **PMIX_RANGE_SESSION** . The callback is to be executed upon completion of the delete procedure.

Advice to PMIx server hosts

The **PMIX_USERID** and **PMIX_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range.

11.2.10 pmix_server_spawn_fn_t

Summary

Spawn a set of applications/processes as per the **PMIx_Spawn** API.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_spawn_fn_t)(
                                const pmix_proc_t *proc,
                                const pmix_info_t job_info[],
                                size_t ninfo,
                                const pmix_app_t apps[],
                                size_t napps,
                                pmix_spawn_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proc**
pmix_proc_t structure of the process making the request (handle)

IN **job_info**
Array of info structures (array of handles)

IN **ninfo**
Number of elements in the *jobinfo* array (integer)

IN **apps**
Array of **pmix_app_t** structures (array of handles)

IN **napps**
Number of elements in the *apps* array (integer)

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 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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group id.

Host environments that provide this module entry point are required to pass the **PMIX_SPAWNED** and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX_WDIR "pmix.wdir" (char*)
Working directory for spawned processes.

PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX "pmix.prefix" (char*)
Prefix to use for starting spawned processes.

PMIX_HOST "pmix.host" (char*)
Comma-delimited list of hosts to use for spawned processes.

PMIX_HOSTFILE "pmix.hostfile" (char*)
Hostfile to use for spawned processes.

Optional Attributes

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

PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)
Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST "pmix.addhost" (char*)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
Preload binaries onto nodes.

PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
Comma-delimited list of files to pre-position on nodes.

PMIX_PERSONALITY "pmix.pers" (char*)
Name of personality to use.

PMIX_MAPPER "pmix.mapper" (char*)

Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP **"pmix.dispmap"** (**bool**)
Display process mapping upon spawn.

PMIX_PPR **"pmix.ppr"** (**char***)
Number of processes to spawn on each identified resource.

PMIX_MAPBY **"pmix.mapby"** (**char***)
Process mapping policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY **"pmix.rankby"** (**char***)
Process ranking policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO **"pmix.bindto"** (**char***)
Process binding policy - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI **"pmix.nonpmi"** (**bool**)
Spawned processes will not call **PMIx_Init** .

PMIX_STDIN_TGT **"pmix.stdin"** (**uint32_t**)
Spawned process rank that is to receive **stdin**.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
Forward this process's **stdin** to the designated process.

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
Forward **stderr** from spawned processes to this process.

PMIX_DEBUGGER_DAEMONS **"pmix.debugger"** (**bool**)
Spawned application consists of debugger daemons.

PMIX_TAG_OUTPUT **"pmix.tagout"** (**bool**)
Tag application output with the identity of the source process.

PMIX_TIMESTAMP_OUTPUT **"pmix.tsout"** (**bool**)
Timestamp output from applications.

PMIX_MERGE_STDERR_STDOUT **"pmix.mergeerrout"** (**bool**)

Merge **stdout** and **stderr** streams from application processes.

PMIX_OUTPUT_TO_FILE **"pmix.outfile"** (**char***)
Output application output to the specified file.

PMIX_INDEX_ARGV **"pmix.indxargv"** (**bool**)
Mark the **argv** with the rank of the process.

PMIX_CPUS_PER_PROC **"pmix.cpuperproc"** (**uint32_t**)
Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

PMIX_NO_PROCS_ON_HEAD **"pmix.nolocal"** (**bool**)
Do not place processes on the head node.

PMIX_NO_OVERSUBSCRIBE **"pmix.noover"** (**bool**)
Do not oversubscribe the cpus.

PMIX_REPORT_BINDINGS **"pmix.repbind"** (**bool**)
Report bindings of the individual processes.

PMIX_CPU_LIST **"pmix.cpulist"** (**char***)
List of cpus to use for this job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

PMIX_JOB_RECOVERABLE **"pmix.recover"** (**bool**)
Application supports recoverable operations.

PMIX_JOB_CONTINUOUS **"pmix.continuous"** (**bool**)
Application is continuous, all failed processes should be immediately restarted.

PMIX_MAX_RESTARTS **"pmix.maxrestarts"** (**uint32_t**)
Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided namespace

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Spawn a set of applications/processes as per the **PMIx_Spawn** API. Note that applications are not required to be MPI or any other programming model. Thus, the host server cannot make any assumptions as to their required support. The callback function is to be executed once all processes have been started. An error in starting any application or process in this request shall cause all applications and processes in the request to be terminated, and an error returned to the originating caller.

Note that a timeout can be specified in the job_info array to indicate that failure to start the requested job within the given time should result in termination to avoid hangs.

11.2.11 pmix_server_connect_fn_t

Summary

Record the specified processes as *connected*.

Format

PMIx v1.0

C

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

C

IN **procs**
Array of **pmix_proc_t** structures identifying participants (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

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

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- 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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

PMIX_COLLECTIVE_ALGO **"pmix.calgo"** (**char***)
: Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.

PMIX_COLLECTIVE_ALGO_REQD **"pmix.calreqd"** (**bool**)
: If **true**, indicates that the requested choice of algorithm is mandatory.

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.

11.2.12 pmix_server_disconnect_fn_t

Summary

Disconnect a previously connected set of processes.

Format

PMIx v1.0

C

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

C

IN **procs**
Array of **pmix_proc_t** structures identifying participants (array of handles)

IN **nprocs**
Number of elements in the *procs* array (integer)

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

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- 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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Description

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server's has called the **pmix_server_disconnect_fn_t** function, and the host environment has completed any required supporting operations.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

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

A **PMIX_ERR_INVALID_OPERATION** error must be returned if the specified set of *procs* was not previously *connected* via a call to the **pmix_server_connect_fn_t** function.

11.2.13 pmix_server_register_events_fn_t

Summary

Register to receive notifications for the specified events.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_register_events_fn_t)(
                                pmix_status_t *codes,
                                size_t ncodes,
                                const pmix_info_t info[],
                                size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **codes**
Array of **pmix_status_t** values (array of handles)

IN **ncodes**
Number of elements in the *codes* array (integer)

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

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Description

Register to receive notifications for the specified status codes. The *info* array included in this API is reserved for possible future directives to further steer notification.

Advice to PMIx library implementers

The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:

- the client has requested notification of an environmental code (i.e., a PMIx code in the range beyond **PMIX_ERR_SYS_OTHER**) or a code 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

Advice to PMIx server hosts

The host environment is required to pass to its PMIx server library all non-environmental events that directly relate to a registered namespace without the PMIx server library explicitly requesting them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in the range between **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER** (inclusive).

11.2.14 pmix_server_deregister_events_fn_t

Summary

Deregister to receive notifications for the specified events.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(
                                pmix_status_t *codes,
                                size_t ncodes,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **codes**
Array of **pmix_status_t** values (array of handles)

IN **ncodes**
Number of elements in the *codes* array (integer)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

Description

Deregister to receive notifications for the specified events to which the PMIx server has previously registered.

Advice to PMIx library implementers

The PMIx server library must track all client registrations. This module function shall only be called when:

- the library is deregistering environmental codes (i.e., a PMIx codes in the range between **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER** , inclusive) or codes that lies outside the defined PMIx range of constants; and
- no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it remain.

11.2.15 pmix_server_notify_event_fn_t

Summary

Notify the specified processes of an event.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_notify_event_fn_t)(pmix_status_t code,
                                const pmix_proc_t *source,
                                pmix_data_range_t range,
                                pmix_info_t info[],
                                size_t ninfo,
                                pmix_op_cbfunc_t cbfunc,
                                void *cbdata);
```

C

IN **code**
The **pmix_status_t** event code being referenced structure (handle)

IN **source**
pmix_proc_t of process that generated the event (handle)

IN **range**
pmix_data_range_t range over which the event is to be distributed (handle)

IN **info**
Optional array of **pmix_info_t** structures containing additional information on the event (array of handles)

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

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- **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 required to pass any provided attributes to the host environment for processing.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

Description

Notify the specified processes (described through a combination of *range* and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. The process generating the event is provided in the *source* parameter, and any further descriptive information is included in the *info* array.

Advice to PMIx server hosts

The callback function is to be executed once the host environment no longer requires that the PMIx server library maintain the provided data structures. It does not necessarily indicate that the event has been delivered to any process, nor that the event has been distributed for delivery

11.2.16 pmix_server_listener_fn_t

Summary

Register a socket the host server can monitor for connection requests.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_listener_fn_t)(
                                int listening_sd,
                                pmix_connection_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **incoming_sd**
(integer)

IN **cbfunc**
Callback function **pmix_connection_cbfunc_t** (function reference)

IN **cbdata**
(memory reference)

Returns **PMIX_SUCCESS** indicating that the request is accepted, or a negative value corresponding to a PMIx error constant indicating that the request has been rejected.

Description

Register a socket the host environment can monitor for connection requests, harvest them, and then call the PMIx server library's internal callback function for further processing. A listener thread is essential to efficiently harvesting connection requests from large numbers of local clients such as occur when running on large SMPs. The host server listener is required to call accept on the incoming connection request, and then pass the resulting socket to the provided cbfunc. A **NULL** for this function will cause the internal PMIx server to spawn its own listener thread.

11.2.17 pmix_server_query_fn_t

Summary

Query information from the resource manager.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_query_fn_t)(
                                pmix_proc_t *proct,
                                pmix_query_t *queries, size_t nqueries,
                                pmix_info_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **proct**
pmix_proc_t structure of the requesting process (handle)

IN **queries**
Array of pmix_query_t 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 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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Optional Attributes

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

PMIX_QUERY_NAMESPACES **"pmix.qry.ns"** (**char*****)**
Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS **"pmix.qry.jst"** (**pmix_status_t**)
Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST **"pmix.qry.qlst"** (**char*****)**
Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS **"pmix.qry.qst"** (**TBD**)
Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE **"pmix.qry.ptable"** (**char*****)**
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE **"pmix.qry.lptable"** (**char*****)**
Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same node.

PMIX_QUERY_SPAWN_SUPPORT **"pmix.qry.spawn"** (**bool**)
Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT **"pmix.qry.debug"** (**bool**)
Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE **"pmix.qry.mem"** (**bool**)
Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_LOCAL_ONLY **"pmix.qry.local"** (**bool**)
Constrain the query to local information only.

PMIX_QUERY_REPORT_AVG **"pmix.qry.avg"** (**bool**)
Report only average values for sampled information.

PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (**bool**)
Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (**char***)
String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING "pmix.time.remaining" (**char***)
Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

Description

Query information from the host environment. The query will include the namespace/rank of the process that is requesting the info, an array of **pmix_query_t** describing the request, and a callback function/data for the return.

Advice to PMIx library implementers

The PMIx server library should not block in this function as the host environment may, depending upon the information being requested, require significant time to respond.

11.2.18 pmix_server_tool_connection_fn_t

Summary

Register that a tool has connected to the server.

Format

PMIx v2.0

C

```
typedef void (*pmix_server_tool_connection_fn_t)(
                                pmix_info_t info[], size_t ninfo,
                                pmix_tool_connection_cbfunc_t cbfunc,
                                void *cbdata)
```

C

IN **info**
Array of **pmix_info_t** structures (array of handles)

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

IN **cbfunc**
Callback function **pmix_tool_connection_cbfunc_t** (function reference)

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

Required Attributes

PMIx libraries are required to pass the following attributes in the *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
: Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
: Effective group id.

Optional Attributes

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

PMIX_FWD_STDOUT **"pmix.fwd.stdout"** (**bool**)
: Forward **stdout** from spawned processes to this process.

PMIX_FWD_STDERR **"pmix.fwd.stderr"** (**bool**)
: Forward **stderr** from spawned processes to this process.

PMIX_FWD_STDIN **"pmix.fwd.stdin"** (**bool**)
: Forward this process's **stdin** to the designated process.

Description

Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The **pmix_info_t** array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

Advice to PMIx server hosts

The host environment is solely responsible for authenticating and authorizing the connection, and for authorizing all subsequent tool requests. The host must not execute the callback function prior to returning from the API.

11.2.19 pmix_server_log_fn_t

Summary

Log data on behalf of a client.

Format

PMIx v2.0

C

```
typedef void (*pmix_server_log_fn_t)(
                                const pmix_proc_t *client,
                                const pmix_info_t data[], size_t ndata,
                                const pmix_info_t directives[], size_t ndirs,
                                pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **client**
pmix_proc_t structure (handle)

IN **data**
Array of info structures (array of handles)

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

IN **directives**
Array of info structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (integer)

IN **cbfunc**
Callback function pmix_op_cbfunc_t (function reference)

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)
Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_LOG_STDERR "pmix.log.stderr" (char*)
Log string to **stderr**.

PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
Log string to **stdout**.

PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)

Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog

Optional Attributes

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

PMIX_LOG_MSG **"pmix.log.msg"** (**pmix_byte_object_t**)
Message blob to be sent somewhere.

PMIX_LOG_EMAIL **"pmix.log.email"** (**pmix_data_array_t**)
Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR **"pmix.log.emaddr"** (**char***)
Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SUBJECT **"pmix.log.emsub"** (**char***)
Subject line for email.

PMIX_LOG_EMAIL_MSG **"pmix.log.emmsg"** (**char***)
Message to be included in email.

Description

Log data on behalf of a client. This function is not intended for output of computational results, but rather for reporting status and error messages. The host must not execute the callback function prior to returning from the API.

11.2.20 pmix_server_alloc_fn_t

Summary

Request allocation operations on behalf of a client.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_alloc_fn_t)(
                                const pmix_proc_t *client,
                                pmix_alloc_directive_t directive,
                                const pmix_info_t data[], size_t ndata,
                                pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **client**
pmix_proc_t structure of process making request (handle)

IN **directive**
Specific action being requested (**pmix_alloc_directive_t**)

IN **data**
Array of info structures (array of handles)

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

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_ALLOC_ID "**pmix.alloc.id**" (**char***)
A string 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** .

PMIX_ALLOC_NUM_NODES "**pmix.alloc.nnodes**" (**uint64_t**)
The number of nodes.

PMIX_ALLOC_NUM_CPUS "**pmix.alloc.ncpus**" (**uint64_t**)
Number of cpus.

PMIX_ALLOC_TIME "**pmix.alloc.time**" (**uint32_t**)
Time in seconds.

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.

PMIX_ALLOC_NUM_CPU_LIST "**pmix.alloc.ncpulist**" (**char***)
Regular expression of the number of cpus for each node.

PMIX_ALLOC_CPU_LIST "**pmix.alloc.cpulist**" (**char***)
Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE "**pmix.alloc.msize**" (**float**)
Number of Megabytes.

PMIX_ALLOC_NETWORK "**pmix.alloc.net**" (**array**)
Array of **pmix_info_t** describing requested network resources. This must include at least: **PMIX_ALLOC_NETWORK_ID** , **PMIX_ALLOC_NETWORK_TYPE** , and **PMIX_ALLOC_NETWORK_ENDPTS** , plus whatever other descriptors are desired.

PMIX_ALLOC_NETWORK_ID "**pmix.alloc.netid**" (**char***)
The key to be used when accessing this requested network allocation. The allocation will be returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the network support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as *"32000-32100,33005,38123-38146"*. Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include:
PMIX_ALLOC_NETWORK_TYPE - the type of resources provided;
PMIX_ALLOC_NETWORK_PLANE - if applicable, what plane the resources were assigned

from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the requested network allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

PMIX_ALLOC_BANDWIDTH **"pmix.alloc.bw"** (**float**)
Mbits/sec.

PMIX_ALLOC_NETWORK_QOS **"pmix.alloc.netqos"** (**char***)
Quality of service level.

Description

Request new allocation or modifications to an existing allocation on behalf of a client. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute for an existing allocation. Any additional allocated resources will be considered as part of the current 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 affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities.
- Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to the scheduler with a promise to return them upon subsequent request.

The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

11.2.21 pmix_server_job_control_fn_t

Summary

Execute a job control action on behalf of a client.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_job_control_fn_t)(
                                const pmix_proc_t *requestor,
                                const pmix_proc_t targets[], size_t ntargets,
                                const pmix_info_t directives[], size_t ndirs,
                                pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **requestor**
pmix_proc_t structure of requesting process (handle)

IN **targets**
Array of proc structures (array of handles)

IN **ntargets**
Number of elements in the *targets* array (integer)

IN **directives**
Array of info structures (array of handles)

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

IN **cbfunc**
Callback function pmix_op_cbfunc_t (function reference)

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

Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any attributes provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_JOB_CTRL_ID **"pmix.jctrl.id"** (**char***)
Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

PMIX_JOB_CTRL_PAUSE **"pmix.jctrl.pause"** (**bool**)
Pause the specified processes.

PMIX_JOB_CTRL_RESUME **"pmix.jctrl.resume"** (**bool**)
Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_KILL **"pmix.jctrl.kill"** (**bool**)
Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_SIGNAL **"pmix.jctrl.sig"** (**int**)
Send given signal to specified processes.

PMIX_JOB_CTRL_TERMINATE **"pmix.jctrl.term"** (**bool**)
Politely terminate the specified processes.

Optional Attributes

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

PMIX_JOB_CTRL_CANCEL **"pmix.jctrl.cancel"** (**char***)
Cancel the specified request - the provided request ID must match the **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of **NULL** implies cancel all requests from this requestor.

PMIX_JOB_CTRL_RESTART **"pmix.jctrl.restart"** (**char***)
Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT **"pmix.jctrl.ckpt"** (**char***)
Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT **"pmix.jctrl.ckptev"** (**bool**)
Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL **"pmix.jctrl.ckptsig"** (**int**)
Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT **"pmix.jctrl.ckptsig"** (**int**)
Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (**pmix_data_array_t**)
Array of **pmix_info_t** declaring each method and value supported by this application.

PMIX_JOB_CTRL_PROVISION **"pmix.jctrl.pvn"** (**char***)
Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE **"pmix.jctrl.pvnimg"** (**char***)
Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE **"pmix.jctrl.preempt"** (**bool**)
Indicate that the job can be pre-empted.

Description

Execute a job control action on behalf of a client. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX_RANK_WILDCARD** can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix_info_t** structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix_info_cbfunc_t** array of **pmix_info_t** structures.

11.2.22 pmix_server_monitor_fn_t

Summary

Request that a client be monitored for activity.

Format

PMIx v2.0

C

```
typedef pmix_status_t (*pmix_server_monitor_fn_t)(
                                const pmix_proc_t *requestor,
                                const pmix_info_t *monitor, pmix_status_t error
                                const pmix_info_t directives[], size_t ndirs,
                                pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

IN **requestor**
pmix_proc_t structure of requesting process (handle)

IN **monitor**
pmix_info_t identifying the type of monitor being requested (handle)

IN **error**
Status code to use in generating event if alarm triggers (integer)

IN **`directives`**
Array of info structures (array of handles)

IN **`ndirs`**
Number of elements in the *info* array (integer)

IN **`cbfunc`**
Callback function **`pmix_op_cbfunc_t`** (function reference)

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

Returns one of the following:

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

This entry point is only called for monitoring requests that are not directly supported by the PMIx 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:

`PMIX_USERID` **`"pmix.euid"`** (**`uint32_t`**)
Effective user id.

`PMIX_GRPID` **`"pmix.egid"`** (**`uint32_t`**)
Effective group id.

Host environments are not required to support any specific monitoring attributes.

Optional Attributes

The following attributes may be implemented by a host environment.

PMIX_MONITOR_ID **"pmix.monitor.id"** (**char***)
Provide a string identifier for this request.

PMIX_MONITOR_CANCEL **"pmix.monitor.cancel"** (**char***)
Identifier to be canceled (**NULL** means cancel all monitoring for this process).

PMIX_MONITOR_APP_CONTROL **"pmix.monitor.appctrl"** (**bool**)
The application desires to control the response to a monitoring event.

PMIX_MONITOR_HEARTBEAT **"pmix.monitor.mbeat"** (**void**)
Register to have the PMIx server monitor the requestor for heartbeats.

PMIX_MONITOR_HEARTBEAT_TIME **"pmix.monitor.btime"** (**uint32_t**)
Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS **"pmix.monitor.bdrop"** (**uint32_t**)
Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE **"pmix.monitor.fmon"** (**char***)
Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE **"pmix.monitor.fsize"** (**bool**)
Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS **"pmix.monitor.faccess"** (**char***)
Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY **"pmix.monitor.fmod"** (**char***)
Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME **"pmix.monitor.ftime"** (**uint32_t**)
Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS **"pmix.monitor.fdrop"** (**uint32_t**)
Number of file checks that can be missed before generating the event.

Description

Request that a client be monitored for activity.

Advice to PMIx server hosts

If this module entry is provided and called by the PMIx server library, then the host environment must either provide the requested services or return **PMIX_ERR_NOT_SUPPORTED** to the provided *cbfunc*.

11.2.23 pmix_server_get_cred_fn_t

Summary

Request a credential from the host environment

Format

PMIx v3.0

C

```
typedef pmix_status_t (*pmix_server_get_cred_fn_t)(
                                const pmix_proc_t *proc,
                                const pmix_info_t directives[],
                                size_t ndirs,
                                pmix_credential_cbfunc_t cbfunc,
                                void *cbdata);
```

C

IN **proc**
pmix_proc_t structure of requesting process (handle)

IN **directives**
Array of info structures (array of handles)

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

IN **cbfunc**
Callback function to return the credential (pmix_credential_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.

Required Attributes

If the PMIx library does not itself provide the requested credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)
Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group id.

Optional Attributes

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

PMIX_CRED_TYPE "pmix.sec.ctype" (char*)
When passed in **PMIx_Get_credential** , a prioritized, comma-delimited list of desired credential types for use in environments where multiple authentication mechanisms may be available. When returned in a callback function, a string identifier of the credential type.

PMIX_TIMEOUT "pmix.timeout" (int)
Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Request a credential from the host environment

Advice to PMIx server hosts

If this module entry is provided and called by the PMIx server library, then the host environment must either provide the requested credential in the callback function or immediately return an error to the caller.

11.2.24 pmix_server_validate_cred_fn_t

Summary

Request validation of a credential

Format

PMIx v3.0

C

```
typedef pmix_status_t (*pmix_server_validate_cred_fn_t)(
                                const pmix_proc_t *proc,
                                const pmix_byte_object_t *cred,
                                const pmix_info_t directives[],
                                size_t ndirs,
                                pmix_validation_cbfunc_t cbfunc,
                                void *cbdata);
```

C

IN **proc**
pmix_proc_t structure of requesting process (handle)

IN **cred**
Pointer to **pmix_byte_object_t** containing the credential (handle)

IN **directives**
Array of info structures (array of handles)

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

IN **cbfunc**
Callback function to return the result (**pmix_validation_cbfunc_t** function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

Required Attributes

If the PMIx library does not itself validate the credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments are not required to support any specific attributes.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Request validation of a credential obtained from the host environment via a prior call to the **pmix_server_get_cred_fn_t** module entry.

11.2.25 pmix_server_iof_fn_t

Summary

Request the specified IO channels be forwarded from the given array of processes.

Format

PMIx v3.0

C

```
typedef pmix_status_t (*pmix_server_iof_fn_t)(
                          const pmix_proc_t procs[], size_t nprocs,
                          const pmix_info_t directives[], size_t ndirs,
                          pmix_iof_channel_t channels,
                          pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

IN **procs**
Array **pmix_proc_t** identifiers whose IO is being requested (handle)

IN **nprocs**
Number of elements in *procs* (**size_t**)

IN **directives**
Array of **pmix_info_t** structures further defining the request (array of handles)

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

IN **channels**
Bitmask identifying the channels to be forwarded (**pmix_iof_channel_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

Required Attributes

The following attributes are required to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_IOF_CACHE_SIZE **"pmix.iof.csize"** (**uint32_t**)
: The requested size of the 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, drop the oldest bytes to make room in the cache.

PMIX_IOF_DROP_NEWEST **"pmix.iof.new"** (**bool**)
: In an overflow situation, drop any new bytes received until room becomes available in the cache (default).

Optional Attributes

The following attributes may be supported by a host environment.

PMIX_IOF_BUFFERING_SIZE **"pmix.iof.bsize"** (**uint32_t**)
: Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of IO arrives. The library will execute the callback whenever the specified number of bytes becomes available. Any remaining buffered data will be "flushed" upon 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.

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.

11.2.26 pmix_server_stdin_fn_t

Summary

Pass standard input data to the host environment for transmission to specified recipients.

Format

PMIx v3.0

C

```
typedef pmix_status_t (*pmix_server_stdin_fn_t)(
                                const pmix_proc_t *source,
                                const pmix_proc_t targets[],
                                size_t ntargets,
                                const pmix_info_t directives[],
                                size_t ndirs,
                                const pmix_byte_object_t *bo,
                                pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

IN **source**
pmix_proc_t structure of source process (handle)

IN **targets**
Array of **pmix_proc_t** target identifiers (handle)

IN **ntargets**
Number of elements in the *targets* array (integer)

IN **directives**
Array of info structures (array of handles)

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

IN **bo**
Pointer to **pmix_byte_object_t** containing the payload (handle)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

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 to be included in the passed *info* array:

PMIX_USERID **"pmix.euid"** (**uint32_t**)
Effective user id.

PMIX_GRPID **"pmix.egid"** (**uint32_t**)
Effective group id.

Description

Passes stdin to the host environment for transmission to specified recipients. The host environment is responsible for forwarding the data to all locations that host the specified *targets* and delivering the payload to the PMIx server library connected to those clients.

Advice to PMIx server hosts

If this module entry is provided and called by the PMIx server library, then the host environment must either provide the requested services or return **PMIX_ERR_NOT_SUPPORTED** to the provided *cbfunc*.

11.2.27 pmix_server_grp_fn_t

Summary

Request group operations (construct, destruct, etc.) on behalf of a set of processes.

Format

PMIx v4.0

C

```
typedef pmix_status_t (*pmix_server_grp_fn_t)(
                                pmix_group_operation_t op, char grp[],
                                const pmix_proc_t procs[], size_t nprocs,
                                const pmix_info_t directives[],
                                size_t ndirs,
                                pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

IN **op**
pmix_group_operation_t value indicating operation the host is requested to perform (integer)

IN **grp**
Character string identifying the group (string)

IN **procs**
Array of **pmix_proc_t** identifiers of participants (handle)

IN **nprocs**
Number of elements in the *procs* array (integer)

IN **directives**
Array of info structures (array of handles)

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

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Returns one of the following:

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

Optional Attributes

The following attributes may be supported by a host environment.

PMIX_GROUP_ASSIGN_CONTEXT_ID **"pmix.grp.actxid"** (**bool**)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

PMIX_GROUP_LOCAL_ONLY **"pmix.grp.lcl"** (**bool**)
Group operation only involves local processes. PMIx implementations are *required* to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false

PMIX_GROUP_ENDPT_DATA **"pmix.grp.endpt"** (**pmix_byte_object_t**)
Data collected to be shared during group construction

PMIX_GROUP_OPTIONAL **"pmix.grp.opt"** (**bool**)
Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false

PMIX_RANGE **"pmix.range"** (**pmix_data_range_t**)
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

The following attributes may be included in the host's response:

PMIX_GROUP_ID **"pmix.grp.id"** (**char***)
User-provided group identifier

PMIX_GROUP_MEMBERSHIP **"pmix.grp.mbrs"** (**pmix_data_array_t***)
Array of group member ID's

PMIX_GROUP_CONTEXT_ID **"pmix.grp.ctxid"** (**size_t**)
Context identifier assigned to the group by the host RM.

PMIX_GROUP_ENDPT_DATA **"pmix.grp.endpt"** (**pmix_byte_object_t**)
Data collected to be shared during group construction

Description

Perform the specified operation across the identified processes, plus any special actions included in the directives. Return the result of any special action requests in the callback function when the operation is completed. Actions may include a request (**PMIX_GROUP_ASSIGN_CONTEXT_ID**) that the host assign a unique numerical (size_t) ID to this group - if given, the **PMIX_RANGE** attribute will specify the range across which the ID must be unique (default to **PMIX_RANGE_SESSION**).

Advice to PMIx server hosts

If this module entry is provided and called by the PMIx server library, then the host environment must either provide the requested services or return **PMIX_ERR_NOT_SUPPORTED** to the provided *cbfunc*.

CHAPTER 12

Scheduler-Specific Interfaces

The PMIx server library includes several interfaces specifically intended to support WLMs (also known as *schedulers*) by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric. Due to their high cost in terms of execution, memory consumption, and interactions with other SMS components (e.g., a fabric manager), it is strongly advised that use be restricted to a single PMIx server in a system that is supporting the SMS component responsible for the scheduling of allocations (i.e., the system **scheduler**).

Accordingly, access to the functions described in this chapter requires that the PMIx server library be initialized with the **PMIX_SERVER_SCHEDULER** attribute.

12.1 Scheduler Support Datatypes

12.1.1 Fabric registration structure

The **pmix_fabric_t** structure is used by a WLM to interact with fabric-related PMIx interfaces.

PMIx v4.0

C

```
typedef struct pmix_fabric_s {
    char *name;
    size_t index;
    uint16_t **commcost;
    uint32_t nverts;
    void *module;
} pmix_fabric_t;;
```

C

Note that in this structure:

- the *name* is an optional user-supplied string name identifying the fabric being referenced by this struct;
- a PMIx-provided index identifying this object;
- the *commcost* element is a square, two-dimensional array of **uint16_t** values representing the relative communication cost between the two *(row,col)* vertices. Note that PMIx makes no assumption as to the symmetry of the matrix - while the communication cost of many fabrics is independent of direction (and hence, the *commcost* matrix is symmetric), others may be direction sensitive;

- *nverts* indicates the number of rows and columns in the *commcost* array; and
- *module* points to an opaque object reserved for use by the PMIx server library.

The *name* field must be a **NULL**-terminated string composed of standard alphanumeric values supported by common utilities such as *strcmp*.

12.1.2 Scheduler Support Error Constants

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.

12.1.3 Scheduler Support Attributes

PMIX_SERVER_SCHEDULER **"pmix.srv.sched"** (**bool**)
Server requests access to WLM-supporting features.

12.2 Scheduler Support Functions

The following APIs allow the scheduler that hosts the PMIx server library to request specific services from the PMIx library.

12.2.1 PMIx_server_register_fabric

Summary

Register for access to fabric-related information.

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_server_register_fabric(pmix_fabric_t *fabric,
                            const pmix_info_t directives[],
                            size_t ndirs)
```

C

IN **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 attributes are required to be supported by all PMIx libraries:

PMIX_NETWORK_PLANE **"pmix.net.plane"** (**char***)
ID string of a network 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 in a request, returns a **pmix_data_array_t** of string identifiers for all network planes in the system.

Description

Register for access to fabric-related information, including the communication cost matrix. This call must be made prior to requesting information from a fabric. The caller may request access to a particular **network plane** via the **PMIX_NETWORK_PLANE** attribute - otherwise, the default fabric will be returned.

If available, the *fabric* struct shall contain the address and size of the communication cost matrix associated with the specified network plane. For performance reasons, the PMIx server library does *not* provide thread protection for cost matrix access. Instead, users are required to register for **PMIX_FABRIC_UPDATE_PENDING** events indicating that an update to the cost matrix is pending. When received, users are required to terminate any actions involving access to the cost matrix before returning from the event.

Completion of the **PMIX_FABRIC_UPDATE_PENDING** event handler indicates to the PMIx server library that the fabric object's entries are available for updating. This may include releasing and re-allocating memory as the number of vertices may have changed (e.g., due to addition or removal of one or more NICs). When the update has been completed, the PMIx server library will generate a **PMIX_FABRIC_UPDATED** event indicating that it is safe to begin using the updated fabric object(s).

12.2.2 PMIx_server_deregister_fabric

Summary

Deregister a fabric object.

Format

PMIx v4.0

C

```
pmix_status_t PMIx_server_deregister_fabric(pmix_fabric_t *fabric)
```

C

IN **input**
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 server library to cleanup any information (e.g., cost matrix) associated with it.

12.2.3 PMIx_server_get_vertex_info

Summary

Given a communication cost matrix index for a specified fabric, return the corresponding vertex info and the name of the node upon which it resides.

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_server_get_vertex_info(pmix_fabric_t *fabric,
                            uint32_t index, pmix_value_t *vertex,
                            char **nodename)
```

C

IN `fabric`
address of a `pmix_fabric_t` (handle)

IN `index`
communication cost matrix index (integer)

IN `vertex`
pointer to the `pmix_value_t` where the vertex info is to be returned (backed by storage) (handle)

OUT `nodename`
pointer to the location where the string name of the host is to be returned. The caller is responsible for releasing the string when done (handle)

Returns one of the following:

- `PMIX_SUCCESS`, indicating return of a valid value.
- `PMIX_ERR_BAD_PARAM`, indicating that the provided index is out of bounds.
- a PMIx error constant indicating either an error in the input or that the request failed.

Description

12.2.4 PMIx_server_get_index

Summary

Given vertex info, return the corresponding communication cost matrix index.

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_server_get_index(pmix_fabric_t *fabric,
                      pmix_value_t *vertex,
                      uint32_t *index)
```

C

IN `fabric`
address of a `pmix_fabric_t` (handle)

IN `vertex`
pointer to the `pmix_value_t` containing the vertex info (handle)

OUT `index`
pointer to the location where the index is to be returned (memory reference (handle))

Description

Returns one of the following:

- `PMIX_SUCCESS` , indicating return of a valid value.
- a PMIx error constant indicating either an error in the input or that the request failed.

Description

CHAPTER 13

Process Sets and Groups

PMIx supports two slightly related, but functionally different concepts known as *process sets* and *process groups*. This chapter these two concepts and describes how they are utilized, along with their corresponding APIs.

13.1 Process Sets

A PMIx *Process Set* is a user-provided label associated with a given set of application processes. Definition of a PMIx process set typically occurs at time of application execution - e.g., on a PRRTE command line:

C

```
$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp
```

C

In this example, the processes in the first application will be labeled with a **PMIX_PSET_NAME** attribute of *ocean* while those in the second application will be labeled with an *ice* value. During the execution, application processes could lookup the process set attribute for any other process using **PMIx_Get** . Alternatively, other executing applications could utilize the **PMIx_Query_info_nb** API to obtain the number of declared process sets in the system, a list of their names, and other information about them. In other words, the *process set* identifier provides a label by which an application can derive information about a process and its application - it does *not*, however, confer any operational function.

Thus, process *sets* differ from process *groups* in several key ways:

- Process *sets* have no implied relationship between their members - i.e., a process in a process set has no concept of a “pset rank” as it would in a process *group*
- Processes can only have one process *set* identifier, but can simultaneously belong to multiple process *groups*
- Process *set* identifiers are considered job-level information set at launch. No PMIx API is provided by which a user can change the process *set* value of a process on-the-fly. In contrast, PMIx process *groups* can only be defined dynamically by the application.

- 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 identifers of the participating processes - prior to passing the request up to the host environment
- Process *groups* can request that the host environment assign a unique **size_t** PGCID to the group at time of group construction. An MPI library may, for example, use the PGCID as the MPI communicator identifier for the group.

The two concepts do, however, overlap in one specific area. Process *groups* are included in the process *set* information returned by calls to **PMIx_Query_info_nb** . Thus, a *process group* can effectively be considered an extended version of a *process set* that adds dynamic definition and operational context to the *process set* concept.

Advice to PMIx library implementers

PMIx implementations are required to include all active *group* identifiers in the returned list of process *set* names provided in response to the appropriate **PMIx_Query_info_nb** call.

13.2 Process Groups

PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for 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, *groups* differ from **PMIx_Connect** assemblages in the following key areas:

- Relation to the host environment
  - Calls to **PMIx_Connect** are relayed to the host environment. This means that the host RM should treat the failure of any process in the specified assemblage as a reportable event and take appropriate action. However, the environment is not required to define a new identifier for the connected assemblage or any of its member processes, nor does it define a new rank for each process within that assemblage. In addition, the PMIx server does not provide any tracking support for the assemblage. Thus, the caller is responsible for addressing members of the connected assemblage using their RM-provided identifiers.

- 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 utilize the group identifier in PMIx function calls to address the group's members, using either **PMIX_RANK_WILDCARD** to refer to all of them or the group-level rank of specific members. The PMIx server will translate the specified processes into their RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment has no visibility into the group's existence or membership.

Advice to users

User-provided group identifiers must be distinct from anything provided by the RM so as to avoid collisions between group identifiers and RM-assigned namespaces. This can usually be accomplished through the use of an application-specific prefix – e.g., "myapp-foo"

- Construction procedure
  - **PMIx_Connect** calls require that every process call the API before completing – i.e., it is modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a given application thread can only be involved in one connect/accept operation at a time, and is blocked in that operation until all specified processes participate. In addition, there is no provision for replacing processes in the assemblage due to failure to participate, nor a mechanism by which a process might decline participation.
  - PMIx Groups are designed to be more flexible in their construction procedure by relaxing these constraints. While a standard blocking form of constructing groups is provided, the event notification system is utilized to provide a designated *group leader* with the ability to replace participants that fail to participate within a given timeout period. This provides a mechanism by which the application can, if desired, replace members on-the-fly or allow the group to proceed with partial membership. In such cases, the final group membership is returned to all participants upon completion of the operation.

    Additionally, PMIx supports dynamic definition of group membership based on an invite/join model. A process can asynchronously initiate construction of a group of any processes via the **PMIx_Group_invite** function call. Invitations are delivered via a PMIx event (using the **PMIX_GROUP_INVITED** event) to the invited processes which can then either accept or decline the invitation using the **PMIx_Group_join** API. The initiating process tracks responses by registering for the events generated by the call to **PMIx_Group_join** , timeouts, or process terminations, optionally replacing processes that decline the invitation, fail to respond in time, or terminate without responding. Upon completion of the operation, the final list of participants is communicated to each member of the new group.
- Destruct procedure

- Processes that assemble via **PMIx_Connect** must all depart the assemblage together – i.e., no member can depart the assemblage while leaving the remaining members in it. Even the non-blocking form of **PMIx_Disconnect** retains this requirement in that members remain a part of the assemblage until all members have called **PMIx_Disconnect_nb**
- Members of a PMIx Group may depart the group at any time via the **PMIx_Group_leave** API. Other members are notified of the departure via the **PMIX_GROUP_LEFT** event to distinguish such events from those reporting process termination. This leaves the remaining members free to continue group operations. The **PMIx_Group_destruct** operation offers a collective method akin to **PMIx_Disconnect** for deconstructing the entire group.

  Note that applications supporting dynamic group behaviors such as asynchronous departure take responsibility for ensuring global consistency in the group definition prior to executing group collective operations - i.e., it is the application's responsibility to either ensure that knowledge of the current group membership is globally consistent across the participants, or to register for appropriate events to deal with the lack of consistency during the operation.

In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected* when constructed via **PMIx_Connect** . The relevant APIs are explained below.

Advice to users

The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs must register for the corresponding events. Failure to do so will likely lead to operational failures. Users are recommended to utilize the **PMIX_TIMEOUT** directive (or retain an internal timer) on calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have not registered for required events will never respond.

13.2.1 Group Operation Constants

PMIx v4.0

The **pmix_group_operation_t** structure is an enumerated type for specifying group operations. All values were originally defined in version 4 of the standard unless otherwise marked.

PMIX_GROUP_DECLINE Decline an invitation to join a PMIx group - provided for readability of user code

PMIX_GROUP_ACCEPT Accept an invitation to join a PMIx group - provided for readability of user code

PMIX_GROUP_CONSTRUCT Construct a group composed of the specified processes - used by a PMIx server library to direct host operation

PMIX_GROUP_DESTRUCT Destruct the specified group - used by a PMIx server library to direct host operation

13.2.2 PMIx_Group_construct

Summary

Construct a PMIx process group

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_construct(const char grp[],
                     const pmix_proc_t procs[], size_t nprocs,
                     const pmix_info_t directives[], size_t ndirs,
                     pmix_info_t **results, size_t *nresults)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **procs**
Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes (array of handles)

IN **nprocs**
Number of elements in the *procs* array (**size_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

INOUT **results**
Pointer to a location where the array of **pmix_info_t** describing the results of the operation is to be returned (pointer to handle)

INOUT **nresults**
Pointer to a **size_t** location where the number of elements in *results* is to be returned (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation
- a PMIx error constant indicating either an error in the input or that the request failed to be completed

Required Attributes

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. The default is false

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. The default is false

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 (*0* indicating infinite) in error. The timeout parameter can help avoid “hangs” due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less than or equal to **PMIX_MAX_NSLEN** . Only characters accepted by standard string comparison functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

If the **PMIX_GROUP_NOTIFY_TERMINATION** attribute is provided and has a value of **true**, then either the construct leader (if **PMIX_GROUP_LEADER** is provided) or all participants who register for the **PMIX_GROUP_MEMBER_FAILED** event will receive events whenever a process fails or terminates prior to calling **PMIx_Group_construct** – i.e. if a *group leader* is declared, *only* that process will receive the event. In the absence of a declared leader, *all* specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the **PMIX_GROUP_MEMBERSHIP** attribute) as part of the *results* array returned by this API.

Failure of the declared leader at any time will cause a **PMIX_GROUP_LEADER_FAILED** event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected, then the **pmix_info_t** provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from either the **PMIX_GROUP_MEMBER_FAILED** or the **PMIX_GROUP_LEADER_FAILED** event handler will

cause the construct process to abort, returning from the call with a **PMIX_GROUP_CONSTRUCT_ABORT** status.

If the **PMIX_GROUP_NOTIFY_TERMINATION** attribute is not provided or has a value of **false**, then the **PMIx_Group_construct** operation will simply return an error whenever a proposed group member fails or terminates prior to calling **PMIx_Group_construct** .

Providing the **PMIX_GROUP_OPTIONAL** attribute with a value of **true** directs the PMIx library to consider participation by any specified group member as non-required - thus, the operation will return **PMIX_SUCCESS** if all members participate, or **PMIX_ERR_PARTIAL_SUCCESS** if some members fail to participate. The *results* array will contain the final group membership in the latter case. Note that this use-case can cause the operation to hang if the **PMIX_TIMEOUT** attribute is not specified and one or more group members fail to call **PMIx_Group_construct** while continuing to execute. Also, note that no leader or member failed events will be generated during the operation.

Processes in a group under construction are not allowed to leave the group until group construction is complete. Upon completion of the construct procedure, each group member will have access to the job-level information of all namespaces represented in the group plus any information posted via **PMIx_Put** (subject to the usual scoping directives) for every group member.

Advice to PMIx library implementers

At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related information from each participating namespace plus any information posted by group members via **PMIx_Put** (subject to scoping directives) is available to each member via calls to **PMIx_Get** .

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_construct** 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.

13.2.3 PMIx_Group_construct_nb

Summary

Non-blocking form of **PMIx_Group_construct**

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_construct_nb(const char grp[],
                        const pmix_proc_t procs[], size_t nprocs,
                        const pmix_info_t directives[], size_t ndirs,
                        pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **procs**
Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes (array of handles)

IN **nprocs**
Number of elements in the *procs* array (**size_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

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.
- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc* will *not* be called
- a non-zero PMIx error constant indicating a reason for the request to have been rejected - the *cbfunc* will *not* be called

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

- **PMIX_SUCCESS** The operation succeeded and all specified members participated.

- **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function
- **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. The default is false

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. The default is false

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking version of the **PMIx_Group_construct** operation. The callback function will be called once all group members have called either **PMIx_Group_construct** or **PMIx_Group_construct_nb** .

13.2.4 PMIx_Group_destruct

Summary

Destruct a PMIx process group

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_destruct(const char grp[],
                    const pmix_info_t directives[], size_t ndirs)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the identifier of the group to be destructed (string)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation
- a PMIx error constant indicating either an error in the input or that the request failed to be completed

Required Attributes

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

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.

13.2.5 PMIx_Group_destruct_nb

Summary

Non-blocking form of **PMIx_Group_destruct**

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_destruct_nb(const char grp[],
                       const pmix_info_t directives[], size_t ndirs,
                       pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the identifier of the group to be destructed (string)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

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

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

- **PMIX_SUCCESS** The operation was successfully completed
- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

PMIx libraries that choose not to support this operation *must* return **PMIX_ERR_NOT_SUPPORTED** when the function is called. For implementations and host environments that support the operation, there are no identified required attributes for this API.

Optional Attributes

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

PMIX_TIMEOUT **"pmix.timeout"** (**int**)
: Time in seconds before the specified operation should time out (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking version of the **PMIx_Group_destruct** operation. The callback function will be called once all members of the group have executed either **PMIx_Group_destruct** or **PMIx_Group_destruct_nb** .

13.2.6 PMIx_Group_invite

Summary

Asynchronously construct a PMIx process group

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_invite(const char grp[],
                  const pmix_proc_t procs[], size_t nprocs,
                  const pmix_info_t directives[], size_t ndirs,
                  pmix_info_t **results, size_t *nresult)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **procs**
Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be invited (array of handles)

IN **nprocs**
Number of elements in the *procs* array (**size_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

INOUT **results**
Pointer to a location where the array of **pmix_info_t** describing the results of the operation is to be returned (pointer to handle)

INOUT **nresults**
Pointer to a **size_t** location where the number of elements in *results* is to be returned (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation
- a PMIx error constant indicating either an error in the input or that the request failed to be completed

Required Attributes

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

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. The default is false

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Explicitly invite the specified processes to join a group. The process making the **PMIx_Group_invite** call is automatically declared to be the *group leader*. Each invited process will be notified of the invitation via the **PMIX_GROUP_INVITED** event - the processes being invited must therefore register for the **PMIX_GROUP_INVITED** event in order to be notified of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of invite versus event registration is required.

The invitation event will include the identity of the inviting process plus the name of the group. When ready to respond, each invited process provides a response using either the blocking or non-blocking form of **PMIx_Group_join** . This will notify the inviting process that the invitation was either accepted (via the **PMIX_GROUP_INVITE_ACCEPTED** event) or declined (via the **PMIX_GROUP_INVITE_DECLINED** event). The **PMIX_GROUP_INVITE_ACCEPTED** event is captured by the PMIx client library of the inviting process – i.e., the application itself does not need to register for this event. The library will track the number of accepting processes and alert the inviting process (by returning from the blocking form of **PMIx_Group_invite** or calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the **PMIX_GROUP_INVITE_DECLINED** if the application allows invited processes to decline the invitation. This provides an opportunity for the application to either invite a replacement, declare "abort", or choose to remove the declining process from the final group. The inviting process should also register to receive **PMIX_GROUP_INVITE_FAILED** events whenever a process fails or terminates prior to responding to the invitation. Actions taken by the inviting process in response to these events must be communicated at the end of the event handler by returning the corresponding result so that the PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level information of each other's namespaces plus any information posted via **PMIx_Put** by the other members.

The inviting process is automatically considered the leader of the asynchronous group construction procedure and will receive all failure or termination events for invited members prior to completion. The inviting process is required to provide a **PMIX_GROUP_CONSTRUCT_COMPLETE** event once the group has been fully assembled – this event is used by the PMIx library as a trigger to release participants from their call to **PMIx_Group_join** and provides information (e.g., the final group membership) to be returned in the *results* array.

Advice to users

Applications are not allowed to use the group in any operations until group construction is complete. This is required in order to ensure consistent knowledge of group membership across all participants.

Failure of the inviting process at any time will cause a **PMIX_GROUP_LEADER_FAILED** event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

13.2.7 PMIx_Group_invite_nb

Summary

Non-blocking form of **PMIx_Group_invite**

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_invite_nb(const char grp[],
                     const pmix_proc_t procs[], size_t nprocs,
                     const pmix_info_t directives[], size_t ndirs,
                     pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **procs**
Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be invited (array of handles)

IN **nprocs**
Number of elements in the *procs* array (**size_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Returns one of the following:

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

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

- **PMIX_SUCCESS** The operation succeeded and all specified members participated.
- **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function
- **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

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. The default is false

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

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** .

13.2.8 PMIx_Group_join

Summary

Accept an invitation to join a PMIx process group

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_join(const char grp[],
                const pmix_proc_t *leader,
                pmix_group_operation_t opt,
                const pmix_info_t directives[], size_t ndirs,
                pmix_info_t **results, size_t *nresult)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **leader**
Process that generated the invitation (handle)

IN **opt**
Accept or decline flag (**pmix_group_operation_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

INOUT **results**
Pointer to a location where the array of **pmix_info_t** describing the results of the operation is to be returned (pointer to handle)

INOUT **nresults**
Pointer to a **size_t** location where the number of elements in *results* is to be returned (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request has been successfully completed
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation
- a PMIx error constant indicating either an error in the input or that the request failed to be completed

Required Attributes

There are no identified required attributes for implementers.

Optional Attributes

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Respond to an invitation to join a group that is being asynchronously constructed. The process must have registered for the **PMIX_GROUP_INVITED** event in order to be notified of the invitation. When called, the event information will include the **pmix_proc_t** identifier of the process that generated the invitation along with the identifier of the group being constructed. When ready to respond, the process provides a response using either form of **PMIx_Group_join** .

Advice to users

Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the blocking form of this call unless it first "thread shifts" out of the handler and into its own thread context. Likewise, while it is safe to call the non-blocking form of the API from the event handler, the process *must not* block in the handler while waiting for the callback function to be called.

Calling this function causes the inviting process (aka the *group leader*) to be notified that the process has either accepted or declined the request. The blocking form of the API will return once the group has been completely constructed or the group's construction has failed (as described below) – likewise, the callback function of the non-blocking form will be executed upon the same conditions.

Failure of the leader during the call to **PMIx_Group_join** will cause a **PMIX_GROUP_LEADER_FAILED** event to be delivered to all invited participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from the leader failed event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning **PMIX_GROUP_CONSTRUCT_ABORT** from the handler assigned to the **PMIX_GROUP_INVITE_ACCEPTED** or **PMIX_GROUP_INVITE_DECLINED** codes, or by generating an event for the abort code. Abort events will be sent to all invited participants.

13.2.9 PMIx_Group_join_nb

Summary

Non-blocking form of **PMIx_Group_join**

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_join_nb(const char grp[],
                   const pmix_proc_t *leader,
                   pmix_group_operation_t opt,
                   const pmix_info_t directives[], size_t ndirs,
                   pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **leader**
Process that generated the invitation (handle)

IN **opt**
Accept or decline flag (**pmix_group_operation_t**)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_info_cbfunc_t** (function reference)

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

Returns one of the following:

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

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

- **PMIX_SUCCESS** The operation succeeded and group membership is in the callback function parameters
- **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

There are no identified required attributes for implementers.

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 (*0* indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

Advice to PMIx library implementers

We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

Description

Non-blocking version of the **PMIx_Group_join** operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either **PMIx_Group_join** or **PMIx_Group_join_nb** .

13.2.10 PMIx_Group_leave

Summary

Leave a PMIx process group

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_leave(const char grp[],
                 const pmix_info_t directives[], size_t ndirs)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

Returns 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

There are no identified required attributes for implementers.

Description

Leave a PMIx Group. Calls to **PMIx_Group_leave** (or its non-blocking form) will cause a **PMIX_GROUP_LEFT** event to be generated notifying all members of the group of the caller's departure. The function will return (or the non-blocking function will execute the specified callback function) once the event has been locally generated and is not indicative of remote receipt.

Advice to users

The PMIx_Group_leave API is intended solely for asynchronous departures of individual processes from a group as it is not a scalable operation – i.e., when a process determines it should no longer be a part of a defined group, but the remainder of the group retains a valid reason to continue in existence. Developers are advised to use PMIx_Group_destruct (or its non-blocking form) for all other scenarios as it represents a more scalable operation.

13.2.11 PMIx_Group_leave_nb

Summary

Non-blocking form of **PMIx_Group_leave**

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_leave_nb(const char grp[],
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)

IN **directives**
Array of **pmix_info_t** structures (array of handles)

IN **ndirs**
Number of elements in the *directives* array (**size_t**)

IN **cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

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

- **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

Network Coordinates

As the drive for performance continues, interest has grown in optimizing collective communication patterns by structuring them to follow network topology. For example, one might aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches. Creating such optimized patterns therefore relies on detailed knowledge of the network location of each participant.

PMIx supports these efforts by defining datatypes and attributes by which network coordinates for processes and devices can be obtained from the host SMS. When used in conjunction with the PMIx *instant on* methods, this results in the ability of a process to obtain the network coordinate of all other processes without incurring additional overhead associated with the publish/exchange of that information.

14.1 Network Coordinate Datatypes

Several datatype definitions have been created to support network coordinates.

14.1.1 Network Coordinate Structure

The **pmix_coord_t** structure describes the network coordinates of a specified process in a given view

PMIx v4.0

C

```
typedef struct pmix_coord {
    char *fabric;
    char *plane;
    pmix_coord_view_t view;
    uint32_t *coord;
    size_t dims;
} pmix_coord_t;
```

C

All coordinate values shall be expressed as unsigned integers due to their units being defined in network devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

The fabric and plane fields are assigned by the fabric provider to help the user identify the network to which the coordinates refer. Note that providers are not required to assign any particular value to the fields and may choose to leave the fields blank. Example entries include {"Ethernet", "mgmt"} or {"infiniband", "data1"}.

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 network coordinate is usually associated with a given network device - e.g., a particular NIC on a node. Thus, while the network coordinate of a device must be unique in a given view, the coordinate may be shared by multiple processes on a node. If the node contains multiple network devices, then either the device closest to the binding location of a process shall be used as its coordinate, or (if the process is unbound or its binding is not known) all devices on the node shall be reported as a **pmix_data_array_t** of **pmix_coord_t** structures.

Nodes with multiple network devices can also have those devices configured as multiple **network planes** . In such cases, a given process (even if bound to a specific location) may be associated with a coordinate on each plane. The resulting set of network coordinates shall be reported as a **pmix_data_array_t** of **pmix_coord_t** structures. The caller may request a coordinate from a specific network plane by passing the **PMIX_NETWORK_PLANE** attribute as a directive/qualifier to the **PMIx_Get** or **PMIx_Query_info_nb** call.

14.1.2 Network Coordinate Support Macros

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

14.1.2.1 Initialize the pmix_coord_t structure

Initialize the **pmix_coord_t** fields

PMIx v4.0

C

```
PMIX_COORD_CONSTRUCT(m)
```

C

IN **m**
Pointer to the structure to be initialized (pointer to **pmix_coord_t**)

14.1.2.2 Destruct the pmix_coord_t structure

Destruct the **pmix_coord_t** fields

PMIx v4.0

C

```
PMIX_COORD_DESTRUCT(m)
```

C

IN **m**
Pointer to the structure to be destructed (pointer to **pmix_coord_t**)

14.1.2.3 Create a pmix_coord_t array

Allocate and initialize a **pmix_coord_t** array

PMIx v4.0

C

```
PMIX_COORD_CREATE(m, n)
```

C

INOUT **m**
: Address where the pointer to the array of **pmix_coord_t** structures shall be stored (handle)

IN **n**
: Number of structures to be allocated (**size_t**)

14.1.2.4 Release a pmix_coord_t array

Release an array of **pmix_coord_t** structures

PMIx v4.0

C

```
PMIX_COORD_FREE(m, n)
```

C

IN **m**
: Pointer to the array of **pmix_coord_t** structures (handle)

IN **n**
: Number of structures in the array (**size_t**)

14.1.3 Network Coordinate Views

PMIx v4.0

C

```
typedef uint8_t pmix_coord_view_t;
#define PMIX_COORD_VIEW_UNDEF        0x00
#define PMIX_COORD_LOGICAL_VIEW      0x01
#define PMIX_COORD_PHYSICAL_VIEW     0x02
```

C

Network coordinates can be reported based on different *views* according to user preference at the time of request. The following views have been defined:

PMIX_COORD_VIEW_UNDEF The coordinate view has not been defined.

PMIX_COORD_LOGICAL_VIEW The coordinates are provided in a *logical* view, typically given in Cartesian (x,y,z) dimensions, that describes the data flow in the network as defined by the arrangement of the hierarchical addressing scheme, network segmentation, routing domains, and other similar factors employed by that network.

PMIX_COORD_PHYSICAL_VIEW The coordinates are provided in a *physical* view based on the actual wiring diagram of the network - i.e., values along each axis reflect the relative position of that interface on the specific network cabling.

Advice to PMIx library implementers

PMIx library implementers are advised to avoid declaring the above constants as actual **enum** values in order to allow host environments to add support for possibly proprietary coordinate views.

If the requester does not specify a view, coordinates shall default to the *logical* view.

14.1.4 Network Coordinate Error Constants

The following error constants are used by PMIx to notify registered processes of events that affect network coordinates.

PMIX_NETWORK_COORDS_UPDATED Network coordinates have been updated - the affected networks/planes are identified in the notification. Coordinates of processes and devices on those affected components should be refreshed prior to next use.

14.1.5 Network Descriptive Attributes

These attributes are used to describe information about network resources as assigned by the RM, and thus are referenced using the process rank except where noted.

PMIX_NETWORK_COORDINATE **"pmix.net.coord"** (**pmix_data_array_t**)
Network coordinate(s) of the specified process in the view and/or plane provided by the requester. If only one NIC has been assigned to the specified process, then the array will contain only one address. Otherwise, the array will contain the coordinates of all NICs available to the process in order of least to greatest distance from the process (NICs equally distant from the process will be listed in arbitrary order).

PMIX_NETWORK_VIEW **"pmix.net.view"** (**pmix_coord_view_t**)
Network coordinate view to be used for the requested data - see **pmix_coord_view_t** for the list of accepted values.

PMIX_NETWORK_DIMS **"pmix.net.dims"** (**uint32_t**)
Request number of dimensions in the specified network plane/view. If no plane is specified, then the dimensions of all planes in the system will be returned as a **pmix_data_array_t** containing an array of **uint32_t** values. Default is to provide dimensions in *logical* view.

PMIX_NETWORK_PLANE **"pmix.net.plane"** (**char***)
ID string of a network 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 in a request, returns a **pmix_data_array_t** of string identifiers for all network planes in the system.

PMIX_NETWORK_NIC **"pmix.net.nic"** (**char***)

ID string of a network interface card (NIC). When used as a modifier in a request for information, specifies the NIC whose information is to be returned. When used directly in a request, returns a **pmix_data_array_t** of string identifiers for all NICs in the specified network plane. If no plane is specified, then the NIC identifiers of each plane in the system will be returned in an array where each element is in turn an array of strings containing the network plane ID followed by the identifiers of the NICs attached to that plane.

PMIX_NETWORK_ENDPT **"pmix.net.endpt"** (**pmix_data_array_t**)
Network endpoints for a specified process. As multiple endpoints may be assigned to a given process (e.g., in the case where multiple NICs are associated with a socket to which the process is bound), the returned values will be provided in a **pmix_data_array_t** - the returned data type of the individual values in the array varies by fabric provider.

PMIX_NETWORK_SHAPE **"pmix.net.shape"** (**pmix_data_array_t***)
The size of each dimension in the specified network plane/view, returned in a **pmix_data_array_t** containing an array of **uint32_t** values. The size is defined as the number of elements present in that dimension - e.g., the number of NICs in one dimension of a physical view of a network plane. If no plane is specified, then the shape of each plane in the system will be returned in an array of network shapes. Default is to provide the shape in *logical* view.

APPENDIX A

Python Bindings

While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of PMIx to that specific language. Support for other languages is captured in the Standard by describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes. Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.

Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx "class" (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and therefore those have been included in the "client" class. Servers can utilize any of those functions plus a set focused on operations not commonly executed by an application process. Finally, tools 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 (nspace, rank) or as a dictionary 'nspace': 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.

A.2 Datatype Definitions

PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range integers (e.g., uint32_t), and structures. Each datatype is represented by a named unsigned 16-bit integer (**uint16_t**) constant. Users are advised to use the named PMIx constants for indicating datatypes instead of integer values to ensure compatibility with future PMIx versions.

With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 20 directly translate to Python. However, Python lacks the size-specific value definitions of C (e.g., **uint8_t**) and thus some care must be taken to protect against overflow/underflow situations when moving between the languages. Python bindings that accept values including PMIx datatypes shall therefore have the datatype and associated value checked for compatibility with their PMIx-defined equivalents, returning an error if:

- datatypes not defined by PMIx are encountered
- provided values fall outside the range of the C-equivalent definition - e.g., if a value identified as **PMIX_UINT8** lies outside the **uint8_t**range

Note that explicit labeling of PMIx datatype, even when Python itself doesn't care, is often required for the Python bindings to know how to properly interpret and label the provided value when passing it to the PMIx library.

Table A.1 lists the correspondence between datatypes in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
bool	PMIX_BOOL	boolean	
byte	PMIX_BYTE	A single element byte array (i.e., a byte array of length one)	
char*	PMIX_STRING	string	
size_t	PMIX_SIZE	integer	
pid_t	PMIX_PID	integer	value shall be limited to the **uint32_t** range
int, int8_t, int16_t, int32_t, int64_t	PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64	integer	value shall be limited to its corresponding range
uint, uint8_t, uint16_t, uint32_t, uint64_t	PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64	integer	value shall be limited to its corresponding range
float, double	PMIX_FLOAT, PMIX_DOUBLE	float	value shall be limited to its corresponding range
struct timeval	PMIX_TIMEVAL	{'sec': sec, 'usec': microsec}	each field is an integer value
time_t	PMIX_TIME	integer	limited to positive values
pmix_data_type_t	PMIX_DATA_TYPE	integer	value shall be limited to the **uint16_t** range
pmix_status_t	PMIX_STATUS	integer	
pmix_key_t	N/A	string	The string's length shall be limited to one less than the size of the **pmix_key_t** array (to reserve space for the terminating **NULL**)
pmix_nspace_t	N/A	string	The string's length shall be limited to one less than the size of the **pmix_nspace_t** array (to reserve space for the terminating **NULL**)

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
pmix_rank_t	PMIX_PROC_RANK	integer	value shall be limited to the **uint32_t** range excepting the reserved values near **UINT32_MAX**
pmix_proc_t	PMIX_PROC	{'nspace': nspace, 'rank': rank}	*nspace* is a Python string and *rank* is an integer value. The *nspace* string's length shall be limited to one less than the size of the **pmix_nspace_t** array (to reserve space for the terminating **NULL**), and the *rank* value shall conform to the constraints associated with **pmix_rank_t**
pmix_byte_object_t	PMIX_BYTE_OBJECT	{'bytes': bytes, 'size': size}	*bytes* is a Python byte array and *size* is the integer number of bytes in that array.
pmix_persistence_t	PMIX_PERSISTENCE	integer	value shall be limited to the **uint8_t** range
pmix_scope_t	PMIX_SCOPE	integer	value shall be limited to the **uint8_t** range
pmix_data_range_t	PMIX_RANGE	integer	value shall be limited to the **uint8_t** range
pmix_proc_state_t	PMIX_PROC_STATE	integer	value shall be limited to the **uint8_t** range
pmix_proc_info_t	PMIX_PROC_INFO	{'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state}	*proc* is a Python **proc** dictionary; *hostname* and *executable* are Python strings; and *pid*, *exitcode*, and *state* are Python integers

Table A.1.: C-to-Python Datatype Correspondence

C-Definition	PMIx Name	Python Definition	Notes
pmix_data_array_t	PMIX_DATA_ARRAY	{'type': type, 'array': array}	*type* is the PMIx type of object in the array and *array* is a Python *list* containing the individual array elements. Note that *array* can consist of *any* PMIx types, including (for example) a Python **info** object that itself contains an **array** value
pmix_info_directives_t	PMIX_INFO_DIRECTIVES	integer	value shall be limited to the **uint32_t** range
pmix_alloc_directive_t	PMIX_ALLOC_DIRECTIVE	integer	value shall be limited to the **uint8_t** range
pmix_iof_channel_t	PMIX_IOF_CHANNEL	integer	value shall be limited to the **uint16_t** range
pmix_envar_t	PMIX_ENVAR	{'envar': envar, 'value': value, 'separator': separator}	*envar* and *value* are Python strings, and *separator* a single-character Python string
pmix_value_t	PMIX_VALUE	{'value': value, 'val_type': type}	*type* is the PMIx datatype of *value*, and *value* is the associated value expressed in the appropriate Python form for the specified datatype
pmix_info_t	PMIX_INFO	{'key': key, 'flags': flags, value': value, 'val_type': type}	*key* is a Python string **key** , *flags* is a bitmask of **info directives** , *type* is the PMIx datatype of *value*, and *value* is the associated value expressed in the appropriate Python form for the specified datatype
pmix_pdata_t	PMIX_PDATA	{'proc': {'nspace': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type}	*proc* is a Python **proc** dictionary; *key* is a Python string **key** ; *type* is the PMIx datatype of *value*; and *value* 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
pmix_app_t	PMIX_APP	{'cmd': cmd, 'argv': [argv], 'env': [env], 'maxprocs': maxprocs, 'info': [info]}	*cmd* is a Python string; *argv* and *env* are Python *lists* containing Python strings; *maxprocs* is an integer; and *info* is a Python *list* of **info** values
pmix_query_t	PMIX_QUERY	{'keys': [keys], 'qualifiers': [info]}	*keys* is a Python *list* of Python strings, and *qualifiers* is a Python *list* of **info** values
pmix_regattr_t	PMIX_REGATTR	{'name': name, 'key': key, 'type': type, info': [info], 'description': [desc]}	*name* and *string* are Python strings; *type* is the PMIx datatype for the attribute's value; *info* is a Python *list* of **info** values; and *description* is a list of Python strings describing the attribute

A.2.1 Example

Converting a C-based program to its Python equivalent requires translation of the relevant datatypes as well as use of the appropriate API form. An example small program may help illustrate the changes. Consider the following C-based program snippet:

```
#include <pmix.h>
...

pmix_info_t info[2];

PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "TEST", PMIX_STRING)
PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "PMIX", PMIX_STRING)

rc = PMIx_Init(&myproc, info, 2);

PMIX_INFO_DESTRUCT(&info[0]);  // free the copied string
PMIX_INFO_DESTRUCT(&info[1]);  // free the copied string
```

Moving to the Python version requires that the **pmix_info_t** be translated to the Python **info** equivalent, and that the returned information be captured in the return parameters as opposed to a pointer parameter in the function call, as shown below:

```
import pmix
...

myclient = PMIxClient()
info = [{'key':PMIX_PROGRAMMING_MODEL,
           'value':'TEST', 'val_type':PMIX_STRING},
        {'key':PMIX_MODEL_LIBRARY_NAME,
           'value':'PMIX', 'val_type':PMIX_STRING}]
rc,myproc = myclient.init(info)
```

Note the use of the **PMIX_STRING** identifier to ensure the Python bindings interpret the provided string value as a PMIx "string" and not an array of bytes.

A.3 Function Definitions

A.3.1 IOF Delivery Function

Summary

Callback function for delivering forwarded IO to a process

Format

PMIx v4.0

Python

```
def iofcbfunc(iofhdlr:integer, channel:integer,
              source:dict, payload:dict, info:list)
```

Python

IN **iofhdlr**
Registration number of the handler being invoked (integer)

IN **channel**
Python **channel** bitmask identifying the channel the data arrived on (integer)

IN **source**
Python **proc** identifying the namespace/rank of the process that generated the data (dict)

IN **payload**
Python **byteobject** containing the data (dict)

IN **info**
List of Python **info** provided by the source containing metadata about the payload. This could include **PMIX_IOF_COMPLETE** (list)

Returns: nothing

See **pmix_iof_cbfunc_t** for details

A.3.2 Event Handler

Summary

Callback function for event handlers

Format

PMIx v4.0

Python

```
def evhandler(evhdlr:integer, status:integer,
              source:dict, info:list, results:list)
```

Python

IN **iofhdlr**
Registration number of the handler being invoked (integer)

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

IN **source**
Python **proc** identifying the namespace/rank of the process that generated the event (dict)

IN **info**
List of Python **info** provided by the source containing metadata about the event (list)

IN **results**
List of Python **info** containing the aggregated results of all prior evhandlers (list)

Returns:

- *rc* - Status returned by the event handler's operation (integer)
- *results* - List of Python **info** containing results from this event handler's operation on the event (list)

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

```
def clientconnected(proc:dict is not None)
```

Python

IN **proc**
Python **proc** identifying the namespace/rank of the process that connected (dict)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the connection should be rejected (integer)

See **pmix_server_client_connected_fn_t** for details

A.3.3.2 Client Finalized

Summary

Notify the host environment that a client called **PMIx_Finalize** .

Format

PMIx v4.0

Python

```
def clientfinalized(proc:dict is not None):
```

Python

IN **proc**
Python **proc** identifying the namespace/rank of the process that finalized (dict)

Returns: nothing

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** .

Format

PMIx v4.0

Python

```
def clientaborted(proc:dict is not None, status:integer,
                  msg:str, targets:list)
```

Python

IN **proc**
Python **proc** identifying the namespace/rank of the process that called abort (dict)

IN **status**
PMIx status to be returned on exit (integer)

IN **msg**
String message to be printed (string)

IN **targets**
List of Python **proc** dictionaries (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

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**

Format

PMIx v4.0

Python

```
def fence(procs:list, directives:list, data:bytearray)
```

Python

IN **procs**
List of Python **proc** dictionaries (list)

IN **directives**
List of Python **info** dictionaries (list)

IN **data**
Python bytearray of data to be circulated during fence operation (bytearray)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *data* - Python bytearray containing the aggregated data from all participants (bytearray)

See **pmix_server_fencenb_fn_t** for details

A.3.3.5 Direct Modex

Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc.

Format

PMIx v4.0

Python

```
def dmodex(proc:dict, directives:list)
```

Python

IN `proc`
Python `proc` dictionary of process whose data is being requested (list)

IN `directives`
List of Python `info` dictionaries (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)
- *data* - Python bytearray containing the data for the specified process (bytearray)

See `pmix_server_dmodex_req_fn_t` for details

A.3.3.6 Publish

Summary

Publish data per the PMIx API specification.

Format

PMIx v4.0

Python

```
def publish(proc:dict, directives:list)
```

Python

IN `proc`
Python `proc` dictionary of process publishing the data (list)

IN `directives`
List of Python `info` dictionaries containing data and directives (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

See `pmix_server_publish_fn_t` for details

A.3.3.7 Lookup

Summary

Lookup published data.

Format

PMIx v4.0

Python

```
def lookup(proc:dict, keys:list, directives:list)
```

Python

IN **`proc`**
Python **`proc`** dictionary of process seeking the data (list)

IN **`keys`**
List of Python strings (list)

IN **`directives`**
List of Python **`info`** dictionaries containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)
- *pdata* - List of **`pdata`** containing the returned results (list)

See **`pmix_server_lookup_fn_t`** for details

A.3.3.8 Unpublish

Summary

Delete data from the data store.

Format

PMIx v4.0

Python

```
def unpublish(proc:dict, keys:list, directives:list)
```

Python

IN **`proc`**
Python **`proc`** dictionary of process making the request (list)

IN **`keys`**
List of Python strings (list)

IN **`directives`**
List of Python **`info`** dictionaries containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)

See **`pmix_server_unpublish_fn_t`** for details

A.3.3.9 Spawn

Summary

Spawn a set of applications/processes as per the **`PMIx_Spawn`** API.

Format

PMIx v4.0

Python

```
def spawn(proc:dict, jobInfo:list, apps:list)
```

Python

IN **proc**
Python **proc** dictionary of process making the request (list)

IN **jobInfo**
List of Python **info** job-level directives and information (list)

IN **apps**
List of Python **app** dictionaries describing applications to be spawned (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- *nspace* - Python string containing namespace of the spawned job (str)

See **pmix_server_spawn_fn_t** for details

A.3.3.10 Connect

Summary

Record the specified processes as *connected*.

Format

PMIx v4.0

Python

```
def connect(procs:list, directives:list)
```

Python

IN **procs**
List of Python **proc** dictionaries identifying participants (list)

IN **directives**
List of Python **info** directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_connect_fn_t** for details

A.3.3.11 Disconnect

Summary

Disconnect a previously connected set of processes.

Format

PMIx v4.0

Python

```
def disconnect(procs:list, directives:list)
```

Python

IN **procs**
List of Python **proc** dictionaries identifying participants (list)

IN **directives**
List of Python **info** directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_disconnect_fn_t** for details

A.3.3.12 Register Events

Summary

Register to receive notifications for the specified events.

Format

PMIx v4.0

Python

```
def register_events(codes:list, directives:list)
```

Python

IN **codes**
List of Python integers (list)

IN **directives**
List of Python **info** directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_register_events_fn_t** for details

A.3.3.13 Deregister Events

Summary

Deregister to receive notifications for the specified events.

Format

PMIx v4.0

Python

```
def deregister_events(codes:list)
```

Python

IN **codes**
List of Python integers (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_deregister_events_fn_t** for details

A.3.3.14 Notify Event

Summary

Notify the specified range of processes of an event.

Format

PMIx v4.0

Python

```
def notify_event(code:integer, source:dict, range:integer, directives:list)
```

Python

IN **code**
Python integer **pmix_status_t** (list)

IN **source**
Python **proc** of process that generated the event (dict)

IN **range**
Python **range** in which the event is to be reported (integer)

IN **directives**
List of Python **info** directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_notify_event_fn_t** for details

A.3.3.15 Query

Summary

Query information from the resource manager.

Format

PMIx v4.0

Python

```
def query(proc:dict, queries:list)
```

Python

IN **`proc`**
Python **`proc`** of requesting process (dict)

IN **`queries`**
List of Python **`query`** directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)
- *info* - List of Python **`info`** containing the returned results (list)

See **`pmix_server_query_fn_t`** for details

A.3.3.16 Tool Connected

Summary

Register that a tool has connected to the server.

Format

PMIx v4.0

Python

```
def tool_connected(info:list)
```

Python

IN **`info`**
List of Python **`info`** containing info on the connecting tool (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)
- *proc* - Python **`proc`** containing the assigned namespace:rank for the tool (dict)

See **`pmix_server_tool_connection_fn_t`** for details

A.3.3.17 Log

Summary

Log data on behalf of a client.

Format

PMIx v4.0

Python

```
def log(proc:dict, data:list, directives:list)
```

Python

IN **proc**
Python **proc** of requesting process (dict)

IN **data**
List of Python **info** containing data to be logged (list)

IN **directives**
List of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_log_fn_t** for details

A.3.3.18 Allocate Resources

Summary

Request allocation operations on behalf of a client.

Format

PMIx v4.0

Python

```
def allocate(proc:dict, action:integer, directives:list)
```

Python

IN **proc**
Python **proc** of requesting process (dict)

IN **action**
Python **allocdir** specifying requested action (integer)

IN **directives**
List of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python **info** containing results of requested operation (list)

See **pmix_server_alloc_fn_t** for details

A.3.3.19 Job Control

Summary

Execute a job control action on behalf of a client.

Format

PMIx v4.0

Python

```
def job_control(proc:dict, targets:list, directives:list)
```

Python

IN **`proc`**
Python **`proc`** of requesting process (dict)

IN **`targets`**
List of Python **`proc`** specifying target processes (list)

IN **`directives`**
List of Python **`info`** containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)

See **`pmix_server_job_control_fn_t`** for details

A.3.3.20 Monitor

Summary

Request that a client be monitored for activity.

Format

PMIx v4.0

Python

```
def monitor(proc:dict, request:list, directives:list)
```

Python

IN **`proc`**
Python **`proc`** of requesting process (dict)

IN **`request`**
List of Python **`info`** specifying requested monitoring operations (list)

IN **`directives`**
List of Python **`info`** containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)

See **`pmix_server_monitor_fn_t`** for details

A.3.3.21 Get Credential

Summary

Request a credential from the host environment

Format

PMIx v4.0

Python

```
def get_credential(proc:dict, directives:list)
```

Python

IN **`proc`**
Python **`proc`** of requesting process (dict)

IN **`directives`**
List of Python **`info`** containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)
- *cred* - Python **`byteobject`** containing returned credential (dict)
- *info* - List of Python **`info`** containing any additional info about the credential (list)

See **`pmix_server_get_cred_fn_t`** for details

A.3.3.22 Validate Credential

Summary

Request validation of a credential

Format

PMIx v4.0

Python

```
def validate_credential(proc:dict, cred:dict, directives:list)
```

Python

IN **`proc`**
Python **`proc`** of requesting process (dict)

IN **`cred`**
Python **`byteobject`** containing credential (dict)

IN **`directives`**
List of Python **`info`** containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a PMIx error code indicating the operation failed (integer)
- *info* - List of Python **`info`** containing any additional info from the credential (list)

See **`pmix_server_validate_cred_fn_t`** for details

A.3.3.23 IO Forward

Summary

Request the specified IO channels be forwarded from the given array of processes.

Format

PMIx v4.0

Python

```
def iof_pull(sources:list, channels:integer, directives:list)
```

Python

IN **sources**
List of Python **proc** whose IO is being requested (list)

IN **channels**
Bitmask of Python **channel** identifying IO channels to be forwarded (integer)

IN **directives**
List of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_iof_fn_t** for details

A.3.3.24 IO Push

Summary

Pass standard input data to the host environment for transmission to specified recipients.

Format

PMIx v4.0

Python

```
def iof_push(source:dict, targets:list, directives:list)
```

Python

IN **source**
Python **proc** whose stdin data is being provided (dict)

IN **targets**
List of Python **proc** identifying targets to receive the provided data (list)

IN **directives**
List of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

See **pmix_server_stdin_fn_t** for details

A.3.3.25 Group Operations

Summary

Request group operations (construct, destruct, etc.) on behalf of a set of processes.

Format

PMIx v4.0

Python

```
def group(op:integer, grp:str, procs:list, directives:list)
```

Python

IN **op**
Operation host is to perform on the specified group (integer)

IN **grp**
String identifier of target group (str)

IN **procs**
List of Python **proc** of participating processes (dict)

IN **directives**
List of Python **info** containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python **info** containing results of requested operation (list)

See **pmix_server_grp_fn_t** for details

A.4 PMIxClient

The client Python class is by far the richest in terms of APIs as it houses all the APIs that an application might utilize. Due to the datatype translation requirements of the C-Python interface, only the blocking form of each API is supported – providing a Python callback function directly to the C interface underlying the bindings was not a supportable option.

A.4.1 Client.init

Summary

Initialize the PMIx client library after obtaining a new PMIxClient object

Format

PMIx v4.0

Python

```
rc, proc = myclient.init(info:list)
```

Python

IN **info**
List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *proc* - a Python **proc** dictionary (dict)

See **PMIx_Init** for description of all relevant attributes and behaviors

A.4.2 Client.initialized

Format

PMIx v4.0

Python

```
rc = myclient.initialized()
```

Python

Returns:

- *rc* - a value of **1** (true) will be returned if the PMIx library has been initialized, and **0** (false) otherwise (integer)

See **PMIx_Initialized** for description of all relevant attributes and behaviors

A.4.3 Client.get_version

Format

PMIx v4.0

Python

```
vers = myclient.get_version()
```

Python

Returns:

- *vers* - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)

See **PMIx_Get_version** for description of all relevant attributes and behaviors

A.4.4 Client.finalize

Summary

Finalize the PMIx client library.

Format

PMIx v4.0

Python

```
rc = myclient.finalize(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_Finalize** for description of all relevant attributes and behaviors

A.4.5 Client.abort

Summary

Request that the provided list of procs be aborted

Format

PMIx v4.0

Python

```
rc = myclient.abort(status:integer, msg:str, targets:list)
```

Python

IN **`status`**
PMIx status to be returned on exit (integer)

IN **`msg`**
String message to be printed (string)

IN **`targets`**
List of Python **`proc`** dictionaries (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_Abort`** for description of all relevant attributes and behaviors

A.4.6 Client.store_internal

Summary

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

Format

PMIx v4.0

Python

```
rc = myclient.store_internal(proc:dict, key:str, value:dict)
```

Python

IN **`proc`**
Python **`proc`** dictionary of the process being referenced (dict)

IN **`key`**
String key of the data (string)

IN **`value`**
Python **`value`** dictionary (dict)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_Store_internal`** for details

A.4.7 Client.put

Summary

Push a key/value pair into the client's namespace.

Format

PMIx v4.0

Python

```
rc = myclient.put(scope:integer, key:str, value:dict)
```

Python

IN **scope**
Scope of the data being posted (integer)

IN **key**
String key of the data (string)

IN **value**
Python **value** dictionary (dict)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Put** for description of all relevant attributes and behaviors

A.4.8 Client.commit

Summary

Push all previously **PMIxClient.put** values to the local PMIx server.

Format

PMIx v4.0

Python

```
rc = myclient.commit()
```

Python

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Commit** for description of all relevant attributes and behaviors

A.4.9 Client.fence

Summary

Execute a blocking barrier across the processes identified in the specified list

Format

PMIx v4.0

Python

```
rc = myclient.fence(peers:list, directives:list)
```

Python

IN **peers**
List of Python **proc** dictionaries (list)

IN **directives**
List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Fence** for description of all relevant attributes and behaviors

A.4.10 Client.get

Summary

Retrieve a key/value pair

Format

PMIx v4.0

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

Publish data for later access via **PMIx_Lookup** .

Format

PMIx v4.0

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

A.4.12 Client.lookup

Summary

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

Format

PMIx v4.0

Python

```
rc,info = myclient.lookup(pdata:list, directives:list)
```

Python

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

A.4.13 Client.unpublish

Summary

Delete data published by this process with **`PMIx_Publish`** .

Format

PMIx v4.0

Python

```
rc = myclient.unpublish(keys:list, directives:list)
```

Python

IN **`keys`**
List of Python string keys identifying data to be deleted (list)

IN **`directives`**
List of Python **`info`** dictionaries (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_Unpublish`** for description of all relevant attributes and behaviors

A.4.14 Client.spawn

Summary

Spawn a new job.

Format

PMIx v4.0

Python

```
rc,nspace = myclient.spawn(jobinfo:list, apps:list)
```

Python

IN **`jobinfo`**
List of Python **`info`** dictionaries (list)

IN **`apps`**
List of Python **`app`** dictionaries (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)
- *nspace* - Python **`nspace`** of the new job (dict)

See **`PMIx_Spawn`** for description of all relevant attributes and behaviors

A.4.15 Client.connect

Summary

Connect namespaces.

Format

PMIx v4.0

Python

```
rc = myclient.connect(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_Connect** for description of all relevant attributes and behaviors

A.4.16 Client.disconnect

Summary

Disconnect namespaces.

Format

PMIx v4.0

Python

```
rc = myclient.disconnect(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_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.

Format

PMIx v4.0

Python

```
rc,procs = myclient.resolve_peers(node:str, nspace:str)
```

Python

IN **node**
Name of node whose processes are being requested (str)

IN **nspace**
Python **nspace** whose processes are to be returned (str)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *procs* - List of Python **proc** dictionaries (list)

See **PMIx_Resolve_peers** for description of all relevant attributes and behaviors

A.4.18 Client.resolve_nodes

Summary

Return list of nodes hosting processes within the specified **nspace** .

Format

PMIx v4.0

Python

```
rc,nodes = myclient.resolve_nodes(nspace:str)
```

Python

IN **nspace**
Python **nspace** (str)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *nodes* - List of Python string node names (list)

See **PMIx_Resolve_nodes** for description of all relevant attributes and behaviors

A.4.19 Client.query

Summary

Query information about the system in general

Format

PMIx v4.0

Python

```
rc,info = myclient.query(queries:list, directives:list)
```

Python

IN **`queries`**
List of Python **`query`** 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)
- *info* - List of Python **`info`** containing results of the query (list)

See **`PMIx_Query_info_nb`** for description of all relevant attributes and behaviors

A.4.20 Client.log

Summary

Log data to a central data service/store

Format

PMIx v4.0

Python

```
rc = myclient.log(data:list, directives:list)
```

Python

IN **`data`**
List of Python **`info`** 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_Log`** for description of all relevant attributes and behaviors

A.4.21 Client.allocate

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v4.0

Python

```
rc,info = myclient.allocate(request:integer, directives:list)
```

Python

IN `request`
Python `allocdir` specifying requested operation (integer)

IN `directives`
List of Python `info` dictionaries describing request (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python `info` containing results of the request (list)

See `PMIx_Allocation_request_nb` for description of all relevant attributes and behaviors

A.4.22 Client.job_ctrl

Summary

Request a job control action

Format

PMIx v4.0

Python

```
rc,info = myclient.job_ctrl(targets:list, directives:list)
```

Python

IN `targets`
List of Python `proc` specifying targets of requested operation (integer)

IN `directives`
List of Python `info` dictionaries 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_Job_control_nb` for description of all relevant attributes and behaviors

A.4.23 Client.monitor

Summary

Request that something be monitored

Format

PMIx v4.0

Python

```
rc,info = myclient.monitor(targets:list, directives:list)
```

Python

IN `targets`
List of Python `proc` specifying targets of requested operation (integer)

IN `directives`
List of Python `info` dictionaries describing request (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python `info` containing results of the request (list)

See `PMIx_Process_monitor_nb` for description of all relevant attributes and behaviors

A.4.24 Client.get_credential

Summary

Request a credential from the PMIx server/SMS

Format

PMIx v4.0

Python

```
rc,cred,info = myclient.get_credential(directives:list)
```

Python

IN `directives`
List of Python `info` dictionaries describing request (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- *cred* - Python `byteobject` containing returned credential (dict)
- *info* - List of Python `info` containing results of the request (list)

See `PMIx_Get_credential` for description of all relevant attributes and behaviors

A.4.25 Client.validate_credential

Summary

Request validation of a credential by the PMIx server/SMS

Format

PMIx v4.0

Python

```
rc,info = myclient.validate_credential(cred:dict, directives:list)
```

Python

IN **`cred`**
Python **`byteobject`** containing credential (dict)

IN **`directives`**
List of Python **`info`** dictionaries describing request (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **`info`** containing additional results of the request (list)

See **`PMIx_Validate_credential`** for description of all relevant attributes and behaviors

A.4.26 Client.group_construct

Summary

Construct a new group composed of the specified processes and identified with the provided group identifier

Format

PMIx v4.0

Python

```
rc,info = myclient.construct_group(grp:string, members:list, directives:list
```

Python

IN **`grp`**
Python string identifier for the group (str)

IN **`members`**
List of Python **`proc`** dictionaries identifying group members (list)

IN **`directives`**
List of Python **`info`** dictionaries describing request (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **`info`** containing results of the request (list)

See **`PMIx_Group_construct`** for description of all relevant attributes and behaviors

A.4.27 Client.group_invite

Summary

Explicitly invite specified processes to join a group

Format

PMIx v4.0

Python

```
rc,info = myclient.group_invite(grp:string, members:list, directives:list)
```

Python

IN **grp**
Python string identifier for the group (str)

IN **members**
List of Python **proc** dictionaries identifying processes to be invited (list)

IN **directives**
List of Python **info** dictionaries describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **info** containing results of the request (list)

See **PMIx_Group_invite** for description of all relevant attributes and behaviors

A.4.28 Client.group_join

Summary

Respond to an invitation to join a group that is being asynchronously constructed

Format

PMIx v4.0

Python

```
rc,info = myclient.group_join(grp:string, leader:dict, opt:integer, directiv
```

Python

IN **grp**
Python string identifier for the group (str)

IN **leader**
Python **proc** dictionary identifying process leading the group (dict)

IN **opt**
One of the **pmix_group_opt_t** values indicating decline/accept (integer)

IN **directives**
List of Python **info** dictionaries describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - List of Python **info** containing results of the request (list)

See **PMIx_Group_join** for description of all relevant attributes and behaviors

A.4.29 Client.group_leave

Summary

Leave a PMIx Group

Format

PMIx v4.0

Python

```
rc = myclient.group_leave(grp:string, directives:list)
```

Python

IN **grp**
Python string identifier for the group (str)

IN **directives**
List of Python **info** dictionaries describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Group_leave** for description of all relevant attributes and behaviors

A.4.30 Client.group_destruct

Summary

Destruct a PMIx Group

Format

PMIx v4.0

Python

```
rc = myclient.group_destruct(grp:string, directives:list)
```

Python

IN **grp**
Python string identifier for the group (str)

IN **directives**
List of Python **info** dictionaries describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Group_destruct** for description of all relevant attributes and behaviors

A.4.31 Client.register_event_handler

Summary

Register an event handler to report events.

Format

PMIx v4.0

Python

```
rc,id = myclient.register_event_handler(codes:list, directives:list, cbfunc)
```

Python

IN **codes**
List of Python integer status codes that should be reported to this handler (llist)

IN **directives**
List of Python **info** dictionaries describing request (list)

IN **cbfunc**
Python **evhandler** to be called when event is received (func)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *id* - PMIx reference identifier for handler (integer)

See **PMIx_Register_event_handler** for description of all relevant attributes and behaviors

A.4.32 Client.deregister_event_handler

Summary

Deregister an event handler

Format

PMIx v4.0

Python

```
myclient.deregister_event_handler(id:integer)
```

Python

IN **id**
PMIx reference identifier for handler (integer)

Returns: None

See **PMIx_Deregister_event_handler** for description of all relevant attributes and behaviors

A.4.33 Client.notify_event

Summary

Report an event for notification via any registered handler.

Format

PMIx v4.0

Python

```
rc = myclient.notify_event(status:integer, source:dict,
                           range:integer, directives:list)
```

Python

IN **status**
PMIx status code indicating the event being reported (integer)

IN **source**
Python **proc** of the process that generated the event (dict)

IN **range**
Python **range** in which the event is to be reported (integer)

IN **directives**
List of Python **info** dictionaries describing request (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Notify_event** for description of all relevant attributes and behaviors

A.4.34 Client.error_string

Summary

Pretty-print string representation of **pmix_status_t** .

Format

PMIx v4.0

Python

```
rep = myclient.error_string(status:integer)
```

Python

IN **status**
PMIx status code (integer)

Returns:

- *rep* - String representation of the provided status code (str)

See **PMIx_Error_string** for further details

A.4.35 Client.proc_state_string

Summary

Pretty-print string representation of **pmix_proc_state_t** .

Format

PMIx v4.0

Python

```
rep = myclient.proc_state_string(state:integer)
```

Python

IN `state`
PMIx process state code (integer)

Returns:

- *rep* - String representation of the provided process state (str)

See `PMIx_Proc_state_string` for further details

A.4.36 Client.scope_string

Summary

Pretty-print string representation of `pmix_scope_t` .

Format

PMIx v4.0

Python

```
rep = myclient.scope_string(scope:integer)
```

Python

IN `scope`
PMIx scope value (integer)

Returns:

- *rep* - String representation of the provided scope (str)

See `PMIx_Scope_string` for further details

A.4.37 Client.persistence_string

Summary

Pretty-print string representation of `pmix_persistence_t` .

Format

PMIx v4.0

Python

```
rep = myclient.persistence_string(persistence:integer)
```

Python

IN `persistence`
PMIx persistence value (integer)

Returns:

- *rep* - String representation of the provided persistence (str)

See `PMIx_Persistence_string` for further details

A.4.38 Client.data_range_string

Summary

Pretty-print string representation of **pmix_data_range_t** .

Format

PMIx v4.0

Python

```
rep = myclient.data_range_string(range:integer)
```

Python

IN **range**
PMIx data range value (integer)

Returns:

- *rep* - String representation of the provided data range (str)

See **PMIx_Data_range_string** for further details

A.4.39 Client.info_directives_string

Summary

Pretty-print string representation of **pmix_info_directives_t** .

Format

PMIx v4.0

Python

```
rep = myclient.info_directives_string(directives:integer)
```

Python

IN **directives**
PMIx info directives value (integer)

Returns:

- *rep* - String representation of the provided info directives (str)

See **PMIx_Info_directives_string** for further details

A.4.40 Client.data_type_string

Summary

Pretty-print string representation of **pmix_data_type_t** .

Format

PMIx v4.0

Python

```
rep = myclient.data_type_string(dtype:integer)
```

Python

IN **`dtype`**
PMIx datatype value (integer)

Returns:

- *rep* - String representation of the provided datatype (str)

See **`PMIx_Data_type_string`** for further details

A.4.41 Client.alloc_directive_string

Summary

Pretty-print string representation of **`pmix_alloc_directive_t`** .

Format

PMIx v4.0

Python

```
rep = myclient.alloc_directive_string(adir:integer)
```

Python

IN **`adir`**
PMIx allocation directive value (integer)

Returns:

- *rep* - String representation of the provided allocation directive (str)

See **`PMIx_Alloc_directive_string`** for further details

A.4.42 Client.iof_channel_string

Summary

Pretty-print string representation of **`pmix_iof_channel_t`** .

Format

PMIx v4.0

Python

```
rep = myclient.iof_channel_string(channel:integer)
```

Python

IN **`channel`**
PMIx IOF channel value (integer)

Returns:

- *rep* - String representation of the provided IOF channel (str)

See **`PMIx_IOF_channel_string`** for further details

A.5 PMIxServer

The server Python class inherits the Python "client" class as its parent. Thus, it includes all client functions in addition to the ones defined in this section.

A.5.1 Server.init

Summary

Initialize the PMIx server library after obtaining a new PMIxServer object

Format

PMIx v4.0

Python

```
rc = myserver.init(directives:list, map:dict)
```

Python

IN `directives`
List of Python `info` dictionaries (list)

IN `map`
Python dictionary key-function pairs that map `server module` callback functions to provided implementations (dict)

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_server_init` for description of all relevant attributes and behaviors

A.5.2 Server.finalize

Summary

Finalize the PMIx server library

Format

PMIx v4.0

Python

```
rc = myserver.finalize()
```

Python

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_server_finalize` for details

A.5.3 Server.generate_regex

Summary

Generate a regular expression representation of the input strings.

Format

PMIx v4.0

Python

```
rc,regex = myserver.generate_regex(input:list)
```

Python

IN **input**
List of Python strings (e.g., node names) (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *regex* - Python **bytearray** containing regular expression representation of the input list (**bytearray**)

See **PMIx_generate_regex** for details

A.5.4 Server.generate_ppn

Summary

Generate a regular expression representation of the input strings.

Format

PMIx v4.0

Python

```
rc,regex = myserver.generate_ppn(input:list)
```

Python

IN **input**
List of Python strings describing the ranks on each node (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *regex* - Python **bytearray** containing regular expression representation of the input list (**bytearray**)

See **PMIx_generate_ppn** for details

A.5.5 Server.register_nspace

Summary

Setup the data about a particular namespace.

Format

PMIx v4.0

Python

```
rc = myserver.register_nspace(nspace:str,
                              nlocalprocs:integer,
                              directives:list)
```

Python

IN **nspace**
Python string containing the namespace (str)

IN **nlocalprocs**
Number of local processes (integer)

IN **directives**
List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_server_register_nspace** for description of all relevant attributes and behaviors

A.5.6 Server.deregister_nspace

Summary

Deregister a namespace.

Format

PMIx v4.0

Python

```
myserver.deregister_nspace(nspace:str)
```

Python

IN **nspace**
Python string containing the namespace (str)

Returns: None

See **PMIx_server_deregister_nspace** for details

A.5.7 Server.register_client

Summary

Register a client process with the PMIx server library.

Format

PMIx v4.0

Python

```
rc = myserver.register_client(proc:dict, uid:integer, gid:integer)
```

Python

IN **`proc`**
Python **`proc`** dictionary identifying the client process (dict)

IN **`uid`**
Linux uid value for user executing client process (integer)

IN **`gid`**
Linux gid value for user executing client process (integer)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_server_register_client`** for details

A.5.8 Server.deregister_client

Summary

Dergister a client process and purge all data relating to it

Format

PMIx v4.0

Python

```
myserver.deregister_client(proc:dict)
```

Python

IN **`proc`**
Python **`proc`** dictionary identifying the client process (dict)

Returns: None

See **`PMIx_server_deregister_client`** for details

A.5.9 Server.setup_fork

Summary

Setup the environment of a child process that is to be forked by the host

Format

PMIx v4.0

Python

```
rc = myserver.setup_fork(proc:dict, envin:dict)
```

Python

IN **`proc`**
Python **`proc`** dictionary identifying the client process (dict)

INOUT **`envin`**
Python dictionary containing the environment to be passed to the client (dict)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_server_setup_fork`** for details

A.5.10 Server.dmodex_request

Summary

Function by which the host server can request modex data from the local PMIx server.

Format

PMIx v4.0

Python

```
rc,data = myserver.dmodex_request(proc:dict)
```

Python

IN **`proc`**
Python **`proc`** dictionary identifying the process whose data is requested (dict)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)
- *data* - Python **`byteobject`** containing the returned data (dict)

See **`PMIx_server_dmodex_request`** for details

A.5.11 Server.setup_application

Summary

Function by which the resource manager can request application-specific setup data prior to launch of a **`job`** .

Format

PMIx v4.0

Python

```
rc,info = myserver.setup_application(nspace:str, directives:list)
```

Python

IN **`nspace`**
Namespace whose setup information is being requested (str)

IN **`directives`**
Python list of **`info`** directives

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)
- *info* - Python list of **`info`** dictionaries containing the returned data (list)

See **`PMIx_server_setup_application`** for details

A.5.12 Server.register_attributes

Summary

Register host environment attribute support for a function.

Format

PMIx v4.0

Python

```
rc = myserver.register_attributes(function:str, attrs:list)
```

Python

IN **`function`**
Name of the function (str)

IN **`attrs`**
Python list of **`regattr`** dictionaries describing the supported attributes

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_Register_attributes`** for details

A.5.13 Server.setup_local_support

Summary

Function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application

Format

PMIx v4.0

Python

```
rc = myserver.setup_local_support(nspace:str, info:list)
```

Python

IN **`nspace`**
Namespace whose setup information is being requested (str)

IN **`info`**
Python list of **`info`** dictionaries containing the setup data (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_server_setup_local_support`** for details

A.5.14 Server.iof_deliver

Summary

Function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

Format

PMIx v4.0

Python

```
rc = myserver.iof_deliver(source:dict, channel:integer,
                          data:dict, directives:list)
```

Python

IN **`source`**
Python **`proc`** dictionary identifying the process who generated the data (dict)

IN **`channel`**
Python **`channel`** bitmask identifying IO channel of the provided data (integer)

IN **`data`**
Python **`byteobject`** containing the data (dict)

IN **`directives`**
Python list of **`info`** dictionaries containing directives (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_server_IOF_deliver`** for details

A.5.15 Server.collect_inventory

Summary

Collect inventory of resources on a node

Format

PMIx v4.0

Python

```
rc,info = myserver.collect_inventory(directives:list)
```

Python

IN **directives**
Python list of **info** dictionaries containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *info* - Python list of **info** dictionaries containing the returned data (list)

See **PMIx_server_collect_inventory** for details

A.5.16 Server.deliver_inventory

Summary

Pass collected inventory to the PMIx server library for storage

Format

PMIx v4.0

Python

```
rc = myserver.deliver_inventory(info:list, directives:list)
```

Python

IN **info**
- Python list of **info** dictionaries containing the inventory data (list)

IN **directives**
Python list of **info** dictionaries containing directives (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_server_deliver_inventory** for details

A.6 PMIxTool

The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and server functions in addition to the ones defined in this section.

A.6.1 Tool.init

Summary

Initialize the PMIx tool library after obtaining a new PMIxTool object

Format

PMIx v4.0

Python

```
rc,proc = mytool.init(info:list)
```

Python

IN `info`
List of Python **`info`** dictionaries (list)

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)
- *proc* - a Python **`proc`** dictionary (dict)

See **`PMIx_tool_init`** for description of all relevant attributes and behaviors

A.6.2 Tool.finalize

Summary

Finalize the PMIx tool library, closing the connection to the server.

Format

PMIx v4.0

Python

```
rc = mytool.finalize()
```

Python

Returns:

- *rc* - **`PMIX_SUCCESS`** or a negative value corresponding to a PMIx error constant (integer)

See **`PMIx_tool_finalize`** for description of all relevant attributes and behaviors

A.6.3 Tool.connect_to_server

Summary

Switch connection from the current PMIx server to another one, or initialize a connection to a specified server.

Format

PMIx v4.0

Python

```
rc,proc = mytool.connect_to_server(info:list)
```

Python

IN **info**
List of Python **info** dictionaries (list)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *proc* - a Python **proc** dictionary (dict)

See **PMIx_tool_connect_to_server** for description of all relevant attributes and behaviors

A.6.4 Tool.iof_pull

Summary

Register to receive output forwarded from a remote process.

Format

PMIx v4.0

Python

```
rc,id = mytool.iof_pull(sources:list, channel:integer, directives:list, cbfu
```

Python

IN **sources**
List of Python **proc** dictionaries of processes whose IO is being requested (list)
IN **channel**
Python **channel** bitmask identifying IO channels to be forwarded (integer)
IN **directives**
List of Python **info** dictionaries describing request (list)
IN **cbfunc**
Python **iofcbfunc** to receive IO payloads (func)

Returns:

- *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- *id* - PMIx reference identifier for request (integer)

See **PMIx_IOF_pull** for description of all relevant attributes and behaviors

A.6.5 Tool.iof_deregister

Summary

Deregister from output forwarded from a remote process.

Format

PMIx v4.0

Python

```
rc = mytool.iof_deregister(id:integer, directives:list)
```

Python

IN `id`
PMIx reference identifier returned by pull request (list)

IN `directives`
List of Python `info` dictionaries describing request (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_IOF_deregister` for description of all relevant attributes and behaviors

A.6.6 Tool.iof_push

Summary

Push data collected locally (typically from stdin) to stdin of target recipients

Format

PMIx v4.0

Python

```
rc = mytool.iof_push(targets:list, data:dict, directives:list)
```

Python

IN `sources`
List of Python `proc` dictionaries of target processes (list)

IN `data`
Python `byteobject` dictionary containing data to be delivered (dict)

IN `directives`
List of Python `info` dictionaries describing request (list)

Returns:

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_IOF_push` for description of all relevant attributes and behaviors

A.7 Example Usage

The following examples are provided to illustrate the use of the Python bindings.

A.7.1 Python Client

The following example contains a client program that illustrates a fairly common usage pattern. The program instantiates and initializes the PMIxClient class, posts some data that is to be shared across all processes in the job, executes a "fence" that circulates the data, and then retrieves a value posted by one of its peers. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *

def main():
    # Instantiate a client object
    myclient = PMIxClient()
    print("Testing PMIx ", myclient.get_version())

    # Initialize the PMIx client library, declaring the programming model
    # as "TEST" and the library name as "PMIX", just for the example
    info = ['key':PMIX_PROGRAMMING_MODEL,
             'value':'TEST', 'val_type':PMIX_STRING,
            'key':PMIX_MODEL_LIBRARY_NAME,
             'value':'PMIX', 'val_type':PMIX_STRING]
    rc,myname = myclient.init(info)
    if PMIX_SUCCESS != rc:
        print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
        exit(1)

    # try posting a value
    rc = myclient.put(PMIX_GLOBAL, "mykey",
                      'value':1, 'val_type':PMIX_INT32)
    if PMIX_SUCCESS != rc:
        print("PMIx_Put FAILED WITH ERROR", myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)

    # commit it
    rc = myclient.commit()
    if PMIX_SUCCESS != rc:
        print("PMIx_Commit FAILED WITH ERROR",
              myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)
```

```
# execute fence across all processes in my job
procs = []
info = []
rc = myclient.fence(procs, info)
if PMIX_SUCCESS != rc:
    print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
    # cleanly finalize
    myclient.finalize()
    exit(1)

# Get a value from a peer
if 0 != myname['rank']:
    info = []
    rc, get_val = myclient.get('nspace':"testnspace", 'rank': 0,
                               "mykey", info)
    if PMIX_SUCCESS != rc:
        print("PMIx_Commit FAILED WITH ERROR",
              myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)
    print("Get value returned: ", get_val)

# test a fence that should return not_supported because
# we pass a required attribute that the server is known
# not to support
procs = []
info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
         'value':10, 'val_type':PMIX_INT]
rc = myclient.fence(procs, info)
if PMIX_SUCCESS == rc:
    print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
    # cleanly finalize
    myclient.finalize()
    exit(1)

# Publish something
info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
rc = myclient.publish(info)
if PMIX_SUCCESS != rc:
    print("PMIx_Publish FAILED WITH ERROR",
          myclient.error_string(rc))
    # cleanly finalize
```

```
            myclient.finalize()
            exit(1)

        # finalize
        info = []
        myclient.finalize(info)
        print("Client finalize complete")

# Python main program entry point
if __name__ == '__main__':
    main()
```

Python

A.7.2 Python Server

The following example contains a minimum-level server host program that instantiates and initializes the PMIxServer class. The program illustrates passing several server module functions to the bindings and includes code to setup and spawn a simple client application, waiting until the spawned client terminates before finalizing and exiting itself. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *
import signal, time
import os
import select
import subprocess

def clientconnected(proc:tuple is not None):
    print("CLIENT CONNECTED", proc)
    return PMIX_OPERATION_SUCCEEDED

def clientfinalized(proc:tuple is not None):
    print("CLIENT FINALIZED", proc)
    return PMIX_OPERATION_SUCCEEDED

def clientfence(procs:list, directives:list, data:bytearray):
    # check directives
    if directives is not None:
        for d in directives:
            # these are each an info dict
            if "pmix" not in d['key']:
                # we do not support such directives - see if
```

```
                    # it is required
                    try:
                        if d['flags'] & PMIX_INFO_REQD:
                            # return an error
                            return PMIX_ERR_NOT_SUPPORTED
                    except:
                        #it can be ignored
                        pass
        return PMIX_OPERATION_SUCCEEDED

    def main():
        try:
            myserver = PMIxServer()
        except:
            print("FAILED TO CREATE SERVER")
            exit(1)
        print("Testing server version ", myserver.get_version())

        args = ['key':PMIX_SERVER_SCHEDULER,
                'value':'T', 'val_type':PMIX_BOOL]
        map = 'clientconnected': clientconnected,
              'clientfinalized': clientfinalized,
              'fencenb': clientfence
        my_result = myserver.init(args, map)

        # get our environment as a base
        env = os.environ.copy()

        # register an nspace for the client app
        (rc, regex) = myserver.generate_regex("test000,test001,test002")
        (rc, ppn) = myserver.generate_ppn("0")
        kvals = ['key':PMIX_NODE_MAP,
                 'value':regex, 'val_type':PMIX_STRING,
                'key':PMIX_PROC_MAP,
                 'value':ppn, 'val_type':PMIX_STRING,
                'key':PMIX_UNIV_SIZE,
                 'value':1, 'val_type':PMIX_UINT32,
                'key':PMIX_JOB_SIZE,
                 'value':1, 'val_type':PMIX_UINT32]
        rc = foo.register_nspace("testnspace", 1, kvals)
        print("RegNspace ", rc)

        # register a client
```

```
uid = os.getuid()
gid = os.getgid()
rc = myserver.register_client('nspace':"testnspace", 'rank':0,
                              uid, gid)
print("RegClient ", rc)
# setup the fork
rc = myserver.setup_fork('nspace':"testnspace", 'rank':0, env)
print("SetupFrk", rc)

# setup the client argv
args = ["./client.py"]
# open a subprocess with stdout and stderr
# as distinct pipes so we can capture their
# output as the process runs
p = subprocess.Popen(args, env=env,
    stdout=subprocess.PIPE, stderr=subprocess.PIPE)
# define storage to catch the output
stdout = []
stderr = []
# loop until the pipes close
while True:
    reads = [p.stdout.fileno(), p.stderr.fileno()]
    ret = select.select(reads, [], [])

    stdout_done = True
    stderr_done = True

    for fd in ret[0]:
        # if the data
        if fd == p.stdout.fileno():
            read = p.stdout.readline()
            if read:
                read = read.decode('utf-8').rstrip()
                print('stdout: ' + read)
                stdout_done = False
        elif fd == p.stderr.fileno():
            read = p.stderr.readline()
            if read:
                read = read.decode('utf-8').rstrip()
                print('stderr: ' + read)
                stderr_done = False

    if stdout_done and stderr_done:
```

```
                break
        print("FINALIZING")
        myserver.finalize()


if __name__ == '__main__':
        main()
```

Python

APPENDIX B

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.

B.1 Version 3.0

The following list includes some of the active participants in the PMIx v3 standardization process.

- Ralph H. Castain, Andrew Friedley, Brandon Yates
- Joshua Hursey
- Aurelien Bouteiller and George Bosilca
- Dirk Schubert
- Kevin Harms

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- The Exascale Computing Project, an initiative of the US Department of Energy
- National Science Foundation
- Argonne National Laboratory
- Allinea (ARM)

B.2 Version 2.0

The following list includes some of the active participants in the PMIx v2 standardization process.

- Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm and Terry Wilmarth
- Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
- Aurelien Bouteiller and George Bosilca
- Artem Polyakov, Igor Ivanov and Boris Karasev
- Gilles Gouaillardet
- Michael A Raymond and Jim Stoffel
- Dirk Schubert
- Moe Jette
- Takahiro Kawashima and Shinji Sumimoto
- Howard Pritchard
- David Beer
- Brice Goglin
- Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
- Adam Moody and Martin Schulz
- Ryan Grant and Stephen Olivier
- Michael Karo

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
- 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

B.3 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
- Adaptive Solutions, Inc.
- 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.

Index

General terms and other items not induced in the other indices.

application, 10, 12, **15**, 75, 76, 133, 191, 254, 256

host environment, **16**

job, 10, 12, **15**, 75–77, 133, 135, 191, 248, 249, 253, 254, 256, 266, 268, 412

namespace, **15**
network plane, **16**, 332
network planes, **16**, 364

rank, **15**, 135, 257
resource manager, **16**

scheduler, **16**, 330
session, 10, 12, **15**, 75, 76, 133, 191, 253
slot, **15**
slots, **15**

workflow, **15**
workflows, **15**, 107

Index of APIs

PMIx_Abort, 8, 32, **154**, 155, 279, 280, 377, 390
    PMIxClient.abort (Python), 390
PMIx_Alloc_directive_string, 9, **111**, 407
    PMIxClient.alloc_directive_string (Python), 407
PMIx_Allocation_request, 11–13, 89, 192, **193**, 198
PMIx_Allocation_request_nb, 9, 89, 180, **196**, 199, 398
    PMIxClient.allocate (Python), 397
PMIx_Commit, 8, 105, 125, 126, **138**, 138, 265, 285, 391
    PMIxClient.commit (Python), 391
PMIx_Connect, 8, 9, 24, 159, **165**, 167, 169, 171, 336–338, 395
    PMIxClient.connect (Python), 394
PMIx_Connect_nb, 8, **167**, 167
pmix_connection_cbfunc_t, **105**, 304
pmix_credential_cbfunc_t, **106**, 239, 319
PMIx_Data_copy, 9, **233**
PMIx_Data_copy_payload, 9, **234**
PMIx_Data_pack, 9, **229**, 230, 246, 247
PMIx_Data_print, 9, **233**
PMIx_Data_range_string, 9, **110**, 406
    PMIxClient.data_range_string (Python), 406
PMIx_Data_type_string, 9, **111**, 407
    PMIxClient.data_type_string (Python), 406
PMIx_Data_unpack, 9, **231**
PMIx_Deregister_event_handler, 9, 13, **221**, 403
    PMIxClient.deregister_event_handler (Python), 403
PMIx_Disconnect, 8, 9, 24, 167, **169**, 171, 173, 338, 395
    PMIxClient.disconnect (Python), 395
PMIx_Disconnect_nb, 8, **171**, 173, 338
pmix_dmodex_response_fn_t, **104**, 265
PMIx_Error_string, 8, **109**, 404
    PMIxClient.error_string (Python), 404
pmix_event_notification_cbfunc_fn_t, **100**, 100, 102
pmix_evhdlr_reg_cbfunc_t, **99**, 99, 219
PMIx_Fence, 3, 6, 8, 14, 124, 125, **138**, 140, 142, 167, 171, 265, 280, 283, 336, 342, 347, 377, 392
    PMIxClient.fence (Python), 391
PMIx_Fence_nb, 8, 12, 97, **140**, 142, 280, 283, 377
PMIx_Finalize, 8, 24, 32, 80, 115, **116**, 116, 164, 278, 279, 376, 389
    PMIxClient.finalize (Python), 389

PMIx_generate_ppn, 8, **246**, 409
PMIxServer.generate_ppn (Python), 409
PMIx_generate_regex, 8, **245**, 247, 253, 409
PMIxServer.generate_regex (Python), 409
PMIx_Get, 3, 8–10, 25, 37, 69–73, 77, 79–91, 115, **126**, 127, 129, 131–135, 137, 157–159, 161–164, 186, 191, 192, 248, 250, 251, 274, 293–295, 335, 342, 364, 392
PMIxClient.get (Python), 392
PMIx_Get_credential, 11, 13, 92, **237**, 320, 399
PMIxClient.get_credential (Python), 399
PMIx_Get_credential_nb, **238**
PMIx_Get_nb, 8, 18, 98, 99, **129**
PMIx_Get_version, 8, 18, **113**, 389
PMIxClient.get_version (Python), 389
PMIx_Group_construct, 337, **339**, 341, 342, 345, 400
PMIxClient.group_construct (Python), 400
PMIx_Group_construct_nb, **342**, 345
PMIx_Group_destruct, 338, **345**, 347, 349, 402
PMIxClient.group_destruct (Python), 402
PMIx_Group_destruct_nb, **347**, 349
PMIx_Group_invite, 337, **349**, 352, 353, 355, 401
PMIxClient.group_invite (Python), 400
PMIx_Group_invite_nb, **353**
PMIx_Group_join, 60, 337, 352, **355**, 355, 357, 358, 360, 401
PMIxClient.group_join (Python), 401
PMIx_Group_join_nb, 355, **358**, 360
PMIx_Group_leave, 338, **360**, 361, 362, 402
PMIxClient.group_leave (Python), 402
PMIx_Group_leave_nb, **361**
pmix_hdlr_reg_cbfunc_t, **109**, 174, 176
pmix_info_cbfunc_t, 95, **99**, 99, 187, 196, 202, 205, 208, 209, 273, 305, 311, 313, 316, 327, 343, 353, 359
PMIx_Info_directives_string, 9, **110**, 406
PMIxClient.info_directives_string (Python), 406
PMIx_Init, 9, 84, 87, 88, **113**, 113, 115, 116, 158, 162, 277, 294, 388
PMIxClient.init (Python), 388
PMIx_Initialized, 8, **112**, 389
PMIxClient.initialized (Python), 389
pmix_iof_cbfunc_t, **108**, 174, 375
PMIx_IOF_channel_string, 11, **111**, 407
PMIxClient.iof_channel_string (Python), 407
PMIx_IOF_deregister, 11, 13, **176**, 418
PMIxTool.iof_deregister (Python), 417
PMIx_IOF_pull, 11, 13, **174**, 176, 417

PMIxTool.iof_pull (Python), 417
PMIx_IOF_push, 11, 13, **177**, 418
PMIxTool.iof_push (Python), 418
PMIx_Job_control, 11, 13, 90, **199**, 201, 204, 205, 315
PMIx_Job_control_nb, 9, 90, 180, 199, **202**, 252, 398
PMIxClient.job_ctrl (Python), 398
PMIx_Log, 11, **210**, 212, 397
PMIxClient.log (Python), 397
PMIx_Log_nb, 9, 86, **212**, 215
PMIx_Lookup, 8, 50, 143, **146**, 149, 150, 392, 393
PMIxClient.lookup (Python), 393
pmix_lookup_cbfunc_t, **98**, 98, 288
PMIx_Lookup_nb, 98, **149**
pmix_modex_cbfunc_t, 95, **96**, 96, 281, 284
pmix_notification_fn_t, **101**, 101, 219, 376
PMIx_Notify_event, 9, 13, **222**, 404
PMIxClient.notify_event (Python), 403
pmix_op_cbfunc_t, **97**, 97, 100, 104, 145, 152, 168, 172, 178, 213, 222, 223, 248, 261–263, 270, 272, 274, 277, 278, 280, 285, 290, 296, 298, 300, 302, 303, 309, 314, 317, 323, 325, 348, 361
PMIx_Persistence_string, 9, **110**, 405
PMIxClient.persistence_string (Python), 405
PMIx_Proc_state_string, 9, **110**, 405
PMIxClient.proc_state_string (Python), 404
PMIx_Process_monitor, 11, 13, **205**, 209
PMIx_Process_monitor_nb, 9, 91, 180, **207**, 209, 399
PMIxClient.monitor (Python), 398
PMIx_Publish, 8, 35, 80, **143**, 144–146, 286, 287, 393
PMIxClient.publish (Python), 392
PMIx_Publish_nb, 8, **145**, 146
PMIx_Put, 8, 34, 35, 37, 105, **125**, 125, 126, 129, 132, 138, 140, 164, 191, 265, 285, 342, 352, 391
PMIxClient.put (Python), 391
PMIx_Query_info, **182**, 186, 191
PMIx_Query_info_nb, 9, 10, 55, 77, 85, 137, 164, 180, **186**, 186, 191, 192, 269, 335, 336, 364, 397
PMIxClient.query (Python), 396
PMIx_Register_attributes, 13, **268**, 413
PMIxServer.register_attributes (Python), 413
PMIx_Register_event_handler, 9, 13, 100, 180, **218**, 403
PMIxClient.register_event_handler (Python), 402
pmix_release_cbfunc_t, **95**, 95
PMIx_Resolve_nodes, 8, **181**, 396
PMIxClient.resolve_nodes (Python), 396
PMIx_Resolve_peers, 8, **180**, 396

PMIxClient.resolve_peers (Python), 395
PMIx_Scope_string, 9, **110**, 405
PMIxClient.scope_string (Python), 405
pmix_server_abort_fn_t, **279**, 377
pmix_server_alloc_fn_t, **310**, 384
pmix_server_client_connected_fn_t, 97, 236, 263, **276**, 277, 376
pmix_server_client_finalized_fn_t, **278**, 279, 376
PMIx_server_collect_inventory, 11, **272**, 415
PMIxServer.collect_inventory (Python), 414
pmix_server_connect_fn_t, 164, **296**, 297, 299, 380
PMIx_server_deliver_inventory, 11, **273**, 415
PMIxServer.deliver_inventory (Python), 415
PMIx_server_deregister_client, 8, **263**, 411
PMIxServer.deregister_client (Python), 411
pmix_server_deregister_events_fn_t, **301**, 382
PMIx_server_deregister_fabric, **332**
PMIx_server_deregister_nspace, 8, **260**, 263, 410
PMIxServer.deregister_nspace (Python), 410
pmix_server_disconnect_fn_t, **298**, 299, 381
pmix_server_dmodex_req_fn_t, 10, 11, 96, **284**, 378
PMIx_server_dmodex_request, 8, 104, **264**, 265, 412
PMIxServer.dmodex_request (Python), 412
pmix_server_fencenb_fn_t, 12, 96, **280**, 283, 377
PMIx_server_finalize, 8, **124**, 408
PMIxServer.finalize (Python), 408
pmix_server_get_cred_fn_t, **319**, 322, 386
PMIx_server_get_index, **333**
PMIx_server_get_vertex_info, **333**
pmix_server_grp_fn_t, **326**, 388
PMIx_server_init, 8, 113, **122**, 269, 275, 408
PMIxServer.init (Python), 408
PMIx_server_IOF_deliver, 11, 173, **271**, 414
PMIxServer.iof_deliver (Python), 414
pmix_server_iof_fn_t, **322**, 387
pmix_server_job_control_fn_t, **313**, 385
pmix_server_listener_fn_t, **304**
pmix_server_log_fn_t, **308**, 384
pmix_server_lookup_fn_t, **287**, 379
pmix_server_module_t, 122, 124, 269, 270, **275**, 275
pmix_server_monitor_fn_t, **316**, 385
pmix_server_notify_event_fn_t, 103, **303**, 382
pmix_server_publish_fn_t, **285**, 378
pmix_server_query_fn_t, **305**, 383

PMIx_server_register_client, 8, 236, **262**, 263, 277, 279, 411
PMIxServer.register_client (Python), 410
pmix_server_register_events_fn_t, **299**, 381
PMIx_server_register_fabric, **331**
PMIx_server_register_nspace, 8, 10, 18, 76, 97, 246, **247**, 253, 256, 410
PMIxServer.register_nspace (Python), 409
PMIx_server_setup_application, 9, 12, 103, 104, **266**, 271, 274, 413
PMIxServer.setup_application (Python), 412
PMIx_server_setup_fork, 8, **264**, 412
PMIxServer.setup_fork (Python), 411
PMIx_server_setup_local_support, 9, **270**, 414
PMIxServer.setup_local_support (Python), 413
pmix_server_spawn_fn_t, 96, **291**, 380
pmix_server_stdin_fn_t, **325**, 387
pmix_server_tool_connection_fn_t, 236, **307**, 383
pmix_server_unpublish_fn_t, **289**, 379
pmix_server_validate_cred_fn_t, **320**, 386
pmix_setup_application_cbfunc_t, **103**, 266
PMIx_Spawn, 8, 12, 53, 74, 83, 87–89, **155**, 155, 156, 160, 161, 164, 196, 252, 264, 291, 296, 312, 379, 394
PMIxClient.spawn (Python), 394
pmix_spawn_cbfunc_t, **96**, 96, 160, 292
PMIx_Spawn_nb, 8, 53, 96, 97, **160**
PMIx_Store_internal, 8, **132**, 390
PMIxClient.store_internal (Python), 390
PMIx_tool_connect_to_server, 11, **120**, 417
PMIxTool.connect_to_server (Python), 416
pmix_tool_connection_cbfunc_t, **105**, 307
PMIx_tool_finalize, 9, **120**, 416
PMIxTool.finalize (Python), 416
PMIx_tool_init, 9, 70, 113, **116**, 120, 416
PMIxTool.init (Python), 415
PMIx_Unpublish, 8, **150**, 152, 153, 394
PMIxClient.unpublish (Python), 393
PMIx_Unpublish_nb, 8, **152**
PMIx_Validate_credential, 11, 13, **240**, 400
PMIxClient.validate_credential (Python), 399
PMIx_Validate_credential_nb, **242**
pmix_validation_cbfunc_t, **107**, 243, 321
pmix_value_cbfunc_t, 18, **98**, 98

Index of Support Macros

PMIX_APP_CONSTRUCT, **54**
PMIX_APP_CREATE, **54**
PMIX_APP_DESTRUCT, **54**
PMIX_APP_FREE, **54**
PMIX_APP_INFO_CREATE, 10, 11, **55**
PMIX_ARGV_APPEND, **63**
PMIX_ARGV_APPEND_UNIQUE, **64**
PMIX_ARGV_COPY, **66**
PMIX_ARGV_COUNT, **66**
PMIX_ARGV_FREE, **65**
PMIX_ARGV_JOIN, **66**
PMIX_ARGV_SPLIT, **65**
PMIX_BYTE_OBJECT_CREATE, **61**
PMIX_BYTE_OBJECT_DESTRUCT, **60**
PMIX_BYTE_OBJECT_FREE, **61**
PMIX_BYTE_OBJECT_LOAD, **61**
PMIX_CHECK_KEY, **27**
PMIX_CHECK_NSPACE, **28**
PMIX_CHECK_PROCID, **31**
PMIX_COORD_CONSTRUCT, **364**
PMIX_COORD_CREATE, **365**
PMIX_COORD_DESTRUCT, **364**
PMIX_COORD_FREE, **365**
PMIX_DATA_ARRAY_CONSTRUCT, **36**, **62**
PMIX_DATA_ARRAY_CREATE, **37**, **63**
PMIX_DATA_ARRAY_DESTRUCT, **36**, **62**
PMIX_DATA_ARRAY_FREE, **37**
PMIX_DATA_ARRAY_RELEASE, **63**
PMIX_DATA_BUFFER_CONSTRUCT, **227**, 230, 232
PMIX_DATA_BUFFER_CREATE, **227**, 230, 232
PMIX_DATA_BUFFER_DESTRUCT, **228**
PMIX_DATA_BUFFER_LOAD, **228**
PMIX_DATA_BUFFER_RELEASE, **227**
PMIX_DATA_BUFFER_UNLOAD, **229**, 246, 247
PMIX_ENVAR_CONSTRUCT, **49**
PMIX_ENVAR_CREATE, **49**
PMIX_ENVAR_DESTRUCT, **49**
PMIX_ENVAR_FREE, **50**

PMIX_ENVAR_LOAD, **50**
PMIx_Heartbeat, 9, **209**
PMIX_INFO_CONSTRUCT, **42**
PMIX_INFO_CREATE, **43**, 46, 47
PMIX_INFO_DESTRUCT, **43**
PMIX_INFO_FREE, **43**
PMIX_INFO_IS_END, 10, 12, **47**
PMIX_INFO_IS_OPTIONAL, **47**
PMIX_INFO_IS_REQUIRED, 45, 46, **47**
PMIX_INFO_LOAD, **44**
PMIX_INFO_OPTIONAL, **46**
PMIX_INFO_REQUIRED, 45, **46**
PMIX_INFO_TRUE, **45**
PMIX_INFO_XFER, **44**, 253
PMIX_PDATA_CONSTRUCT, **51**
PMIX_PDATA_CREATE, **51**
PMIX_PDATA_DESTRUCT, **51**
PMIX_PDATA_FREE, **51**
PMIX_PDATA_LOAD, **52**
PMIX_PDATA_XFER, **53**
PMIX_PROC_CONSTRUCT, 30, **60**
PMIX_PROC_CREATE, **30**
PMIX_PROC_DESTRUCT, **30**
PMIX_PROC_FREE, **30**, 181
PMIX_PROC_INFO_CONSTRUCT, **33**
PMIX_PROC_INFO_CREATE, **34**
PMIX_PROC_INFO_DESTRUCT, **33**
PMIX_PROC_INFO_FREE, **34**
PMIX_PROC_LOAD, **31**
PMIX_QUERY_CONSTRUCT, **55**
PMIX_QUERY_CREATE, **56**
PMIX_QUERY_DESTRUCT, **56**
PMIX_QUERY_FREE, **56**
PMIX_QUERY_QUALIFIERS_CREATE, 10, 11, **56**
PMIX_REGATTR_CONSTRUCT, **58**
PMIX_REGATTR_CREATE, **58**
PMIX_REGATTR_DESTRUCT, **58**
PMIX_REGATTR_FREE, **59**
PMIX_REGATTR_LOAD, **59**
PMIX_REGATTR_XFER, **59**
PMIX_SETENV, **67**
PMIX_SYSTEM_EVENT, **26**
PMIX_VALUE_CONSTRUCT, **39**

PMIX_VALUE_CREATE, 39
PMIX_VALUE_DESTRUCT, 39
PMIX_VALUE_FREE, 39
PMIX_VALUE_GET_NUMBER, 42
PMIX_VALUE_LOAD, 40
PMIX_VALUE_UNLOAD, 40
PMIX_VALUE_XFER, 41

Index of Data Structures

pmix_alloc_directive_t, **48**, 48, 69, 111, 311, 372, 407
pmix_app_t, 10, 11, **53**, 53–55, 64, 67, 156, 160, 292, 373
pmix_byte_object_t, **60**, 60, 61, 68, 106, 178, 237, 241, 243, 272, 321, 325, 371
pmix_coord_t, 69, **363**, 363–365
pmix_coord_view_t, **365**, 366
pmix_data_array_t, 10, 11, **36**, 36, 37, **62**, 62, 63, 69, 85, 89, 185, 190, 192, 195, 198, 255–258, 267, 306, 312, 332, 364, 366, 367, 372
pmix_data_buffer_t, **226**, 226–231, 235
pmix_data_range_t, **35**, 35, 69, 110, 223, 303, 371, 406
pmix_data_type_t, 36, 37, 40, 42, 44, 52, 59, 62, 63, **68**, 68, 69, 111, 230, 232–234, 370, 406
pmix_envar_t, **48**, 49, 50, 69, 372
pmix_fabric_t, **330**, 330–334
pmix_group_operation_t, 327, **338**, 338, 356, 359
pmix_group_opt_t, **60**, 60, 401
pmix_info_directives_t, **45**, 45, 46, 69, 110, 372, 406
pmix_info_t, 3, 9, 10, 12, 14, 27, 35, **42**, 42–48, 55–57, 59, 69, 76, 77, 87, 89, 91, 99–102, 107, 108, 114–117, 122, 124, 144, 148, 174, 176, 178, 182, 186, 192–198, 200–202, 204–206, 209, 212, 215, 223, 237, 239, 241, 243, 253, 255–258, 267, 272–274, 303, 307, 308, 310, 312, 313, 316, 323, 339, 341, 343, 346, 348, 350, 353, 356, 359–361, 372, 374
pmix_iof_channel_t, **48**, 48, 69, 108, 111, 174, 272, 323, 372, 407
pmix_key_t, **27**, 27, 59, 125, 127, 370
pmix_nspace_t, **28**, 28, 31, 97, 370, 371
pmix_pdata_t, **50**, 50–53, 98, 148, 372
pmix_persistence_t, **35**, 35, 69, 110, 371, 405
pmix_proc_info_t, **32**, 32–34, 69, 85, 185, 190, 306, 371
pmix_proc_state_t, **31**, 31, 69, 110, 371, 404
pmix_proc_t, **29**, 29–31, 52, 59, 68, 75, 82, 102, 105, 108, 115, 117, 119, 121, 129, 139–141, 154, 220, 223, 224, 230, 231, 251, 262–265, 272, 277, 278, 280, 281, 284, 285, 288, 290, 292, 296, 298, 303, 305, 309, 311, 314, 316, 319, 321, 323, 325, 327, 339, 343, 350, 353, 357, 371
pmix_query_t, 10, 11, **55**, 55, 56, 69, 184, 189, 191, 305, 307, 373
pmix_rank_t, **29**, 29, 31, 69, 371
pmix_regattr_t, 13, **57**, 57–59, 69, 94, 192, 269, 373
pmix_scope_t, **34**, 34, 69, 110, 126, 371, 405
pmix_status_t, **22**, 22, 26, 40–42, 63, 64, 67, 68, 99, 100, 102, 104–107, 109, 218, 223, 300, 302, 303, 370, 382, 404
pmix_value_t, **37**, 37–42, 68, 98, 99, 125, 126, 333, 334, 372

Index of Constants

PMIX_ALLOC_DIRECTIVE, 69
PMIX_ALLOC_EXTEND, 48
PMIX_ALLOC_EXTERNAL, 48
PMIX_ALLOC_NEW, 48
PMIX_ALLOC_REAQUIRE, 48
PMIX_ALLOC_RELEASE, 48
PMIX_APP, 68
PMIX_APP_WILDCARD, 21
PMIX_BOOL, 68
PMIX_BUFFER, 68
PMIX_BYTE, 68
PMIX_BYTE_OBJECT, 68
PMIX_COMMAND, 69
PMIX_COMPRESSED_STRING, 69
PMIX_CONNECT_REQUESTED, 24
PMIX_COORD, 69
PMIX_COORD_LOGICAL_VIEW, 365
PMIX_COORD_PHYSICAL_VIEW, 365
PMIX_COORD_VIEW_UNDEF, 365
PMIX_DATA_ARRAY, 69
PMIX_DATA_RANGE, 69
PMIX_DATA_TYPE, 69
PMIX_DOUBLE, 68
PMIX_ENVAR, 69
PMIX_ERR_BAD_PARAM, 23
PMIX_ERR_COMM_FAILURE, 23
PMIX_ERR_DATA_VALUE_NOT_FOUND, 23
PMIX_ERR_DEBUGGER_RELEASE, 22
PMIX_ERR_DUPLICATE_KEY, 24
PMIX_ERR_EVENT_REGISTRATION, 24
PMIX_ERR_GET_MALLOC_REQD, 25
PMIX_ERR_HANDSHAKE_FAILED, 22
PMIX_ERR_IN_ERRNO, 23
PMIX_ERR_INIT, 23
PMIX_ERR_INVALID_ARG, 23
PMIX_ERR_INVALID_ARGS, 23
PMIX_ERR_INVALID_CRED, 22
PMIX_ERR_INVALID_KEY, 23

PMIX_ERR_INVALID_KEY_LENGTH, 23
PMIX_ERR_INVALID_KEYVALP, 23
PMIX_ERR_INVALID_LENGTH, 23
PMIX_ERR_INVALID_NAMESPACE, 23
PMIX_ERR_INVALID_NUM_ARGS, 23
PMIX_ERR_INVALID_NUM_PARSED, 23
PMIX_ERR_INVALID_OPERATION, 24
PMIX_ERR_INVALID_SIZE, 23
PMIX_ERR_INVALID_TERMINATION, 24
PMIX_ERR_INVALID_VAL, 23
PMIX_ERR_INVALID_VAL_LENGTH, 23
PMIX_ERR_IOF_COMPLETE, 25
PMIX_ERR_IOF_FAILURE, 25
PMIX_ERR_JOB_TERMINATED, 24
PMIX_ERR_LOST_CONNECTION_TO_CLIENT, 23
PMIX_ERR_LOST_CONNECTION_TO_SERVER, 23
PMIX_ERR_LOST_PEER_CONNECTION, 23
PMIX_ERR_NO_PERMISSIONS, 23
PMIX_ERR_NODE_DOWN, 26
PMIX_ERR_NODE_OFFLINE, 26
PMIX_ERR_NOMEM, 23
PMIX_ERR_NOT_FOUND, 23
PMIX_ERR_NOT_IMPLEMENTED, 23
PMIX_ERR_NOT_SUPPORTED, 23
PMIX_ERR_OUT_OF_RESOURCE, 23
PMIX_ERR_PACK_FAILURE, 23
PMIX_ERR_PACK_MISMATCH, 23
PMIX_ERR_PARTIAL_SUCCESS, 24
PMIX_ERR_PROC_ABORTED, 22
PMIX_ERR_PROC_ABORTING, 22
PMIX_ERR_PROC_CHECKPOINT, 22
PMIX_ERR_PROC_ENTRY_NOT_FOUND, 22
PMIX_ERR_PROC_MIGRATE, 22
PMIX_ERR_PROC_REQUESTED_ABORT, 22
PMIX_ERR_PROC_RESTART, 22
PMIX_ERR_READY_FOR_HANDSHAKE, 22
PMIX_ERR_REPEAT_ATTR_REGISTRATION, 25
PMIX_ERR_RESOURCE_BUSY, 23
PMIX_ERR_SERVER_FAILED_REQUEST, 22
PMIX_ERR_SERVER_NOT_AVAIL, 23
PMIX_ERR_SILENT, 22
PMIX_ERR_SYS_BASE, 25
PMIX_ERR_SYS_OTHER, 26

PMIX_ERR_TIMEOUT, 23
PMIX_ERR_TYPE_MISMATCH, 22
PMIX_ERR_UNKNOWN_DATA_TYPE, 22
PMIX_ERR_UNPACK_FAILURE, 23
PMIX_ERR_UNPACK_INADEQUATE_SPACE, 22
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, 23
PMIX_ERR_UNREACH, 23
PMIX_ERR_UPDATE_ENDPOINTS, 24
PMIX_ERR_WOULD_BLOCK, 22
PMIX_ERROR, 22
PMIX_EVENT_ACTION_COMPLETE, 26
PMIX_EVENT_ACTION_DEFERRED, 26
PMIX_EVENT_NO_ACTION_TAKEN, 26
PMIX_EVENT_PARTIAL_ACTION_TAKEN, 26
PMIX_EXISTS, 22
PMIX_EXTERNAL_ERR_BASE, 26
PMIX_FABRIC_UPDATE_PENDING, 331
PMIX_FABRIC_UPDATED, 331
PMIX_FLOAT, 68
PMIX_FWD_ALL_CHANNELS, 48
PMIX_FWD_NO_CHANNELS, 48
PMIX_FWD_STDDIAG_CHANNEL, 48
PMIX_FWD_STDERR_CHANNEL, 48
PMIX_FWD_STDIN_CHANNEL, 48
PMIX_FWD_STDOUT_CHANNEL, 48
PMIX_GDS_ACTION_COMPLETE, 24
PMIX_GLOBAL, 34
PMIX_GROUP_ACCEPT, 60, 338
PMIX_GROUP_CONSTRUCT, 338
PMIX_GROUP_CONSTRUCT_ABORT, 25
PMIX_GROUP_CONSTRUCT_COMPLETE, 25
PMIX_GROUP_CONTEXT_ID_ASSIGNED, 25
PMIX_GROUP_DECLINE, 60, 338
PMIX_GROUP_DESTRUCT, 338
PMIX_GROUP_INVITE_ACCEPTED, 25
PMIX_GROUP_INVITE_DECLINED, 25
PMIX_GROUP_INVITE_FAILED, 25
PMIX_GROUP_INVITED, 24
PMIX_GROUP_LEADER_FAILED, 25
PMIX_GROUP_LEADER_SELECTED, 25
PMIX_GROUP_LEFT, 25
PMIX_GROUP_MEMBER_FAILED, 25
PMIX_GROUP_MEMBERSHIP_UPDATE, 25

PMIX_INFO, 68
PMIX_INFO_ARRAY_END, 46
PMIX_INFO_DIRECTIVES, 69
PMIX_INFO_REQD, 46
PMIX_INT, 68
PMIX_INT16, 68
PMIX_INT32, 68
PMIX_INT64, 68
PMIX_INT8, 68
PMIX_INTERNAL, 34
PMIX_IOF_CHANNEL, 69
PMIX_JCTRL_CHECKPOINT, 23
PMIX_JCTRL_CHECKPOINT_COMPLETE, 23
PMIX_JCTRL_PREEMPT_ALERT, 23
PMIX_KVAL, 68
PMIX_LAUNCH_COMPLETE, 24
PMIX_LAUNCH_DIRECTIVE, 24
PMIX_LAUNCHER_READY, 24
PMIX_LOCAL, 34
PMIX_MAX_KEYLEN, 21
PMIX_MAX_NSLEN, 21
PMIX_MODEL_DECLARED, 24
PMIX_MODEL_RESOURCES, 24
PMIX_MONITOR_FILE_ALERT, 24
PMIX_MONITOR_HEARTBEAT_ALERT, 24
PMIX_NETWORK_COORDS_UPDATED, 366
PMIX_NOTIFY_ALLOC_COMPLETE, 23
PMIX_OPENMP_PARALLEL_ENTERED, 24
PMIX_OPENMP_PARALLEL_EXITED, 24
PMIX_OPERATION_IN_PROGRESS, 24
PMIX_OPERATION_SUCCEEDED, 24
PMIX_PDATA, 68
PMIX_PERSIST, 69
PMIX_PERSIST_APP, 35
PMIX_PERSIST_FIRST_READ, 35
PMIX_PERSIST_INDEF, 35
PMIX_PERSIST_INVALID, 35
PMIX_PERSIST_PROC, 35
PMIX_PERSIST_SESSION, 35
PMIX_PID, 68
PMIX_POINTER, 69
PMIX_PROC, 68
PMIX_PROC_HAS_CONNECTED, 24

PMIX_PROC_INFO, 69
PMIX_PROC_RANK, 69
PMIX_PROC_STATE, 69
PMIX_PROC_STATE_ABORTED, 32
PMIX_PROC_STATE_ABORTED_BY_SIG, 32
PMIX_PROC_STATE_CALLED_ABORT, 32
PMIX_PROC_STATE_CANNOT_RESTART, 32
PMIX_PROC_STATE_COMM_FAILED, 32
PMIX_PROC_STATE_CONNECTED, 32
PMIX_PROC_STATE_ERROR, 32
PMIX_PROC_STATE_FAILED_TO_LAUNCH, 32
PMIX_PROC_STATE_FAILED_TO_START, 32
PMIX_PROC_STATE_KILLED_BY_CMD, 32
PMIX_PROC_STATE_LAUNCH_UNDERWAY, 32
PMIX_PROC_STATE_MIGRATING, 32
PMIX_PROC_STATE_PREPPED, 32
PMIX_PROC_STATE_RESTART, 32
PMIX_PROC_STATE_RUNNING, 32
PMIX_PROC_STATE_TERM_NON_ZERO, 32
PMIX_PROC_STATE_TERM_WO_SYNC, 32
PMIX_PROC_STATE_TERMINATE, 32
PMIX_PROC_STATE_TERMINATED, 32
PMIX_PROC_STATE_UNDEF, 32
PMIX_PROC_STATE_UNTERMINATED, 32
PMIX_PROC_TERMINATED, 24
PMIX_QUERY, 69
PMIX_QUERY_PARTIAL_SUCCESS, 23
PMIX_RANGE_CUSTOM, 35
PMIX_RANGE_GLOBAL, 35
PMIX_RANGE_INVALID, 35
PMIX_RANGE_LOCAL, 35
PMIX_RANGE_NAMESPACE, 35
PMIX_RANGE_PROC_LOCAL, 35
PMIX_RANGE_RM, 35
PMIX_RANGE_SESSION, 35
PMIX_RANGE_UNDEF, 35
PMIX_RANK_INVALID, 29
PMIX_RANK_LOCAL_NODE, 29
PMIX_RANK_LOCAL_PEERS, 29
PMIX_RANK_UNDEF, 29
PMIX_RANK_VALID, 29
PMIX_RANK_WILDCARD, 29
PMIX_REGATTR, 69

PMIX_REGEX, 69
PMIX_REMOTE, 34
PMIX_SCOPE, 69
PMIX_SCOPE_UNDEF, 34
PMIX_SIZE, 68
PMIX_STATUS, 68
PMIX_STRING, 68
PMIX_SUCCESS, 22
PMIX_TIME, 68
PMIX_TIMEVAL, 68
PMIX_UINT, 68
PMIX_UINT16, 68
PMIX_UINT32, 68
PMIX_UINT64, 68
PMIX_UINT8, 68
PMIX_UNDEF, 68
PMIX_VALUE, 68

Index of Attributes

PMIX_ADD_ENVAR, **88**
PMIX_ADD_HOST, **83**, 157, 162, 293
PMIX_ADD_HOSTFILE, **83**, 157, 162, 293
PMIX_ALLOC_BANDWIDTH, **89**, 89, 195, 198, 267, 313
PMIX_ALLOC_CPU_LIST, **89**, 194, 197, 312
PMIX_ALLOC_ID, 12, **89**, 196, 312
PMIX_ALLOC_MEM_SIZE, **89**, 194, 197, 312
PMIX_ALLOC_NETWORK, **89**, 194, 197, 267, 312
PMIX_ALLOC_NETWORK_ENDPTS, 89, **90**, 194, 195, 197, 198, 267, 312
PMIX_ALLOC_NETWORK_ENDPTS_NODE, **90**, 195, 198, 267
PMIX_ALLOC_NETWORK_ID, **89**, 89, 194, 197, 267, 312
PMIX_ALLOC_NETWORK_PLANE, 89, **90**, 195, 198, 267, 312
PMIX_ALLOC_NETWORK_QOS, 89, **90**, 195, 198, 267, 313
PMIX_ALLOC_NETWORK_SEC_KEY, 89, **90**, 195, 198, 267, 313
PMIX_ALLOC_NETWORK_TYPE, 89, **90**, 194, 195, 197, 198, 267, 312
PMIX_ALLOC_NODE_LIST, **89**, 194, 197, 312
PMIX_ALLOC_NUM_CPU_LIST, **89**, 194, 197, 312
PMIX_ALLOC_NUM_CPUS, **89**, 194, 197, 312
PMIX_ALLOC_NUM_NODES, **89**, 194, 197, 312
PMIX_ALLOC_REQ_ID, 12, **89**, 194, 197
PMIX_ALLOC_TIME, **90**, 194, 197, 267, 312
PMIX_ALLOCATED_NODELIST, **75**, 250
PMIX_ANL_MAP, **81**, 251
PMIX_APP_INFO, **76**, 128, 131, 135, 183, 188
PMIX_APP_INFO_ARRAY, **76**, 77, 257
PMIX_APP_MAP_REGEX, **81**
PMIX_APP_MAP_TYPE, **81**
PMIX_APP_RANK, **74**, 250
PMIX_APP_SIZE, **77**, 135, 250, 256
PMIX_APPEND_ENVAR, **88**
PMIX_APPLDR, **75**, 250, 256
PMIX_APPNUM, **74**, 76, 128, 131, 135, 183, 188, 249, 250, 257
PMIX_ARCH, **74**
PMIX_ATTR_UNDEF, **69**
PMIX_AVAIL_PHYS_MEMORY, **78**, 251
PMIX_BINDTO, **83**, 158, 162, 251, 294
PMIX_CLEANUP_EMPTY, **91**, 201, 203
PMIX_CLEANUP_IGNORE, **91**, 201, 203

PMIX_CLEANUP_LEAVE_TOPDIR, **91**, 201, 204
PMIX_CLEANUP_RECURSIVE, **91**, 201, 203
PMIX_CLIENT_ATTRIBUTES, 13, **93**, 183, 188
PMIX_CLIENT_AVG_MEMORY, **78**
PMIX_CLIENT_FUNCTIONS, **93**
PMIX_CLUSTER_ID, **74**
PMIX_COLLECT_DATA, **79**, 139, 141, 282
PMIX_COLLECTIVE_ALGO, 9, **79**, 139, 142, 166, 169, 282, 297
PMIX_COLLECTIVE_ALGO_REQD, **80**, 139, 142, 166, 169, 282, 297
PMIX_CONNECT_MAX_RETRIES, **71**, 118
PMIX_CONNECT_RETRY_DELAY, **71**, 117
PMIX_CONNECT_SYSTEM_FIRST, **71**, 117, 119, 121
PMIX_CONNECT_TO_SYSTEM, **70**, 117, 119, 121
PMIX_COSPAWN_APP, **84**
PMIX_CPU_LIST, **85**, 159, 163, 295
PMIX_CPUS_PER_PROC, **84**, 158, 163, 295
PMIX_CPUSET, **73**
PMIX_CRED_TYPE, **92**, 320
PMIX_CREDENTIAL, **73**
PMIX_CRYPTO_KEY, **92**
PMIX_DAEMON_MEMORY, **78**
PMIX_DATA_SCOPE, **80**, 127, 131
PMIX_DEBUG_APP_DIRECTIVES, **88**
PMIX_DEBUG_JOB, **88**
PMIX_DEBUG_JOB_DIRECTIVES, **88**
PMIX_DEBUG_STOP_IN_INIT, **87**
PMIX_DEBUG_STOP_ON_EXEC, **87**
PMIX_DEBUG_WAIT_FOR_NOTIFY, **88**
PMIX_DEBUG_WAITING_FOR_NOTIFY, **88**
PMIX_DEBUGGER_DAEMONS, **84**, 158, 163, 294
PMIX_DISPLAY_MAP, **83**, 157, 162, 294
PMIX_DSTPATH, **71**
PMIX_EMBED_BARRIER, **80**, 116
PMIX_ENUM_VALUE, 13, 57, **94**
PMIX_EVENT_ACTION_TIMEOUT, **82**, 220
PMIX_EVENT_AFFECTED_PROC, **82**, 220, 224
PMIX_EVENT_AFFECTED_PROCS, **82**, 220, 224
PMIX_EVENT_BASE, **70**, 115, 118, 124
PMIX_EVENT_CUSTOM_RANGE, **82**, 220, 224
PMIX_EVENT_DO_NOT_CACHE, **82**
PMIX_EVENT_HDLR_AFTER, **81**, 219
PMIX_EVENT_HDLR_APPEND, **82**, 220
PMIX_EVENT_HDLR_BEFORE, **81**, 219

PMIX_EVENT_HDLR_FIRST, **81**, 219
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, **81**, 219
PMIX_EVENT_HDLR_LAST, **81**, 219
PMIX_EVENT_HDLR_LAST_IN_CATEGORY, **81**, 219
PMIX_EVENT_HDLR_NAME, **81**, 219
PMIX_EVENT_HDLR_PREPEND, **82**, 220
PMIX_EVENT_NO_TERMINATION, **82**
PMIX_EVENT_NON_DEFAULT, **82**, 224
PMIX_EVENT_PROXY, **82**
PMIX_EVENT_RETURN_OBJECT, **82**, 220
PMIX_EVENT_SILENT_TERMINATION, **82**, 220
PMIX_EVENT_TERMINATE_JOB, **82**, 220
PMIX_EVENT_TERMINATE_NODE, **82**, 220
PMIX_EVENT_TERMINATE_PROC, **82**, 220
PMIX_EVENT_TERMINATE_SESSION, **82**, 220
PMIX_EVENT_TEXT_MESSAGE, **82**
PMIX_EVENT_WANT_TERMINATION, **82**
PMIX_EXIT_CODE, **75**
PMIX_FWD_STDDIAG, 11, **84**
PMIX_FWD_STDERR, **84**, 158, 163, 294, 308
PMIX_FWD_STDIN, **84**, 158, 163, 294, 308
PMIX_FWD_STDOUT, **84**, 158, 163, 294, 308
PMIX_GDS_MODULE, **73**, 115, 118, 124
PMIX_GET_STATIC_VALUES, **80**, 128, 131
PMIX_GLOBAL_RANK, **74**, 251
PMIX_GROUP_ASSIGN_CONTEXT_ID, **94**, 327, 329, 340, 344, 351, 354
PMIX_GROUP_CONTEXT_ID, **94**, 328
PMIX_GROUP_ENDPT_DATA, **95**, 328
PMIX_GROUP_ID, **94**, 328
PMIX_GROUP_INVITE_DECLINE, **94**
PMIX_GROUP_LEADER, **94**, 340, 341, 344, 353, 358
PMIX_GROUP_LOCAL_ONLY, **94**, 328, 340, 344
PMIX_GROUP_MEMBERSHIP, **94**, 328, 341
PMIX_GROUP_NOTIFY_TERMINATION, **94**, 340–342, 344, 347, 351, 354
PMIX_GROUP_OPTIONAL, **94**, 328, 340, 342, 344, 350, 354
PMIX_GRPID, **71**, 108, 143, 145, 147, 149, 151, 153, 184, 189, 194, 197, 200, 203, 206, 208, 210, 213, 238, 239, 241, 243, 286–291, 293, 300, 306, 308, 309, 311, 314, 317, 319, 321, 323, 326
PMIX_HOST, **83**, 157, 161, 293
PMIX_HOST_ATTRIBUTES, 13, **93**, 184, 188, 192
PMIX_HOST_FUNCTIONS, **93**
PMIX_HOSTFILE, **83**, 157, 161, 293
PMIX_HOSTNAME, **75**, 76, 77, 128, 131, 137, 183, 188, 251

PMIX_HWLOC_HOLE_KIND, **79**
PMIX_HWLOC_SHARE_TOPO, **79**
PMIX_HWLOC_SHMEM_ADDR, **79**
PMIX_HWLOC_SHMEM_FILE, **79**
PMIX_HWLOC_SHMEM_SIZE, **79**
PMIX_HWLOC_XML_V1, **79**, 251
PMIX_HWLOC_XML_V2, **79**, 251
PMIX_IMMEDIATE, **79**, 127, 131
PMIX_INDEX_ARGV, **84**, 158, 163, 295
PMIX_IOF_BUFFERING_SIZE, **92**, 175, 179, 324
PMIX_IOF_BUFFERING_TIME, **92**, 175, 179, 324
PMIX_IOF_CACHE_SIZE, **92**, 175, 179, 324
PMIX_IOF_COMPLETE, **93**, 108, 375
PMIX_IOF_DROP_NEWEST, **92**, 175, 179, 324
PMIX_IOF_DROP_OLDEST, **92**, 175, 179, 324
PMIX_IOF_TAG_OUTPUT, **93**, 175
PMIX_IOF_TIMESTAMP_OUTPUT, **93**, 175
PMIX_IOF_XML_OUTPUT, **93**, 176
PMIX_JOB_CONTINUOUS, **85**, 159, 163, 295
PMIX_JOB_CTRL_CANCEL, **90**, 201, 204, 315
PMIX_JOB_CTRL_CHECKPOINT, **90**, 201, 204, 315
PMIX_JOB_CTRL_CHECKPOINT_EVENT, **90**, 201, 204, 315
PMIX_JOB_CTRL_CHECKPOINT_METHOD, **91**, 201, 204, 316
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL, **90**, 201, 204, 315
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT, **90**, 201, 204, 315
PMIX_JOB_CTRL_ID, **90**, 90, 200, 201, 203, 204, 315
PMIX_JOB_CTRL_KILL, **90**, 200, 203, 315
PMIX_JOB_CTRL_PAUSE, **90**, 200, 203, 315
PMIX_JOB_CTRL_PREEMPTIBLE, **91**, 201, 204, 316
PMIX_JOB_CTRL_PROVISION, **91**, 201, 204, 316
PMIX_JOB_CTRL_PROVISION_IMAGE, **91**, 201, 204, 316
PMIX_JOB_CTRL_RESTART, **90**, 201, 204, 315
PMIX_JOB_CTRL_RESUME, **90**, 200, 203, 315
PMIX_JOB_CTRL_SIGNAL, **91**, 200, 203, 315
PMIX_JOB_CTRL_TERMINATE, **91**, 200, 203, 315
PMIX_JOB_INFO, **76**, 127, 131, 135, 183, 188
PMIX_JOB_INFO_ARRAY, 10, **76**, 77, 256
PMIX_JOB_NUM_APPS, **77**, 135, 250, 256
PMIX_JOB_RECOVERABLE, **85**, 159, 163, 295
PMIX_JOB_SIZE, 10, 12, **77**, 129, 132, 135, 248, 256
PMIX_JOB_TERM_STATUS, **80**
PMIX_JOBID, **74**, 76, 128, 131, 135, 183, 188, 248, 256
PMIX_LAUNCHER, **71**

PMIX_LOCAL_CPUSETS, **75**, 249, 259
PMIX_LOCAL_PEERS, **75**, 249, 259
PMIX_LOCAL_PROCS, **75**, 251
PMIX_LOCAL_RANK, **74**, 183, 184, 188, 189, 249
PMIX_LOCAL_SIZE, **77**, 249
PMIX_LOCAL_TOPO, **78**
PMIX_LOCALITY, **75**
PMIX_LOCALITY_STRING, **78**
PMIX_LOCALLDR, **75**, 251
PMIX_LOG_EMAIL, **87**, 212, 215, 310
PMIX_LOG_EMAIL_ADDR, **87**, 212, 215, 310
PMIX_LOG_EMAIL_MSG, **87**, 212, 215, 310
PMIX_LOG_EMAIL_SENDER_ADDR, **87**
PMIX_LOG_EMAIL_SERVER, **87**
PMIX_LOG_EMAIL_SRVR_PORT, **87**
PMIX_LOG_EMAIL_SUBJECT, **87**, 212, 215, 310
PMIX_LOG_GENERATE_TIMESTAMP, **87**, 211, 214
PMIX_LOG_GLOBAL_DATASTORE, **87**, 212, 215
PMIX_LOG_GLOBAL_SYSLOG, **86**, 211, 214
PMIX_LOG_JOB_RECORD, **87**, 212, 215
PMIX_LOG_LOCAL_SYSLOG, **86**, 211, 214
PMIX_LOG_MSG, **87**, 310
PMIX_LOG_ONCE, **87**, 211, 214
PMIX_LOG_SOURCE, **86**, 211, 214
PMIX_LOG_STDERR, **86**, 211, 214, 309
PMIX_LOG_STDOUT, **86**, 211, 214, 309
PMIX_LOG_SYSLOG, **86**, 211, 214, 309
PMIX_LOG_SYSLOG_PRI, **86**, 211, 214
PMIX_LOG_TAG_OUTPUT, **87**, 211, 214
PMIX_LOG_TIMESTAMP, **86**, 211, 214
PMIX_LOG_TIMESTAMP_OUTPUT, **87**, 211, 214
PMIX_LOG_XML_OUTPUT, **87**, 211, 214
PMIX_MAP_BLOB, **81**
PMIX_MAPBY, **83**, 157, 162, 250, 294
PMIX_MAPPER, **83**, 83, 157, 162, 293
PMIX_MAX_PROCS, 12, 57, 77, **78**, 78, 137, 248, 249
PMIX_MAX_RESTARTS, **85**, 159, 164, 295
PMIX_MAX_VALUE, 13, 57, **94**
PMIX_MERGE_STDERR_STDOUT, **84**, 158, 163, 294
PMIX_MIN_VALUE, 13, 57, **94**
PMIX_MODEL_AFFINITY_POLICY, **72**
PMIX_MODEL_CPU_TYPE, **72**
PMIX_MODEL_LIBRARY_NAME, **72**, 252, 268

PMIX_MODEL_LIBRARY_VERSION, **72**, 252, 268
PMIX_MODEL_NUM_CPUS, **72**
PMIX_MODEL_NUM_THREADS, **72**
PMIX_MODEL_PHASE_NAME, **72**
PMIX_MODEL_PHASE_TYPE, **72**
PMIX_MONITOR_APP_CONTROL, **91**, 206, 208, 318
PMIX_MONITOR_CANCEL, **91**, 206, 208, 318
PMIX_MONITOR_FILE, **92**, 206, 207, 209, 318
PMIX_MONITOR_FILE_ACCESS, **92**, 206, 209, 318
PMIX_MONITOR_FILE_CHECK_TIME, **92**, 207, 209, 318
PMIX_MONITOR_FILE_DROPS, **92**, 207, 209, 318
PMIX_MONITOR_FILE_MODIFY, **92**, 207, 209, 318
PMIX_MONITOR_FILE_SIZE, **92**, 206, 209, 318
PMIX_MONITOR_HEARTBEAT, **91**, 206, 208, 318
PMIX_MONITOR_HEARTBEAT_DROPS, **92**, 206, 209, 318
PMIX_MONITOR_HEARTBEAT_TIME, **91**, 206, 208, 318
PMIX_MONITOR_ID, **91**, 206, 208, 318
PMIX_NETWORK_COORDINATE, **366**
PMIX_NETWORK_DIMS, **366**
PMIX_NETWORK_ENDPT, **367**
PMIX_NETWORK_NIC, **366**
PMIX_NETWORK_PLANE, 332, 364, **366**
PMIX_NETWORK_SHAPE, **367**
PMIX_NETWORK_VIEW, **366**
PMIX_NO_OVERSUBSCRIBE, **84**, 159, 163, 295
PMIX_NO_PROCS_ON_HEAD, **84**, 159, 163, 295
PMIX_NODE_INFO, **76**, 128, 131, 137, 183, 188
PMIX_NODE_INFO_ARRAY, **77**, 77, 256, 258
PMIX_NODE_LIST, **75**
PMIX_NODE_MAP, 12, 75, **81**, 249, 255–257, 268
PMIX_NODE_RANK, **74**, 249
PMIX_NODE_SIZE, **78**, 137, 251
PMIX_NODEID, **75**, 76, 77, 128, 131, 137, 183, 188, 249
PMIX_NON_PMI, **84**, 158, 162, 294
PMIX_NOTIFY_COMPLETION, **80**, 159
PMIX_NPROC_OFFSET, **74**, 250
PMIX_NSDIR, **74**, 74
PMIX_NSPACE, **74**, 76, 128, 131, 135, 183, 184, 188, 189, 256
PMIX_NUM_NODES, **78**, 129, 132, 133, 135, 255, 256
PMIX_NUM_SLOTS, **78**
PMIX_OPTIONAL, **80**, 127, 130
PMIX_OUTPUT_TO_FILE, **84**, 158, 163, 295
PMIX_PARENT_ID, **75**, 156, 161, 293

PMIX_PERSISTENCE, **80**, 144, 146, 286
PMIX_PERSONALITY, **83**, 157, 162, 293
PMIX_PPR, **83**, 157, 162, 294
PMIX_PREFIX, **83**, 157, 161, 293
PMIX_PRELOAD_BIN, **84**, 157, 162, 293
PMIX_PRELOAD_FILES, **84**, 157, 162, 293
PMIX_PREPEND_ENVAR, **88**
PMIX_PROC_BLOB, **81**
PMIX_PROC_DATA, **81**, 257
PMIX_PROC_MAP, 12, **81**, 249, 255, 256, 268
PMIX_PROC_PID, **75**
PMIX_PROC_STATE_STATUS, **80**
PMIX_PROC_TERM_STATUS, **80**
PMIX_PROC_URI, **75**, 186, 190
PMIX_PROCDIR, **74**
PMIX_PROCID, **74**, 183, 184, 188, 189, 251
PMIX_PROGRAMMING_MODEL, **72**, 252, 268
PMIX_PSET_NAME, **71**, 335
PMIX_QUERY_ALLOC_STATUS, **86**, 185, 190, 307
PMIX_QUERY_ATTRIBUTE_SUPPORT, **86**, 183, 188, 191
PMIX_QUERY_AUTHORIZATIONS, **85**
PMIX_QUERY_DEBUG_SUPPORT, **86**, 185, 190, 306
PMIX_QUERY_JOB_STATUS, **85**, 185, 190, 306
PMIX_QUERY_LOCAL_ONLY, **86**, 306
PMIX_QUERY_LOCAL_PROC_TABLE, **85**, 185, 190, 306
PMIX_QUERY_MEMORY_USAGE, **86**, 185, 190, 306
PMIX_QUERY_NAMESPACES, **85**, 185, 189, 306
PMIX_QUERY_NUM_PSETS, **86**
PMIX_QUERY_PROC_TABLE, **85**, 185, 190, 306
PMIX_QUERY_PSET_NAMES, **86**
PMIX_QUERY_QUEUE_LIST, **85**, 185, 190, 306
PMIX_QUERY_QUEUE_STATUS, **85**, 185, 190, 306
PMIX_QUERY_REFRESH_CACHE, **85**, 182, 186, 187, 191
PMIX_QUERY_REPORT_AVG, **86**, 185, 190, 306
PMIX_QUERY_REPORT_MINMAX, **86**, 185, 190, 307
PMIX_QUERY_SPAWN_SUPPORT, **85**, 185, 190, 306
PMIX_RANGE, **80**, 144, 146, 147, 150, 151, 153, 207, 220, 286, 288, 291, 304, 328, 329
PMIX_RANK, **74**, 183, 184, 188, 189, 249
PMIX_RANKBY, **83**, 158, 162, 250, 294
PMIX_RECONNECT_SERVER, **71**
PMIX_REGISTER_CLEANUP, **91**, 200, 203
PMIX_REGISTER_CLEANUP_DIR, **91**, 200, 203
PMIX_REGISTER_NODATA, **80**, 248

PMIX_REPORT_BINDINGS, **85**, 159, 163, 295
PMIX_REQUESTOR_IS_CLIENT, **71**, 156, 161
PMIX_REQUESTOR_IS_TOOL, **71**, 156, 161
PMIX_RM_NAME, **88**
PMIX_RM_VERSION, **88**
PMIX_SEND_HEARTBEAT, **91**
PMIX_SERVER_ATTRIBUTES, 13, **93**, 183, 188
PMIX_SERVER_ENABLE_MONITORING, **70**
PMIX_SERVER_FUNCTIONS, **93**
PMIX_SERVER_GATEWAY, **70**
PMIX_SERVER_HOSTNAME, **71**
PMIX_SERVER_NSPACE, **70**, 121, 122, 250
PMIX_SERVER_PIDINFO, **70**, 117, 119, 121
PMIX_SERVER_RANK, **70**, 122, 250
PMIX_SERVER_REMOTE_CONNECTIONS, **70**, 123
PMIX_SERVER_SCHEDULER, 330, **331**
PMIX_SERVER_SYSTEM_SUPPORT, **70**, 122
PMIX_SERVER_TMPDIR, **70**, 122
PMIX_SERVER_TOOL_SUPPORT, **70**, 122, 124, 236
PMIX_SERVER_URI, **71**, 117, 119, 121, 185, 190
PMIX_SESSION_ID, **75**, 76, 127, 131, 134, 183, 187, 250, 255, 256
PMIX_SESSION_INFO, **75**, 127, 131, 133, 182, 187
PMIX_SESSION_INFO_ARRAY, 10, **76**, 77, 248, 255
PMIX_SET_ENVAR, **88**
PMIX_SET_SESSION_CWD, **84**, 157, 161, 293
PMIX_SETUP_APP_ALL, **93**, 266
PMIX_SETUP_APP_ENVARS, **93**, 266
PMIX_SETUP_APP_NONENVARS, **93**, 266
PMIX_SINGLE_LISTENER, **72**, 114
PMIX_SOCKET_MODE, **72**, 114, 118, 123
PMIX_SPAWN_TOOL, **85**
PMIX_SPAWNED, **73**, 156, 161, 293
PMIX_STDIN_TGT, **84**, 158, 162, 294
PMIX_SYSTEM_TMPDIR, **70**, 122
PMIX_TAG_OUTPUT, **84**, 158, 163, 294
PMIX_TCP_DISABLE_IPV4, **73**, 115, 118, 123
PMIX_TCP_DISABLE_IPV6, **73**, 115, 118, 123
PMIX_TCP_IF_EXCLUDE, **73**, 114, 118, 123
PMIX_TCP_IF_INCLUDE, **73**, 114, 118, 123
PMIX_TCP_IPV4_PORT, **73**, 115, 118, 123
PMIX_TCP_IPV6_PORT, **73**, 115, 118, 123
PMIX_TCP_REPORT_URI, **73**, 114, 118, 123
PMIX_TCP_URI, **73**, 117, 119

PMIX_TDIR_RMCLEAN, **74**
PMIX_THREADING_MODEL, **72**
PMIX_TIME_REMAINING, **86**, 180, 185, 190, 307
PMIX_TIMEOUT, 3, 14, **79**, 128, 129, 131, 132, 139, 140, 142, 144, 146–148, 150, 151, 153, 166, 169, 170, 172, 173, 238, 240, 242, 244, 282, 285, 286, 289, 291, 295, 297, 299, 320, 322, 338, 340–342, 345–347, 349, 351, 355, 357, 359, 360
PMIX_TIMESTAMP_OUTPUT, **84**, 158, 163, 294
PMIX_TMPDIR, **74**, 74
PMIX_TOOL_ATTRIBUTES, 13, **93**, 184, 189
PMIX_TOOL_DO_NOT_CONNECT, **71**, 117, 119
PMIX_TOOL_FUNCTIONS, **93**
PMIX_TOOL_NSPACE, **70**, 117
PMIX_TOOL_RANK, **70**, 117
PMIX_TOPOLOGY, **78**
PMIX_TOPOLOGY_FILE, **78**
PMIX_TOPOLOGY_SIGNATURE, **78**
PMIX_TOPOLOGY_XML, **78**
PMIX_UNIV_SIZE, 10, 12, **77**, 129, 132, 133, 248, 255
PMIX_UNSET_ENVAR, **88**
PMIX_USERID, **71**, 108, 143, 145, 147, 149, 151, 153, 184, 189, 194, 197, 200, 203, 206, 208, 210, 213, 238, 239, 241, 243, 286–292, 300, 306, 308, 309, 311, 314, 317, 319, 321, 323, 326
PMIX_USOCK_DISABLE, **72**, 114, 123
PMIX_VERSION_INFO, **71**
PMIX_WAIT, **79**, 147, 148, 150, 288
PMIX_WDIR, **83**, 156, 161, 293