



Process Management Interface for Exascale (PMIx) Standard

Version 4.0 (Draft)

1H2019

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. Background | 1 |
| 1.2. PMIx Architecture Overview | 2 |
| 1.3. Portability of Functionality | 3 |
| 1.3.1. Optional Nature of Attributes | 3 |
| 1.4. Organization of this document | 5 |
| 1.5. Version 1.0: June 12, 2015 | 5 |
| 1.6. Version 2.0: Sept. 2018 | 6 |
| 1.7. Version 2.1: Dec. 2018 | 7 |
| 1.8. Version 2.2: Jan 2019 | 8 |
| 1.9. Version 3.0: Dec. 2018 | 8 |
| 1.10. Version 3.1: Jan. 2019 | 9 |
| 1.11. Version 4.0: June 2019 | 10 |
| 2. PMIx Terms and Conventions | 11 |
| 2.1. Notational Conventions | 13 |
| 2.2. Semantics | 14 |
| 2.3. Naming Conventions | 14 |
| 2.4. Procedure Conventions | 15 |
| 2.5. Standard vs Reference Implementation | 15 |
| 3. Data Structures and Types | 16 |
| 3.1. Constants | 17 |
| 3.1.1. PMIx Error Constants | 18 |
| 3.1.2. Macros for use with PMIx constants | 21 |
| 3.2. Data Types | 21 |
| 3.2.1. Key Structure | 21 |
| 3.2.2. Namespace Structure | 22 |
| 3.2.3. Rank Structure | 23 |

| | | |
|---------|---|----|
| 3.2.4. | Process Structure | 24 |
| 3.2.5. | Process structure support macros | 24 |
| 3.2.6. | Process State Structure | 26 |
| 3.2.7. | Process Information Structure | 27 |
| 3.2.8. | Process Information Structure support macros | 27 |
| 3.2.9. | Scope of Put Data | 29 |
| 3.2.10. | Range of Published Data | 29 |
| 3.2.11. | Data Persistence Structure | 30 |
| 3.2.12. | Data Array Structure | 30 |
| 3.2.13. | Data array structure support macros | 30 |
| 3.2.14. | Value Structure | 32 |
| 3.2.15. | Value structure support macros | 33 |
| 3.2.16. | Info Structure | 36 |
| 3.2.17. | Info structure support macros | 37 |
| 3.2.18. | Info Type Directives | 39 |
| 3.2.19. | Info Directive support macros | 40 |
| 3.2.20. | Job Allocation Directives | 42 |
| 3.2.21. | IO Forwarding Channels | 42 |
| 3.2.22. | Environmental Variable Structure | 42 |
| 3.2.23. | Environmental variable support macros | 43 |
| 3.2.24. | Lookup Returned Data Structure | 44 |
| 3.2.25. | Lookup data structure support macros | 44 |
| 3.2.26. | Application Structure | 47 |
| 3.2.27. | App structure support macros | 48 |
| 3.2.28. | Query Structure | 49 |
| 3.2.29. | Query structure support macros | 49 |
| 3.2.30. | Attribute registration structure | 51 |
| 3.2.31. | Attribute registration structure support macros | 52 |
| 3.2.32. | PMIx Group Directives | 54 |
| 3.2.33. | Byte Object Type | 54 |
| 3.2.34. | Byte object support macros | 54 |
| 3.2.35. | Data Buffer Type | 56 |
| 3.2.36. | Data buffer support macros | 56 |

| | | |
|---------|--|----|
| 3.2.37. | Data Array Structure | 58 |
| 3.2.38. | Data array support macros | 58 |
| 3.3. | Generalized Data Types Used for Packing/Unpacking | 60 |
| 3.4. | Reserved attributes | 61 |
| 3.4.1. | Initialization attributes | 62 |
| 3.4.2. | Tool-related attributes | 62 |
| 3.4.3. | Identification attributes | 63 |
| 3.4.4. | Programming model attributes | 64 |
| 3.4.5. | UNIX socket rendezvous socket attributes | 64 |
| 3.4.6. | TCP connection attributes | 65 |
| 3.4.7. | Global Data Storage (GDS) attributes | 65 |
| 3.4.8. | General process-level attributes | 65 |
| 3.4.9. | Scratch directory attributes | 66 |
| 3.4.10. | Relative Rank Descriptive Attributes | 66 |
| 3.4.11. | Information retrieval attributes | 67 |
| 3.4.12. | Information storage attributes | 68 |
| 3.4.13. | Size information attributes | 69 |
| 3.4.14. | Memory information attributes | 70 |
| 3.4.15. | Topology information attributes | 70 |
| 3.4.16. | Request-related attributes | 71 |
| 3.4.17. | Server-to-PMIx library attributes | 72 |
| 3.4.18. | Server-to-Client attributes | 73 |
| 3.4.19. | Event handler registration and notification attributes | 73 |
| 3.4.20. | Fault tolerance attributes | 74 |
| 3.4.21. | Spawn attributes | 74 |
| 3.4.22. | Query attributes | 77 |
| 3.4.23. | Log attributes | 78 |
| 3.4.24. | Debugger attributes | 79 |
| 3.4.25. | Resource manager attributes | 80 |
| 3.4.26. | Environment variable attributes | 80 |
| 3.4.27. | Job Allocation attributes | 80 |
| 3.4.28. | Job control attributes | 82 |
| 3.4.29. | Monitoring attributes | 83 |

| | | |
|-----------|---|------------|
| 3.4.30. | Security attributes | 84 |
| 3.4.31. | IO Forwarding attributes | 84 |
| 3.4.32. | Application setup attributes | 85 |
| 3.4.33. | Attribute support level attributes | 85 |
| 3.4.34. | Descriptive attributes | 85 |
| 3.4.35. | Process group attributes | 86 |
| 3.5. | Callback Functions | 86 |
| 3.5.1. | Release Callback Function | 87 |
| 3.5.2. | Modex Callback Function | 87 |
| 3.5.3. | Spawn Callback Function | 88 |
| 3.5.4. | Op Callback Function | 89 |
| 3.5.5. | Lookup Callback Function | 89 |
| 3.5.6. | Value Callback Function | 90 |
| 3.5.7. | Info Callback Function | 90 |
| 3.5.8. | Event Handler Registration Callback Function | 91 |
| 3.5.9. | Notification Handler Completion Callback Function | 92 |
| 3.5.10. | Notification Function | 93 |
| 3.5.11. | Server Setup Application Callback Function | 94 |
| 3.5.12. | Server Direct Modex Response Callback Function | 95 |
| 3.5.13. | PMIx Client Connection Callback Function | 96 |
| 3.5.14. | PMIx Tool Connection Callback Function | 96 |
| 3.5.15. | Credential callback function | 97 |
| 3.5.16. | Credential validation callback function | 98 |
| 3.5.17. | IOF delivery function | 99 |
| 3.5.18. | IOF and Event registration function | 100 |
| 3.6. | Constant String Functions | 101 |
| 4. | Initialization and Finalization | 104 |
| 4.1. | Query | 104 |
| 4.1.1. | PMIx_Initialized | 104 |
| 4.1.2. | PMIx_Get_version | 105 |
| 4.2. | Client Initialization and Finalization | 105 |
| 4.2.1. | PMIx_Init | 105 |
| 4.2.2. | PMIx_Finalize | 108 |

| | | |
|-----------|--|------------|
| 4.3. | Tool Initialization and Finalization | 108 |
| 4.3.1. | PMIx_tool_init | 108 |
| 4.3.2. | PMIx_tool_finalize | 112 |
| 4.3.3. | PMIx_tool_connect_to_server | 112 |
| 4.4. | Server Initialization and Finalization | 114 |
| 4.4.1. | PMIx_server_init | 114 |
| 4.4.2. | PMIx_server_finalize | 116 |
| 5. | Key/Value Management | 117 |
| 5.1. | Setting and Accessing Key/Value Pairs | 117 |
| 5.1.1. | PMIx_Put | 117 |
| 5.1.2. | PMIx_Get | 118 |
| 5.1.3. | PMIx_Get_nb | 121 |
| 5.1.4. | PMIx_Store_internal | 124 |
| 5.1.5. | Accessing information: examples | 125 |
| 5.2. | Exchanging Key/Value Pairs | 129 |
| 5.2.1. | PMIx_Commit | 130 |
| 5.2.2. | PMIx_Fence | 130 |
| 5.2.3. | PMIx_Fence_nb | 132 |
| 5.3. | Publish and Lookup Data | 135 |
| 5.3.1. | PMIx_Publish | 135 |
| 5.3.2. | PMIx_Publish_nb | 137 |
| 5.3.3. | PMIx_Lookup | 138 |
| 5.3.4. | PMIx_Lookup_nb | 140 |
| 5.3.5. | PMIx_Unpublish | 142 |
| 5.3.6. | PMIx_Unpublish_nb | 143 |
| 6. | Process Management | 146 |
| 6.1. | Abort | 146 |
| 6.1.1. | PMIx_Abort | 146 |
| 6.2. | Process Creation | 147 |
| 6.2.1. | PMIx_Spawn | 147 |
| 6.2.2. | PMIx_Spawn_nb | 152 |

| | | |
|-----------|--|------------|
| 6.3. | Connecting and Disconnecting Processes | 156 |
| 6.3.1. | PMIx_Connect | 156 |
| 6.3.2. | PMIx_Connect_nb | 159 |
| 6.3.3. | PMIx_Disconnect | 160 |
| 6.3.4. | PMIx_Disconnect_nb | 162 |
| 6.4. | IO Forwarding | 164 |
| 6.4.1. | PMIx_IOF_pull | 165 |
| 6.4.2. | PMIx_IOF_deregister | 167 |
| 6.4.3. | PMIx_IOF_push | 168 |
| 7. | Job Management and Reporting | 171 |
| 7.1. | Query | 171 |
| 7.1.1. | PMIx_Resolve_peers | 171 |
| 7.1.2. | PMIx_Resolve_nodes | 172 |
| 7.1.3. | PMIx_Query_info_nb | 173 |
| 7.2. | Allocation Requests | 179 |
| 7.2.1. | PMIx_Allocation_request | 180 |
| 7.2.2. | PMIx_Allocation_request_nb | 182 |
| 7.3. | Job Control | 185 |
| 7.3.1. | PMIx_Job_control | 186 |
| 7.3.2. | PMIx_Job_control_nb | 188 |
| 7.4. | Process and Job Monitoring | 191 |
| 7.4.1. | PMIx_Process_monitor | 191 |
| 7.4.2. | PMIx_Process_monitor_nb | 193 |
| 7.4.3. | PMIx_Heartbeat | 195 |
| 7.5. | Logging | 196 |
| 7.5.1. | PMIx_Log | 196 |
| 7.5.2. | PMIx_Log_nb | 198 |
| 8. | Event Notification | 202 |
| 8.1. | Notification and Management | 202 |
| 8.1.1. | PMIx_Register_event_handler | 204 |
| 8.1.2. | PMIx_Deregister_event_handler | 207 |
| 8.1.3. | PMIx_Notify_event | 208 |

| | |
|---|------------|
| 9. Data Packing and Unpacking | 211 |
| 9.1. Support Macros | 211 |
| 9.1.1. <code>PMIX_DATA_BUFFER_CREATE</code> | 211 |
| 9.1.2. <code>PMIX_DATA_BUFFER_RELEASE</code> | 211 |
| 9.1.3. <code>PMIX_DATA_BUFFER_CONSTRUCT</code> | 212 |
| 9.1.4. <code>PMIX_DATA_BUFFER_DESTRUCT</code> | 212 |
| 9.1.5. <code>PMIX_DATA_BUFFER_LOAD</code> | 213 |
| 9.1.6. <code>PMIX_DATA_BUFFER_UNLOAD</code> | 213 |
| 9.2. General Routines | 214 |
| 9.2.1. <code>PMIx_Data_pack</code> | 214 |
| 9.2.2. <code>PMIx_Data_unpack</code> | 215 |
| 9.2.3. <code>PMIx_Data_copy</code> | 217 |
| 9.2.4. <code>PMIx_Data_print</code> | 218 |
| 9.2.5. <code>PMIx_Data_copy_payload</code> | 218 |
| | |
| 10. Security | 220 |
| 10.1. Obtaining Credentials | 221 |
| 10.1.1. <code>PMIx_Get_credential</code> | 221 |
| 10.2. Validating Credentials | 223 |
| 10.2.1. <code>PMIx_Validate_credential</code> | 223 |
| | |
| 11. Server-Specific Interfaces | 225 |
| 11.1. Server Support Functions | 225 |
| 11.1.1. <code>PMIx_generate_regex</code> | 225 |
| 11.1.2. <code>PMIx_generate_ppn</code> | 226 |
| 11.1.3. <code>PMIx_server_register_nspace</code> | 227 |
| 11.1.4. <code>PMIx_server_deregister_nspace</code> | 239 |
| 11.1.5. <code>PMIx_server_register_client</code> | 241 |
| 11.1.6. <code>PMIx_server_deregister_client</code> | 242 |
| 11.1.7. <code>PMIx_server_setup_fork</code> | 243 |
| 11.1.8. <code>PMIx_server_dmodex_request</code> | 243 |
| 11.1.9. <code>PMIx_server_setup_application</code> | 245 |
| 11.1.10. <code>PMIx_Register_attributes</code> | 247 |
| 11.1.11. <code>PMIx_server_setup_local_support</code> | 248 |

| | | |
|----------|---|------------|
| 11.1.12. | PMIx_server_IOF_deliver | 250 |
| 11.1.13. | PMIx_server_collect_inventory | 251 |
| 11.1.14. | PMIx_server_deliver_inventory | 252 |
| 11.2. | Server Function Pointers | 253 |
| 11.2.1. | pmix_server_module_t Module | 253 |
| 11.2.2. | pmix_server_client_connected_fn_t | 254 |
| 11.2.3. | pmix_server_client_finalized_fn_t | 256 |
| 11.2.4. | pmix_server_abort_fn_t | 257 |
| 11.2.5. | pmix_server_fence_nb_fn_t | 258 |
| 11.2.6. | pmix_server_dmodex_req_fn_t | 262 |
| 11.2.7. | pmix_server_publish_fn_t | 263 |
| 11.2.8. | pmix_server_lookup_fn_t | 265 |
| 11.2.9. | pmix_server_unpublish_fn_t | 267 |
| 11.2.10. | pmix_server_spawn_fn_t | 269 |
| 11.2.11. | pmix_server_connect_fn_t | 274 |
| 11.2.12. | pmix_server_disconnect_fn_t | 276 |
| 11.2.13. | pmix_server_register_events_fn_t | 277 |
| 11.2.14. | pmix_server_deregister_events_fn_t | 279 |
| 11.2.15. | pmix_server_notify_event_fn_t | 281 |
| 11.2.16. | pmix_server_listener_fn_t | 282 |
| 11.2.17. | pmix_server_query_fn_t | 283 |
| 11.2.18. | pmix_server_tool_connection_fn_t | 285 |
| 11.2.19. | pmix_server_log_fn_t | 286 |
| 11.2.20. | pmix_server_alloc_fn_t | 288 |
| 11.2.21. | pmix_server_job_control_fn_t | 291 |
| 11.2.22. | pmix_server_monitor_fn_t | 294 |
| 11.2.23. | pmix_server_get_cred_fn_t | 296 |
| 11.2.24. | pmix_server_validate_cred_fn_t | 298 |
| 11.2.25. | pmix_server_iof_fn_t | 300 |
| 11.2.26. | pmix_server_stdin_fn_t | 303 |
| | 12. Process Sets and Groups | 305 |
| 12.1. | Process Sets | 305 |

| | |
|--|------------|
| 12.2. Process Groups | 306 |
| 12.2.1. <code>PMIx_Group_construct</code> | 308 |
| 12.2.2. <code>PMIx_Group_construct_nb</code> | 312 |
| 12.2.3. <code>PMIx_Group_destruct</code> | 315 |
| 12.2.4. <code>PMIx_Group_destruct_nb</code> | 317 |
| 12.2.5. <code>PMIx_Group_invite</code> | 319 |
| 12.2.6. <code>PMIx_Group_invite_nb</code> | 323 |
| 12.2.7. <code>PMIx_Group_join</code> | 325 |
| 12.2.8. <code>PMIx_Group_join_nb</code> | 328 |
| 12.2.9. <code>PMIx_Group_leave</code> | 330 |
| 12.2.10. <code>PMIx_Group_leave_nb</code> | 331 |
| A. Acknowledgements | 333 |
| A.1. Version 3.0 | 333 |
| A.2. Version 2.0 | 334 |
| A.3. Version 1.0 | 335 |
| Bibliography | 336 |
| Index | 337 |

CHAPTER 1

Introduction

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

14 1.1 Background

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

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

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

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

1.2 PMIx Architecture Overview

The presentation of the PMIx APIs within this document makes some basic assumptions about how these APIs are used and implemented. These assumptions are generally made only to simplify the presentation and explain PMIx with the expectation that most readers have similar concepts on how computing systems are organized today. However, ultimately this document should only be assumed to define a set of APIs.

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

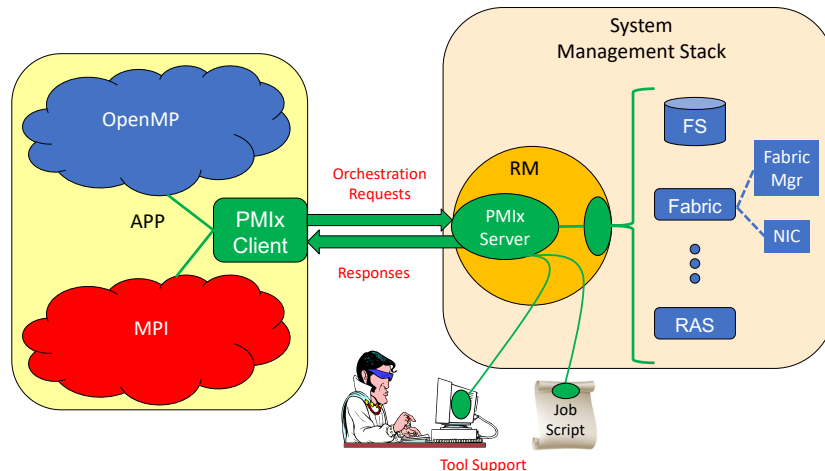


Figure 1.1.: PMIx-SMS Interactions

Fig. 1.1 shows a typical PMIx implementation in which 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. PMIx clients are processes which are started through the PMIx infrastructure, either by the PMIx implementation directly or through an system management software stack (SMS) component, and have registered as clients. A PMIx client is created in such a way that the PMIx client library will have sufficient information available to authenticate with the PMIx server. The PMIx server will have sufficient knowledge about the process which it created, either directly or through other SMS, to authenticate the process and provide information the process requests such as its identity and the identity of its peers.

1 As clients invoke PMIx APIs, is possible that some client requests can be handled at the client level.
2 Other requests might require communication with the local PMIx server, which subsequently might
3 request services from the host SMS (represented here by an resource manager (RM) daemon).

4 The interaction between the PMIx server and SMS might be achieved using callback functions
5 registered during server initialization. The host SMS could indicate its lack of support for any
6 operation by simply providing a *NULL* for the associated callback function, or could create a
7 function entry that returns *not supported* when called. These interactions between a PMIx
8 implementation and the SMS are not defined by the PMIx standard.

9 Fig. 1.1 shows how tools can interact with the PMIx architecture. Tools, whether standalone or
10 embedded in job scripts, are an exception to the normal client registration process. A process can
11 register as a tool, provided the PMIx client library has adequate rendezvous information to connect
12 to the appropriate PMIx server. This allows processes which were not created by the PMIx
13 infrastructure to request access to PMIx functionality. Processes registered as tools do not have
14 peers.

15 1.3 Portability of Functionality

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

26 To help address this issue, the PMIx APIs are designed to allow resource managers and other
27 system management stack components to decide on support of a particular function and allow client
28 applications to query and adjust to the level of support available. The PMIx community continues
29 to look at ways to assist SMS implementers in their decisions on what functionality to support by
30 highlighting functions and attributes that are critical to basic application execution (e.g.,
31 [PMIx_Get](#)) for certain classes of applications.

32 1.3.1 Optional Nature of Attributes

33 An area where differences between support on different systems can be challenging is regarding the
34 attributes that provide information to the client process and/or control the behavior of a PMIx API.
35 Most PMIx API calls can accept additional information or attributes specified in the form of
36 key/value pairs. These attributes provide information to the PMIx implementation which influence

1 the behavior of the API call. In addition to API calls being optional, support for the individual
2 attributes of an API call can vary between systems or implementations.

3 An application can adapt to the attribute support on a particular system in one of two ways. PMIx
4 provides an API to enable an application to query the attributes supported by a particular API (See
5 7.1.3.2). Through this API, the PMIx implementation can provide detailed information about the
6 attributes supported on a system for each API call queried. Alternatively, the application can mark
7 attributes as required using a flag within the `pmix_info_t` (See 3.2.16). If the required attribute
8 is not available on the system or the desired value for the attribute is not available, the call will
9 return the error code for *not supported*.

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

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

19 It is therefore critical that users and application implementers:

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

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

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

39 While a PMIx library implementer, or an SMS component server, may choose to support a

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

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

9 1.4 Organization of this document

10 The remainder of this document is structured as follows:

- 11 • Introduction and Overview in Chapter 1 on page 1
- 12 • Terms and Conventions in Chapter 2 on page 11
- 13 • Data Structures and Types in Chapter 3 on page 16
- 14 • PMIx Initialization and Finalization in Chapter 4 on page 104
- 15 • Key/Value Management in Chapter 5 on page 117
- 16 • Process Management in Chapter 6 on page 146
- 17 • Job Management in Chapter 7 on page 171
- 18 • Event Notification in Chapter 8 on page 202
- 19 • Data Packing and Unpacking in Chapter 9 on page 211
- 20 • PMIx Server Specific Interfaces in Chapter 11 on page 225

21 1.5 Version 1.0: June 12, 2015

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

- 25 • Client APIs
 - 26 – `PMIx_Init`, `PMIx_Initialized`, `PMIx_Abort`, `PMIx_Finalize`
 - 27 – `PMIx_Put`, `PMIx_Commit`,
 - 28 – `PMIx_Fence`, `PMIx_Fence_nb`
 - 29 – `PMIx_Get`, `PMIx_Get_nb`
 - 30 – `PMIx_Publish`, `PMIx_Publish_nb`

- 1 – `PMIx_Lookup` , `PMIx_Lookup`
- 2 – `PMIx_Unpublish` , `PMIx_Unpublish_nb`
- 3 – `PMIx_Spawn` , `PMIx_Spawn_nb`
- 4 – `PMIx_Connect` , `PMIx_Connect_nb`
- 5 – `PMIx_Disconnect` , `PMIx_Disconnect_nb`
- 6 – `PMIx_Resolve_nodes` , `PMIx_Resolve_peers`
- 7 • Server APIs
- 8 – `PMIx_server_init` , `PMIx_server_finalize`
- 9 – `PMIx_generate_regex` , `PMIx_generate_ppn`
- 10 – `PMIx_server_register_nspace` , `PMIx_server_deregister_nspace`
- 11 – `PMIx_server_register_client` , `PMIx_server_deregister_client`
- 12 – `PMIx_server_setup_fork` , `PMIx_server_dmodex_request`
- 13 • Common APIs
- 14 – `PMIx_Get_version` , `PMIx_Store_internal` , `PMIx_Error_string`
- 15 – `PMIx_Register_errhandler` , `PMIx_Deregister_errhandler` , `PMIx_Notify_error`
- 16 The `PMIx_Init` API was subsequently modified in the PRI release v1.1.0.

17 **1.6 Version 2.0: Sept. 2018**

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

- 19 • Client APIs
- 20 – `PMIx_Query_info_nb` , `PMIx_Log_nb`
- 21 – `PMIx_Allocation_request_nb` , `PMIx_Job_control_nb` ,
- 22 – `PMIx_Process_monitor_nb` , `PMIx_Heartbeat`
- 23 • Server APIs
- 24 – `PMIx_server_setup_application` , `PMIx_server_setup_local_support`
- 25 • Tool APIs
- 26 – `PMIx_tool_init` , `PMIx_tool_finalize`
- 27 • Common APIs
- 28 – `PMIx_Register_event_handler` , `PMIx_Deregister_event_handler`

- 1 – `PMIx_Notify_event`
- 2 – `PMIx_Proc_state_string`, `PMIx_Scope_string`
- 3 – `PMIx_Persistence_string`, `PMIx_Data_range_string`
- 4 – `PMIx_Info_directives_string`, `PMIx_Data_type_string`
- 5 – `PMIx_Alloc_directive_string`
- 6 – `PMIx_Data_pack`, `PMIx_Data_unpack`, `PMIx_Data_copy`
- 7 – `PMIx_Data_print`, `PMIx_Data_copy_payload`

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

12 1.7 Version 2.1: Dec. 2018

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

- 15 • Clarify description of `PMIx_Connect` and `PMIx_Disconnect` APIs.
- 16 • Explain that values for the `PMIX_COLLECTIVE_ALGO` are environment-dependent
- 17 • Identify the namespace/rank values required for retrieving attribute-associated information using
 18 the `PMIx_Get` API
- 19 • Provide definitions for `session`, `job`, `application`, and other terms used throughout the
 20 document
- 21 • Clarify definitions of `PMIX_UNIV_SIZE` versus `PMIX_JOB_SIZE`
- 22 • Clarify server module function return values
- 23 • Provide examples of the use of `PMIx_Get` for retrieval of information
- 24 • Clarify the use of `PMIx_Get` versus `PMIx_Query_info_nb`
- 25 • Clarify return values for non-blocking APIs and emphasize that callback functions must not be
 26 invoked prior to return from the API
- 27 • Provide detailed example for construction of the `PMIx_server_register_namespace` input
 28 information array
- 29 • Define information levels (e.g., `session` vs `job`) and associated attributes for both storing
 30 and retrieving values
- 31 • 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`

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

5 1.10 Version 3.1: Jan. 2019

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

- 7 • Direct modex upcall function (`pmix_server_dmodex_req_fn_t`) cannot complete
8 atomically as the API cannot return the requested information except via the provided callback
9 function
- 10 • Fix typo in name of `PMIX_FWD_STDDIAG` attribute
- 11 • Correctly identify the information retrieval and storage attributes as “new” to v3 of the standard
- 12 • Add missing `pmix_data_array_t` definition and support macros
- 13 • Add a rule divider between implementer and host environment required attributes for clarity
- 14 • Add `PMIX_QUERY_QUALIFIERS_CREATE` macro to simplify creation of `pmix_query_t`
15 qualifiers
- 16 • Add `PMIX_APP_INFO_CREATE` macro to simplify creation of `pmix_app_t` directives
- 17 • Add new attributes to specify the level of information being requested where ambiguity may exist
18 (see 3.4.11)
- 19 • Add new attributes to assemble information by its level for storage where ambiguity may exist
20 (see 3.4.12)
- 21 • Add flag and `PMIX_INFO_IS_END` macro for marking and detecting the end of a
22 `pmix_info_t` array
- 23 • Clarify that `PMIX_NUM_SLOTS` is duplicative of (a) `PMIX_UNIV_SIZE` when used at the
24 `session` level and (b) `PMIX_MAX_PROCS` when used at the `job` and `application`
25 levels, but leave it in for backward compatibility.
- 26 • Clarify difference between `PMIX_JOB_SIZE` and `PMIX_MAX_PROCS`
- 27 • Clarify that `PMIx_server_setup_application` must be called per- `job` instead of per-
28 `application` as the name implies. Unfortunately, this is a historical artifact. Note that both
29 `PMIX_NODE_MAP` and `PMIX_PROC_MAP` must be included as input in the `info` array provided
30 to that function. Further descriptive explanation of the “instant on” procedure will be provided in
31 the next version of the PMIx Standard.
- 32 • Clarify how the PMIx server expects data passed to the host by
33 `pmix_server_fencenb_fn_t` should be aggregated across nodes, and provide a code
34 snippet example

1 1.11 Version 4.0: June 2019

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

- 3 • Clarified that the **PMIx_Fence_nb** operation can immediately return
4 **PMIX_OPERATION_SUCCEEDED** in lieu of passing the request to a PMIx server if only the
5 calling process is involved in the operation
- 6 • Added the **PMIx_Register_attributes** API by which a host environment can register
7 the attributes it supports for each server-to-host operation
- 8 • Added the ability to query supported attributes from the PMIx tool, client and server libraries, as
9 well as the host environment via the new **pmix_regattr_t** structure. Both human-readable
10 and machine-parsable output is supported. New attributes to support this operation include:
 - 11 – **PMIX_CLIENT_ATTRIBUTES** , **PMIX_SERVER_ATTRIBUTES** ,
12 **PMIX_TOOL_ATTRIBUTES** , and **PMIX_HOST_ATTRIBUTES** to identify which library
13 supports the attribute; and
 - 14 – **PMIX_MAX_VALUE** , **PMIX_MIN_VALUE** , and **PMIX_ENUM_VALUE** to provide
15 machine-parsable description of accepted values

CHAPTER 2

PMIx Terms and Conventions

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

7 The PMIx community has further adopted a policy that modification of existing released APIs will
8 only be permitted under extreme circumstances. In its effort to avoid introduction of any such
9 backward incompatibility, the community has avoided the definitions of large numbers of APIs that
10 each focus on a narrow scope of functionality, and instead relied on the definition of fewer generic
11 APIs that include arrays of directives for “tuning” the function’s behavior. Thus, modifications to
12 the PMIx standard increasingly consist of the definition of new attributes along with a description
13 of the APIs to which they relate and the expected behavior when used with those APIs.

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

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

Advice to users

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

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

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

- 10 • *session* refers to an allocated set of resources assigned to a particular user by the system WLM.
11 Historically, High Performance Computing (HPC) sessions have consisted of a static allocation
12 of resources - i.e., a block of resources are assigned to a user in response to a specific request and
13 managed as a unified collection. However, this is changing in response to the growing use of
14 dynamic programming models that require on-the-fly allocation and release of system resources.
15 Accordingly, the term *session* in this document refers to the current block of assigned resources
16 and is a potentially dynamic entity.
- 17 • *slot* refers to an allocated entry for a process. WLMs frequently allocate entire nodes to a
18 *session*, but can also be configured to define the maximum number of processes that can
19 simultaneously be executed on each node. This often corresponds to the number of hardware
20 Processing Units (PUs) (typically cores, but can also be defined as hardware threads) on the
21 node. However, the correlation between hardware PUs and slot allocations strictly depends upon
22 system configuration.
- 23 • *job* refers to a set of one or more *applications* executed as a single invocation by the user within a
24 session. For example, “*mpiexec -n 1 app1 : -n 2 app2*” is considered a single Multiple Program
25 Multiple Data (MPMD) job containing two applications.
- 26 • *namespace* refers to a character string value assigned by the RM to a *job*. All *applications*
27 executed as part of that *job* share the same *namespace*. The *namespace* assigned to each *job* must
28 be unique within the scope of the governing RM.
- 29 • *application* refers to a single executable (binary, script, etc.) member of a *job*. Applications
30 consist of one or more *processes*, either operating independently or in parallel at any given time
31 during their execution.
- 32 • *rank* refers to the numerical location (starting from zero) of a process within the defined scope.
33 Thus, global rank is the rank of a process within its *job*, while *application rank* is the rank of that
34 process within its *application*.
- 35 • *workflow* refers to an orchestrated execution plan frequently spanning multiple *jobs* carried out
36 under the control of a *workflow manager* process. An example workflow might first execute a
37 computational job to generate the flow of liquid through a complex cavity, followed by a
38 visualization job that takes the output of the first job as its input to produce an image output.

- *resource manager* is used in a generic sense to represent the system 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.

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_namespace** .
- 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.

1 2.4 Procedure Conventions

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

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

11 2.5 Standard vs Reference Implementation

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

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

CHAPTER 3

Data Structures and Types

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

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

▼ Advice to PMIx library implementers ▼

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

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



▼ Advice to users ▼

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

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

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

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

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

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

5 3.1 Constants

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

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

▼ Advice to PMIx library implementers ▼

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

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

▼ Advice to PMIx library implementers ▼

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

1 3.1.1 PMIx Error Constants

2 The `pmix_status_t` structure is an `int` type for return status.

3 The tables shown in this section define the possible values for `pmix_status_t`. PMIx errors are
4 required to always be negative, with 0 reserved for `PMIX_SUCCESS`. Values in the list that were
5 deprecated in later standards are denoted as such. Values added to the list in this version of the
6 standard are shown in **magenta**.

▼ Advice to PMIx library implementers ▼

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



▼ Advice to users ▼

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



13 3.1.1.1 General Error Constants

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

| | | |
|----|---|---|
| 15 | <code>PMIX_SUCCESS</code> | Success |
| 16 | <code>PMIX_ERROR</code> | General Error |
| 17 | <code>PMIX_ERR_SILENT</code> | Silent error |
| 18 | <code>PMIX_ERR_DEBUGGER_RELEASE</code> | Error in debugger release |
| 19 | <code>PMIX_ERR_PROC_RESTART</code> | Fault tolerance: Error in process restart |
| 20 | <code>PMIX_ERR_PROC_CHECKPOINT</code> | Fault tolerance: Error in process checkpoint |
| 21 | <code>PMIX_ERR_PROC_MIGRATE</code> | Fault tolerance: Error in process migration |
| 22 | <code>PMIX_ERR_PROC_ABORTED</code> | Process was aborted |
| 23 | <code>PMIX_ERR_PROC_REQUESTED_ABORT</code> | Process is already requested to abort |
| 24 | <code>PMIX_ERR_PROC_ABORTING</code> | Process is being aborted |
| 25 | <code>PMIX_ERR_SERVER_FAILED_REQUEST</code> | Failed to connect to the server |
| 26 | <code>PMIX_EXISTS</code> | Requested operation would overwrite an existing value |
| 27 | <code>PMIX_ERR_INVALID_CRED</code> | Invalid security credentials |
| 28 | <code>PMIX_ERR_HANDSHAKE_FAILED</code> | Connection handshake failed |
| 29 | <code>PMIX_ERR_READY_FOR_HANDSHAKE</code> | Ready for handshake |
| 30 | <code>PMIX_ERR_WOULD_BLOCK</code> | Operation would block |
| 31 | <code>PMIX_ERR_UNKNOWN_DATA_TYPE</code> | Unknown data type |
| 32 | <code>PMIX_ERR_PROC_ENTRY_NOT_FOUND</code> | Process not found |
| 33 | <code>PMIX_ERR_TYPE_MISMATCH</code> | Invalid type |
| 34 | <code>PMIX_ERR_UNPACK_INADEQUATE_SPACE</code> | Inadequate space to unpack data |

1 **PMIX_ERR_UNPACK_FAILURE** Unpack failed
2 **PMIX_ERR_PACK_FAILURE** Pack failed
3 **PMIX_ERR_PACK_MISMATCH** Pack mismatch
4 **PMIX_ERR_NO_PERMISSIONS** No permissions
5 **PMIX_ERR_TIMEOUT** Timeout expired
6 **PMIX_ERR_UNREACH** Unreachable
7 **PMIX_ERR_IN_ERRNO** Error defined in **errno**
8 **PMIX_ERR_BAD_PARAM** Bad parameter
9 **PMIX_ERR_RESOURCE_BUSY** Resource busy
10 **PMIX_ERR_OUT_OF_RESOURCE** Resource exhausted
11 **PMIX_ERR_DATA_VALUE_NOT_FOUND** Data value not found
12 **PMIX_ERR_INIT** Error during initialization
13 **PMIX_ERR_NOMEM** Out of memory
14 **PMIX_ERR_INVALID_ARG** Invalid argument
15 **PMIX_ERR_INVALID_KEY** Invalid key
16 **PMIX_ERR_INVALID_KEY_LENGTH** Invalid key length
17 **PMIX_ERR_INVALID_VAL** Invalid value
18 **PMIX_ERR_INVALID_VAL_LENGTH** Invalid value length
19 **PMIX_ERR_INVALID_LENGTH** Invalid argument length
20 **PMIX_ERR_INVALID_NUM_ARGS** Invalid number of arguments
21 **PMIX_ERR_INVALID_ARGS** Invalid arguments
22 **PMIX_ERR_INVALID_NUM_PARSED** Invalid number parsed
23 **PMIX_ERR_INVALID_KEYVALP** Invalid key/value pair
24 **PMIX_ERR_INVALID_SIZE** Invalid size
25 **PMIX_ERR_INVALID_NAMESPACE** Invalid namespace
26 **PMIX_ERR_SERVER_NOT_AVAIL** Server is not available
27 **PMIX_ERR_NOT_FOUND** Not found
28 **PMIX_ERR_NOT_SUPPORTED** Not supported
29 **PMIX_ERR_NOT_IMPLEMENTED** Not implemented
30 **PMIX_ERR_COMM_FAILURE** Communication failure
31 **PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER** Unpacking past the end of the buffer
32 provided
33 **PMIX_ERR_LOST_CONNECTION_TO_SERVER** Lost connection to server
34 **PMIX_ERR_LOST_PEER_CONNECTION** Lost connection to peer
35 **PMIX_ERR_LOST_CONNECTION_TO_CLIENT** Lost connection to client
36 **PMIX_QUERY_PARTIAL_SUCCESS** Query partial success (used by query system)
37 **PMIX_NOTIFY_ALLOC_COMPLETE** Notify that allocation is complete
38 **PMIX_JCTRL_CHECKPOINT** Job control: Monitored by PMIx client to trigger checkpoint
39 operation
40 **PMIX_JCTRL_CHECKPOINT_COMPLETE** Job control: Sent by PMIx client and monitored
41 by PMIx server to notify that requested checkpoint operation has completed.
42 **PMIX_JCTRL_PREEMPT_ALERT** Job control: Monitored by PMIx client to detect an RM
43 intending to preempt the job.

1 **PMIX_MONITOR_HEARTBEAT_ALERT** Job monitoring: Heartbeat alert
2 **PMIX_MONITOR_FILE_ALERT** Job monitoring: File alert
3 **PMIX_PROC_TERMINATED** Process terminated - can be either normal or abnormal
4 termination
5 **PMIX_ERR_INVALID_TERMINATION** Process terminated without calling
6 **PMIx_Finalize**, or was a member of an assemblage formed via **PMIx_Connect** and
7 terminated or called **PMIx_Finalize** without first calling **PMIx_Disconnect** (or its
8 non-blocking form) from that assemblage.

9 3.1.1.2 Operational Error Constants

10 **PMIX_ERR_EVENT_REGISTRATION** Error in event registration
11 **PMIX_ERR_JOB_TERMINATED** Error job terminated
12 **PMIX_ERR_UPDATE_ENDPOINTS** Error updating endpoints
13 **PMIX_MODEL_DECLARED** Model declared
14 **PMIX_GDS_ACTION_COMPLETE** The global data storage (GDS) action has completed
15 **PMIX_ERR_INVALID_OPERATION** The requested operation is supported by the
16 implementation and host environment, but fails to meet a requirement (e.g., requesting to
17 *disconnect* from processes without first *connecting* to them).
18 **PMIX_PROC_HAS_CONNECTED** A tool or client has connected to the PMIx server
19 **PMIX_CONNECT_REQUESTED** Connection has been requested by a PMIx-based tool
20 **PMIX_MODEL_RESOURCES** Resource usage by a programming model has changed
21 **PMIX_OPENMP_PARALLEL_ENTERED** An OpenMP parallel code region has been entered
22 **PMIX_OPENMP_PARALLEL_EXITED** An OpenMP parallel code region has completed
23 **PMIX_LAUNCH_DIRECTIVE** Launcher directives have been received from a PMIx-enabled
24 tool
25 **PMIX_LAUNCHER_READY** Application launcher (e.g., mpiexec) is ready to receive directives
26 from a PMIx-enabled tool
27 **PMIX_OPERATION_IN_PROGRESS** A requested operation is already in progress
28 **PMIX_OPERATION_SUCCEEDED** The requested operation was performed atomically - no
29 callback function will be executed
30 **PMIX_ERR_DUPLICATE_KEY** The provided key has already been published on a different
31 data range

32 3.1.1.3 System error constants

33 **PMIX_ERR_NODE_DOWN** Node down
34 **PMIX_ERR_NODE_OFFLINE** Node is marked as offline
35 **PMIX_ERR_SYS_OTHER** Mark the beginning of a dedicated range of constants for system
36 event reporting.

37 3.1.1.4 Event handler error constants

38 **PMIX_EVENT_NO_ACTION_TAKEN** Event handler: No action taken
39 **PMIX_EVENT_PARTIAL_ACTION_TAKEN** Event handler: Partial action taken
40 **PMIX_EVENT_ACTION_DEFERRED** Event handler: Action deferred
41 **PMIX_EVENT_ACTION_COMPLETE** Event handler: Action complete

1 3.1.1.5 User-Defined Error Constants

2 PMIx establishes an error code boundary for constants defined in the PMIx standard. Negative
3 values larger than this (and any positive values greater than zero) are guaranteed not to conflict with
4 PMIx values.

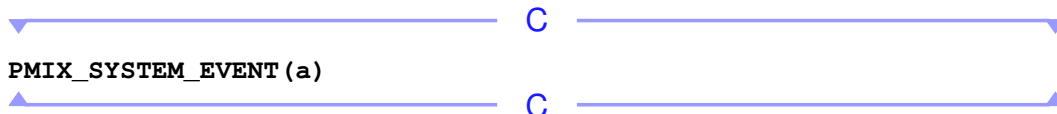
5 **PMIX_EXTERNAL_ERR_BASE** A starting point for user-level defined error constants.
6 Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions
7 should always be based on the **PMIX_EXTERNAL_ERR_BASE** constant and not a specific
8 value as the value of the constant may change.

9 3.1.2 Macros for use with PMIx constants

10 3.1.2.1 Detect system event constant

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

PMIx v2.2

13 **PMIX_SYSTEM_EVENT** (a) 

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

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

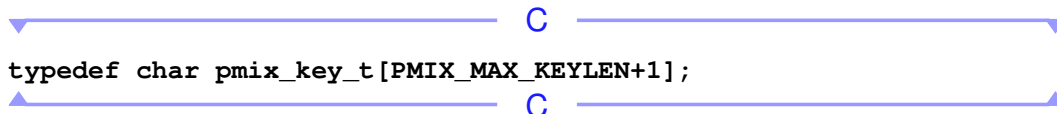
18 3.2 Data Types

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

21 3.2.1 Key Structure

22 The **pmix_key_t** structure is a statically defined character array of length **PMIX_MAX_KEYLEN**
23 +1, thus supporting keys of maximum length **PMIX_MAX_KEYLEN** while preserving space for a
24 mandatory **NULL** terminator.

PMIx v2.0

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

26 Characters in the key must be standard alphanumeric values supported by common utilities such as
27 *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

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

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

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

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

3.2.2.1 Namespace support macro

Compare the string in a `pmix_namespace_t` to a given value

PMIx v3.0

```
PMIX_CHECK_NAMESPACE(a, b)
```

IN a

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

```
typedef uint32_t pmix_rank_t;
```

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

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

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

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

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

6 3.2.4 Process Structure

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

PMIx v1.0

```

9 typedef struct pmix_proc {
10     pmix_namespace_t nspace;
11     pmix_rank_t rank;
12 } pmix_proc_t;
  
```

13 3.2.5 Process structure support macros

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

15 3.2.5.1 Initialize the `pmix_proc_t` structure

16 **PMIX_PROC_CONSTRUCT**

17 Initialize the `pmix_proc_t` fields

PMIx v1.0

```

18 PMIX_PROC_CONSTRUCT (m)
  
```

19 **IN** `m`

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

21 3.2.5.2 Destruct the `pmix_proc_t` structure

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

1 3.2.5.3 Create a `pmix_proc_t` array

2 Allocate and initialize an array of `pmix_proc_t` structures

PMIx v1.0

▼ C ▼

3 `PMIX_PROC_CREATE(m, n)`

▲ C ▲

4 **INOUT** `m`

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

6 **IN** `n`

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

8 3.2.5.4 Free a `pmix_proc_t` array

9 Release an array of `pmix_proc_t` structures

PMIx v1.0

▼ C ▼

10 `PMIX_PROC_FREE(m, n)`

▲ C ▲

11 **IN** `m`

12 Pointer to the array of `pmix_proc_t` structures (handle)

13 **IN** `n`

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

15 3.2.5.5 Load a `pmix_proc_t` structure

16 Load values into a `pmix_proc_t`

PMIx v2.0

▼ C ▼

17 `PMIX_PROC_LOAD(m, n, r)`

▲ C ▲

18 **IN** `m`

19 Pointer to the structure to be loaded (pointer to `pmix_proc_t`)

20 **IN** `n`

21 Namespace to be loaded (`pmix_namespace_t`)

22 **IN** `r`

23 Rank to be assigned (`pmix_rank_t`)

1 3.2.5.6 Compare identifiers

2 Compare two `pmix_proc_t` identifiers

PMIx v3.0

3 `PMIX_CHECK_PROCID(a, b)`

4 **IN a**

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

6 **IN b**

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

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

- 9 • the ranks are the same value
- 10 • one of the ranks is `PMIX_RANK_WILDCARD`

11 3.2.6 Process State Structure

12 *PMIx v2.0* The `pmix_proc_state_t` structure is a `uint8_t` type for process state values. The following
13 constants can be used to set a variable of the type `pmix_proc_state_t`. All values were
14 originally defined in version 2 of the standard unless otherwise marked.

Advice to users

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

| | | |
|----|--|---|
| 18 | <code>PMIX_PROC_STATE_UNDEF</code> | Undefined process state |
| 19 | <code>PMIX_PROC_STATE_PREPPED</code> | Process is ready to be launched |
| 20 | <code>PMIX_PROC_STATE_LAUNCH_UNDERWAY</code> | Process launch is underway |
| 21 | <code>PMIX_PROC_STATE_RESTART</code> | Process is ready for restart |
| 22 | <code>PMIX_PROC_STATE_TERMINATE</code> | Process is marked for termination |
| 23 | <code>PMIX_PROC_STATE_RUNNING</code> | Process has been locally fork 'ed by the RM |
| 24 | <code>PMIX_PROC_STATE_CONNECTED</code> | Process has connected to PMIx server |
| 25 | <code>PMIX_PROC_STATE_UNTERMINATED</code> | Define a "boundary" between the terminated states 26 and <code>PMIX_PROC_STATE_CONNECTED</code> so users can easily and quickly determine if a 27 process is still running or not. Any value less than this constant means that the process has not 28 terminated. |
| 29 | <code>PMIX_PROC_STATE_TERMINATED</code> | Process has terminated and is no longer running |
| 30 | <code>PMIX_PROC_STATE_ERROR</code> | Define a boundary so users can easily and quickly determine if 31 a process abnormally terminated. Any value above this constant means that the process has 32 terminated abnormally. |

```

1     PMIX_PROC_STATE_KILLED_BY_CMD    Process was killed by a command
2     PMIX_PROC_STATE_ABORTED        Process was aborted by a call to PMIx_Abort
3     PMIX_PROC_STATE_FAILED_TO_START Process failed to start
4     PMIX_PROC_STATE_ABORTED_BY_SIG Process aborted by a signal
5     PMIX_PROC_STATE_TERM_WO_SYNC    Process exited without calling PMIx_Finalize
6     PMIX_PROC_STATE_COMM_FAILED     Process communication has failed
7     PMIX_PROC_STATE_CALLED_ABORT    Process called PMIx_Abort
8     PMIX_PROC_STATE_MIGRATING       Process failed and is waiting for resources before
9     restarting
10    PMIX_PROC_STATE_CANNOT_RESTART   Process failed and cannot be restarted
11    PMIX_PROC_STATE_TERM_NON_ZERO    Process exited with a non-zero status
12    PMIX_PROC_STATE_FAILED_TO_LAUNCH Unable to launch process

```

13 3.2.7 Process Information Structure

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

PMIx v2.0

```

16 typedef struct pmix_proc_info {
17     /** Process structure */
18     pmix_proc_t proc;
19     /** Hostname where process resides */
20     char *hostname;
21     /** Name of the executable */
22     char *executable_name;
23     /** Process ID on the host */
24     pid_t pid;
25     /** Exit code of the process. Default: 0 */
26     int exit_code;
27     /** Current state of the process */
28     pmix_proc_state_t state;
29 } pmix_proc_info_t;

```

30 3.2.8 Process Information Structure support macros

31 The following macros are provided to support the `pmix_proc_info_t` structure.

1 **3.2.8.1 Initialize the `pmix_proc_info_t` structure**

2 Initialize the `pmix_proc_info_t` fields

PMIx v2.0



3 **PMIX_PROC_INFO_CONSTRUCT** (*m*)



4 **IN** *m*

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

6 **3.2.8.2 Destruct the `pmix_proc_info_t` structure**

7 Destruct the `pmix_proc_info_t` fields

PMIx v2.0



8 **PMIX_PROC_INFO_DESTRUCT** (*m*)



9 **IN** *m*

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

11 **3.2.8.3 Create a `pmix_proc_info_t` array**

12 Allocate and initialize a `pmix_proc_info_t` array

PMIx v2.0



13 **PMIX_PROC_INFO_CREATE** (*m*, *n*)



14 **INOUT** *m*

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

17 **IN** *n*

18 Number of structures to be allocated (**size_t**)

19 **3.2.8.4 Free a `pmix_proc_info_t` array**

20 Release an array of `pmix_proc_info_t` structures

PMIx v2.0



21 **PMIX_PROC_INFO_FREE** (*m*, *n*)



22 **IN** *m*

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

24 **IN** *n*

25 Number of structures in the array (**size_t**)

1 3.2.9 Scope of Put Data

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

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

8 `PMIX_SCOPE_UNDEF` Undefined scope

9 `PMIX_LOCAL` The data is intended only for other application processes on the same node.

10 Data marked in this way will not be included in data packages sent to remote requestors —
11 i.e., it is only available to processes on the local node.

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

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

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

19 3.2.10 Range of Published Data

20 *PMIx v1.0* The `pmix_data_range_t` structure is a `uint8_t` type that defines a range for data *published*
21 via functions other than `PMIx_Put` - e.g., the `PMIx_Publish` API. The following constants
22 can be used to set a variable of the type `pmix_data_range_t`. Several values were initially
23 defined in version 1 of the standard but subsequently renamed and other values added in version 2.
24 Thus, all values shown below are as they were defined in version 2 except where noted.

25 `PMIX_RANGE_UNDEF` Undefined range

26 `PMIX_RANGE_RM` Data is intended for the host resource manager.

27 `PMIX_RANGE_LOCAL` Data is only available to processes on the local node.

28 `PMIX_RANGE_NAMESPACE` Data is only available to processes in the same namespace.

29 `PMIX_RANGE_SESSION` Data is only available to all processes in the session.

30 `PMIX_RANGE_GLOBAL` Data is available to all processes.

31 `PMIX_RANGE_CUSTOM` Range is specified in the `pmix_info_t` associated with this call.

32 `PMIX_RANGE_PROC_LOCAL` Data is only available to this process.

33 `PMIX_RANGE_INVALID` Invalid value

Advice to users

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

1 3.2.11 Data Persistence Structure

2 *PMIx v1.0* The `pmix_persistence_t` structure is a `uint8_t` type that defines the policy for data
3 published by clients via the `PMIx_Publish` API. The following constants can be used to set a
4 variable of the type `pmix_persistence_t`. All definitions were introduced in version 1 of the
5 standard unless otherwise marked.

6 `PMIX_PERSIST_INDEF` Retain data until specifically deleted.
7 `PMIX_PERSIST_FIRST_READ` Retain data until the first access, then the data is deleted.
8 `PMIX_PERSIST_PROC` Retain data until the publishing process terminates.
9 `PMIX_PERSIST_APP` Retain data until the application terminates.
10 `PMIX_PERSIST_SESSION` Retain data until the session/allocation terminates.
11 `PMIX_PERSIST_INVALID` Invalid value

12 3.2.12 Data Array Structure

PMIx v2.0

```
13 typedef struct pmix_data_array  
14     pmix_data_type_t type;  
15     size_t size;  
16     void *array;  
17     pmix_data_array_t;
```

C

C

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

20 3.2.13 Data array structure support macros

21 The following macros are provided to support the `pmix_data_array_t` structure.

22 3.2.13.1 Initialize the `pmix_data_array_t` structure

23 Initialize the `pmix_data_array_t` fields, allocating memory for the array itself.

PMIx v2.2

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

C

C

25 **IN** `m`
26 Pointer to the structure to be initialized (pointer to `pmix_data_array_t`)
27 **IN** `n`
28 Number of elements in the array (`size_t`)
29 **IN** `t`
30 `PMIx` data type for the array elements (`pmix_data_type_t`)

1 3.2.13.2 Destruct the `pmix_data_array_t` structure

2 Destruct the `pmix_data_array_t` fields, releasing the array's memory.

PMIx v2.2

▼ `C` _____ ▼

3 **PMIX_DATA_ARRAY_DESTRUCT** (m)

▲ _____ ▲ `C` _____ ▲

4 **IN** m

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

6 3.2.13.3 Create and initialize a `pmix_data_array_t` object

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

PMIx v2.2

▼ _____ ▼ `C` _____ ▼

9 **PMIX_DATA_ARRAY_CREATE** (m, n, t)

▲ _____ ▲ `C` _____ ▲

10 **INOUT** m

11 Address where the pointer to the `pmix_data_array_t` structure shall be stored (handle)

12 **IN** n

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

14 **IN** t

15 PMIx data type for the array elements (`pmix_data_type_t`)

16 3.2.13.4 Free a `pmix_data_array_t` object

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

PMIx v2.2

▼ _____ ▼ `C` _____ ▼

18 **PMIX_DATA_ARRAY_FREE** (m)

▲ _____ ▲ `C` _____ ▲

19 **IN** m

20 Pointer to the `pmix_data_array_t` structure (handle)

1 3.2.14 Value Structure

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

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

PMIx v1.0

C

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

```
1     } data;
2 } pmix_value_t;
```

▲  C  ▼

3 3.2.15 Value structure support macros

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

5 3.2.15.1 Initialize the `pmix_value_t` structure

6 Initialize the `pmix_value_t` fields

PMIx v1.0

▼  C  ▼

7 **PMIX_VALUE_CONSTRUCT** (m)

▲  C  ▼

8 **IN** m

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

10 3.2.15.2 Destruct the `pmix_value_t` structure

11 Destruct the `pmix_value_t` fields

PMIx v1.0

▼  C  ▼

12 **PMIX_VALUE_DESTRUCT** (m)

▲  C  ▼

13 **IN** m

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

15 3.2.15.3 Create a `pmix_value_t` array

16 Allocate and initialize an array of `pmix_value_t` structures

PMIx v1.0

▼  C  ▼

17 **PMIX_VALUE_CREATE** (m, n)

▲  C  ▼

18 **INOUT** m

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

20 **IN** n

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

1 3.2.15.4 Free a `pmix_value_t` array

2 Release an array of `pmix_value_t` structures

PMIx v1.0

▼ `C` _____ ▼

3 **PMIX_VALUE_FREE**(`m`, `n`)

▲ _____ `C` _____ ▲

4 **IN** `m`

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

6 **IN** `n`

Number of structures in the array (`size_t`)

8 3.2.15.5 Load a value structure

9 **Summary**

10 Load data into a `pmix_value_t` structure.

PMIx v2.0

▼ _____ `C` _____ ▼

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

▲ _____ `C` _____ ▲

12 **IN** `v`

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

14 **IN** `d`

Pointer to the data value to be loaded (handle)

16 **IN** `t`

Type of the provided data value (`pmix_data_type_t`)

18 **Description**

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

▼ **Advice to users** _____ ▼

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

▲ _____ ▲

1 3.2.15.6 Unload a `pmix_value_t` structure

2 Summary

3 Unload data from a `pmix_value_t` structure.

PMIx v2.2

▼ `C` _____ ▼
▲ `PMIX_VALUE_UNLOAD(r, v, d, t);` _____ ▲

5 **OUT** `r`

6 Status code indicating result of the operation `pmix_status_t`

7 **IN** `v`

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

9 **INOUT** `d`

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

11 **INOUT** `t`

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

13 Description

14 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.
▲ _____ ▲

17 3.2.15.7 Transfer data between `pmix_value_t` structures

18 Summary

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

PMIx v2.0

▼ `C` _____ ▼
▲ `PMIX_VALUE_XFER(r, d, s);` _____ ▲

21 **OUT** `r`

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

23 **IN** `d`

24 Pointer to the `pmix_value_t` destination (handle)

25 **IN** `s`

26 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

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

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

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

1 3.2.17 Info structure support macros

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

3 3.2.17.1 Initialize the `pmix_info_t` structure

4 Initialize the `pmix_info_t` fields

PMIx v1.0 ▼ C _____ ▼

5 **PMIX_INFO_CONSTRUCT** (m)

▲ _____ C _____ ▲

6 **IN** m

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

8 3.2.17.2 Destruct the `pmix_info_t` structure

9 Destruct the `pmix_info_t` fields

PMIx v1.0 ▼ C _____ ▼

10 **PMIX_INFO_DESTRUCT** (m)

▲ _____ C _____ ▲

11 **IN** m

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

13 3.2.17.3 Create a `pmix_info_t` array

14 Allocate and initialize an array of `pmix_info_t` structures

PMIx v1.0 ▼ C _____ ▼

15 **PMIX_INFO_CREATE** (m, n)

▲ _____ C _____ ▲

16 **INOUT** m

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

18 **IN** n

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

20 3.2.17.4 Free a `pmix_info_t` array

21 Release an array of `pmix_info_t` structures

PMIx v1.0 ▼ C _____ ▼

22 **PMIX_INFO_FREE** (m, n)

▲ _____ C _____ ▲

23 **IN** m

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

25 **IN** n

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

1 3.2.17.5 Load key and value data into a `pmix_info_t`

PMIx v1.0

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

3 **IN** v

4 Pointer to the `pmix_info_t` into which the key and data are to be loaded (pointer to
5 `pmix_info_t`)

6 **IN** k

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

9 **IN** d

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

11 **IN** t

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

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

Advice to users

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

18 3.2.17.6 Copy data between `pmix_info_t` structures

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

PMIx v2.0

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

21 **IN** d

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

23 **IN** s

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

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

Advice to users

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

1 3.2.17.7 Test a boolean `pmix_info_t`

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

PMIx v2.0

▼ `C` ▼

3 `PMIX_INFO_TRUE(m)`

▲ `C` ▲

4 **IN** `m`

5 Pointer to a `pmix_info_t` structure (handle)

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

- 7
- the structure reports a type of `PMIX_UNDEF`, or
 - the structure reports a type of `PMIX_BOOL` and the data flag is `true`
- 8

9 3.2.18 Info Type Directives

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

▼ **Advice to users** ▼

13 A PMIx implementation or PMIx-enabled RM may ignore any `pmix_info_t` value passed to a
14 PMIx API if it is not explicitly marked as `PMIX_INFO_REQD`. This is because the values
15 specified default to optional, meaning they can be ignored. This may lead to unexpected behavior if
16 the user is relying on the behavior specified by the `pmix_info_t` value. If the user relies on the
17 behavior defined by the `pmix_info_t` then they must set the `PMIX_INFO_REQD` flag using the
18 `PMIX_INFO_REQUIRED` macro.

▲

▼ **Advice to PMIx library implementers** ▼

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

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

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

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

Advice to PMIx server hosts

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

10 3.2.19 Info Directive support macros

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

12 3.2.19.1 Mark an info structure as required

13 Summary

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

PMIx v2.0

15 `PMIX_INFO_REQUIRED(info);`

16 **IN** `info`

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

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

19 3.2.19.2 Mark an info structure as optional

20 Summary

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

PMIx v2.0

22 `PMIX_INFO_OPTIONAL(info);`

23 **IN** `info`

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

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

1 3.2.19.3 Test an info structure for *required* directive

2 Summary

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

PMIx v2.0

```
▼ _____ C _____ ▼  
4 PMIX_INFO_IS_REQUIRED(info);  
▲ _____ C _____ ▲
```

5 **IN** `info`

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

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

8 3.2.19.4 Test an info structure for *optional* directive

9 Summary

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

PMIx v2.0

```
▼ _____ C _____ ▼  
11 PMIX_INFO_IS_OPTIONAL(info);  
▲ _____ C _____ ▲
```

12 **IN** `info`

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

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

16 3.2.19.5 Test an info structure for *end of array* directive

17 Summary

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

PMIx v2.2

```
▼ _____ C _____ ▼  
20 PMIX_INFO_IS_END(info);  
▲ _____ C _____ ▲
```

21 **IN** `info`

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

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

1 3.2.20 Job Allocation Directives

2 *PMIx v2.0* The `pmix_alloc_directive_t` structure is a `uint8_t` type that defines the behavior of
3 allocation requests. The following constants can be used to set a variable of the type
4 `pmix_alloc_directive_t`. All definitions were introduced in version 2 of the standard
5 unless otherwise marked.

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

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

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

13 **PMIX_ALLOC_REACQUIRE** Reacquire resources that were previously “lent” back to the
14 scheduler.

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

17 3.2.21 IO Forwarding Channels

18 *PMIx v3.0* The `pmix_iof_channel_t` structure is a `uint16_t` type that defines a set of bit-mask flags
19 for specifying IO forwarding channels. These can be bitwise OR’d together to reference multiple
20 channels.

21 **PMIX_FWD_NO_CHANNELS** Forward no channels

22 **PMIX_FWD_STDIN_CHANNEL** Forward stdin

23 **PMIX_FWD_STDOUT_CHANNEL** Forward stdout

24 **PMIX_FWD_STDERR_CHANNEL** Forward stderr

25 **PMIX_FWD_STDDIAG_CHANNEL** Forward stddiag, if available

26 **PMIX_FWD_ALL_CHANNELS** Forward all available channels

27 3.2.22 Environmental Variable Structure

28 *PMIx v3.0* Define a structure for specifying environment variable modifications. Standard environment
29 variables (e.g., **PATH**, **LD_LIBRARY_PATH**, and **LD_PRELOAD**) take multiple arguments
30 separated by delimiters. Unfortunately, the delimiters depend upon the variable itself - some use
31 semi-colons, some colons, etc. Thus, the operation requires not only the name of the variable to be
32 modified and the value to be inserted, but also the separator to be used when composing the
33 aggregate value.

```

1 typedef struct
2     char *envar;
3     char *value;
4     char separator;
5     pmix_envar_t;

```

6 3.2.23 Environmental variable support macros

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

8 3.2.23.1 Initialize the `pmix_envar_t` structure

9 Initialize the `pmix_envar_t` fields

```

10 PMIx v3.0
11 PMIX_ENVAR_CONSTRUCT (m)
12
13 IN m
14     Pointer to the structure to be initialized (pointer to pmix_envar_t)

```

13 3.2.23.2 Destruct the `pmix_envar_t` structure

14 Clear the `pmix_envar_t` fields

```

15 PMIx v3.0
16 PMIX_ENVAR_DESTRUCT (m)
17
18 IN m
19     Pointer to the structure to be destructed (pointer to pmix_envar_t)

```

18 3.2.23.3 Create a `pmix_envar_t` array

19 Allocate and initialize an array of `pmix_envar_t` structures

```

20 PMIx v3.0
21 PMIX_ENVAR_CREATE (m, n)
22
23 INOUT m
24     Address where the pointer to the array of pmix_envar_t structures shall be stored (handle)
25
26 IN n
27     Number of structures to be allocated (size_t)

```

1 3.2.23.4 Free a `pmix_envar_t` array

2 Release an array of `pmix_envar_t` structures

PMIx v3.0

▼  

3 `PMIX_ENVAR_FREE(m, n)`

▲  

4 **IN** `m`

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

6 **IN** `n`

Number of structures in the array (`size_t`)

8 3.2.23.5 Load a `pmix_envar_t` structure

9 Load values into a `pmix_envar_t`

PMIx v2.0

▼  

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

▲  

11 **IN** `m`

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

13 **IN** `e`

Environmental variable name (`char*`)

15 **IN** `v`

Value of variable (`char*`)

17 **IN** `v`

Separator character (`char`)

19 3.2.24 Lookup Returned Data Structure

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

PMIx v1.0

▼  

21 `typedef struct pmix_pdata {`

`pmix_proc_t proc;`

`pmix_key_t key;`

`pmix_value_t value;`

25 `} pmix_pdata_t;`

▲  

26 3.2.25 Lookup data structure support macros

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

1 3.2.25.1 Initialize the `pmix_pdata_t` structure

2 Initialize the `pmix_pdata_t` fields

PMIx v1.0

▼  C 

3 `PMIX_PDATA_CONSTRUCT (m)`

▲  C 

4 **IN** `m`

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

6 3.2.25.2 Destruct the `pmix_pdata_t` structure

7 Destruct the `pmix_pdata_t` fields

PMIx v1.0

▼  C 

8 `PMIX_PDATA_DESTRUCT (m)`

▲  C 

9 **IN** `m`

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

11 3.2.25.3 Create a `pmix_pdata_t` array

12 Allocate and initialize an array of `pmix_pdata_t` structures

PMIx v1.0

▼  C 

13 `PMIX_PDATA_CREATE (m, n)`

▲  C 

14 **INOUT** `m`

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

16 **IN** `n`

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

18 3.2.25.4 Free a `pmix_pdata_t` array

19 Release an array of `pmix_pdata_t` structures

PMIx v1.0

▼  C 

20 `PMIX_PDATA_FREE (m, n)`

▲  C 

21 **IN** `m`

22 Pointer to the array of `pmix_pdata_t` structures (handle)

23 **IN** `n`

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

1 3.2.25.5 Load a lookup data structure

2 Summary

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

PMIx v1.0

C

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

C

5 **IN** `m`

6 Pointer to the `pmix_pdata_t` structure into which the key and data are to be loaded
7 (pointer to `pmix_pdata_t`)

8 **IN** `p`

9 Pointer to the `pmix_proc_t` structure containing the identifier of the process being
10 referenced (pointer to `pmix_proc_t`)

11 **IN** `k`

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

14 **IN** `d`

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

16 **IN** `t`

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

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

Advice to users

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

1 3.2.25.6 Transfer a lookup data structure

2 Summary

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

PMIx v2.0

C

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

C

5 **IN** `d`

6 Pointer to the destination `pmix_pdata_t` (pointer to `pmix_pdata_t`)

7 **IN** `s`

8 Pointer to the source `pmix_pdata_t` (pointer to `pmix_pdata_t`)

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

Advice to users

10 Key, process identifier, and data will all be copied into the destination `pmix_pdata_t` - thus, the
11 source `pmix_pdata_t` may free'd without affecting the copied data once the macro has
12 completed.

13 3.2.26 Application Structure

14 The `pmix_app_t` structure describes the application context for the `PMIx_Spawn` and
15 `PMIx_Spawn_nb` operations.

PMIx v1.0

C

```
16 typedef struct pmix_app {  
17     /** Executable */  
18     char *cmd;  
19     /** Argument set, NULL terminated */  
20     char **argv;  
21     /** Environment set, NULL terminated */  
22     char **env;  
23     /** Current working directory */  
24     char *cwd;  
25     /** Maximum processes with this profile */  
26     int maxprocs;  
27     /** Array of info keys describing this application*/  
28     pmix_info_t *info;  
29     /** Number of info keys in 'info' array */  
30     size_t ninfo;  
31 } pmix_app_t;
```

C

1 3.2.27 App structure support macros

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

3 3.2.27.1 Initialize the `pmix_app_t` structure

4 Initialize the `pmix_app_t` fields

PMIx v1.0 ▼ `PMIX_APP_CONSTRUCT` (m) C

5 `PMIX_APP_CONSTRUCT` (m)

6 **IN** m

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

8 3.2.27.2 Destruct the `pmix_app_t` structure

9 Destruct the `pmix_app_t` fields

PMIx v1.0 ▼ `PMIX_APP_DESTRUCT` (m) C

10 `PMIX_APP_DESTRUCT` (m)

11 **IN** m

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

13 3.2.27.3 Create a `pmix_app_t` array

14 Allocate and initialize an array of `pmix_app_t` structures

PMIx v1.0 ▼ `PMIX_APP_CREATE` (m, n) C

15 `PMIX_APP_CREATE` (m, n)

16 **INOUT** m

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

18 **IN** n

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

20 3.2.27.4 Free a `pmix_app_t` array

21 Release an array of `pmix_app_t` structures

PMIx v1.0 ▼ `PMIX_APP_FREE` (m, n) C

22 `PMIX_APP_FREE` (m, n)

23 **IN** m

24 Pointer to the array of `pmix_app_t` structures (handle)

25 **IN** n

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

1 3.2.27.5 Create the `pmix_info_t` array of application directives

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

PMIx v2.2

▼ C ————— ▼

4 `PMIX_APP_INFO_CREATE(m, n)`

▲ C ————— ▲

5 **IN** `m`
6 Pointer to the `pmix_app_t` structure (handle)

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

9 3.2.28 Query Structure

10 The `pmix_query_t` structure is used by `PMIx_Query_info_nb` to describe a single query
11 operation.

PMIx v2.0

▼ C ————— ▼

```
12 typedef struct pmix_query {  
13     char **keys;  
14     pmix_info_t *qualifiers;  
15     size_t nqual;  
16 } pmix_query_t;
```

▲ C ————— ▲

17 3.2.29 Query structure support macros

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

19 3.2.29.1 Initialize the `pmix_query_t` structure

20 Initialize the `pmix_query_t` fields

PMIx v2.0

▼ C ————— ▼

21 `PMIX_QUERY_CONSTRUCT(m)`

▲ C ————— ▲

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

1 3.2.29.2 Destruct the `pmix_query_t` structure

2 Destruct the `pmix_query_t` fields

PMIx v2.0

▼  C 

3 **PMIX_QUERY_DESTRUCT** (m)

▲  C 

4 **IN** m

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

6 3.2.29.3 Create a `pmix_query_t` array

7 Allocate and initialize an array of `pmix_query_t` structures

PMIx v2.0

▼  C 

8 **PMIX_QUERY_CREATE** (m, n)

▲  C 

9 **INOUT** m

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

11 **IN** n

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

13 3.2.29.4 Free a `pmix_query_t` array

14 Release an array of `pmix_query_t` structures

PMIx v2.0

▼  C 

15 **PMIX_QUERY_FREE** (m, n)

▲  C 

16 **IN** m

17 Pointer to the array of `pmix_query_t` structures (handle)

18 **IN** n

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

20 3.2.29.5 Create the `pmix_info_t` array of query qualifiers

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

PMIx v2.2

▼  C 

23 **PMIX_QUERY_QUALIFIERS_CREATE** (m, n)

▲  C 

24 **IN** m

25 Pointer to the `pmix_query_t` structure (handle)

26 **IN** n

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

1 3.2.30 Attribute registration structure

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

PMIx v4.0

```
3 typedef struct pmix_regattr {  
4     char *name;  
5     pmix_key_t *string;  
6     pmix_data_type_t type;  
7     pmix_info_t *info;  
8     size_t ninfo;  
9     char **description;  
10 } pmix_regattr_t;
```

11 The *type*

12 Note that in this structure:

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

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

Advice to PMIx library implementers

31 Although not strictly required, PMIx library implementers are strongly encouraged to provide both
32 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

PMIX_REGATTR_CONSTRUCT (m)

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

PMIX_REGATTR_DESTRUCT (m)

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

PMIX_REGATTR_CREATE (m, n)

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`)

1 3.2.31.4 Free a `pmix_regattr_t` array

2 Release an array of `pmix_regattr_t` structures

PMIx v4.0

C

3 **PMIX_REGATTR_FREE**(*m*, *n*)

C

4 **INOUT** *m*

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

6 **IN** *n*

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

8 3.2.31.5 Load a `pmix_regattr_t` structure

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

PMIx v4.0

C

12 **PMIX_REGATTR_LOAD**(*a*, *n*, *k*, *t*, *ni*, *v*)

C

13 **IN** *a*

14 Pointer to the structure to be loaded (pointer to `pmix_proc_t`)

15 **IN** *n*

16 String name of the attribute (string)

17 **IN** *k*

18 Key value to be loaded (`pmix_key_t`)

19 **IN** *t*

20 Type of data associated with the provided key (`pmix_data_type_t`)

21 **IN** *ni*

22 Number of `pmix_info_t` elements to be allocated in *info* (**size_t**)

23 **IN** *v*

24 One-line description to be loaded (more can be added separately) (string)

25 3.2.31.6 Transfer a `pmix_regattr_t` to another `pmix_regattr_t`

26

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

PMIx v4.0

C

28 **PMIX_REGATTR_XFER**(*m*, *n*)

C

29 **INOUT** *m*

30 Pointer to the destination `pmix_regattr_t` structure (handle)

31 **IN** *n*

32 Pointer to the source `pmix_regattr_t` structure (handle)





1 3.2.32 PMIx Group Directives

2 *PMIx v4.0* The `pmix_group_opt_t` type is an enumerated type used with the `PMIx_Group_join` API
3 to indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

- 4 `PMIX_GROUP_DECLINE` Decline the invitation
- 5 `PMIX_GROUP_ACCEPT` Accept the invitation.

6 3.2.33 Byte Object Type

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





```
8 PMIx v1.0    
9 typedef struct pmix_byte_object {  
10     char *bytes;  
11     size_t size;  
12 } pmix_byte_object_t;  
13  
```

12 3.2.34 Byte object support macros

13 The following macros support the `pmix_byte_object_t` structure.

14 3.2.34.1 Initialize the `pmix_byte_object_t` structure





15 Initialize the `pmix_byte_object_t` fields

```
16 PMIx v2.0    
17 PMIX_BYTE_OBJECT_CONSTRUCT (m)  
18  
```

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

21 3.2.34.2 Destruct the `pmix_byte_object_t` structure

22 Clear the `pmix_byte_object_t` fields

```
23 PMIx v2.0    
24 PMIX_BYTE_OBJECT_DESTRUCT (m)  
25  
```

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

1 3.2.34.3 Create a `pmix_byte_object_t` structure

2 Allocate and initialize an array of `pmix_byte_object_t` structures

PMIx v2.0

▼ `C` _____ ▼

3 **PMIX_BYTE_OBJECT_CREATE**(*m*, *n*)

▲ _____ `C` _____ ▲

4 **INOUT** *m*

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

7 **IN** *n*

8 Number of structures to be allocated (**size_t**)

9 3.2.34.4 Free a `pmix_byte_object_t` array

10 Release an array of `pmix_byte_object_t` structures

PMIx v2.0

▼ `C` _____ ▼

11 **PMIX_BYTE_OBJECT_FREE**(*m*, *n*)

▲ _____ `C` _____ ▲

12 **IN** *m*

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

14 **IN** *n*

15 Number of structures in the array (**size_t**)

16 3.2.34.5 Load a `pmix_byte_object_t` structure

17 Load values into a `pmix_byte_object_t`

PMIx v2.0

▼ `C` _____ ▼

18 **PMIX_BYTE_OBJECT_LOAD**(*b*, *d*, *s*)

▲ _____ `C` _____ ▲

19 **IN** *b*

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

21 **IN** *d*

22 Pointer to the data to be loaded (**char***)

23 **IN** *s*

24 Number of bytes in the data array (**size_t**)

1 3.2.35 Data Buffer Type

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

PMIx v2.0

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

22 3.2.36 Data buffer support macros

23 The following macros support the `pmix_data_buffer_t` structure.

24 3.2.36.1 Initialize the `pmix_data_buffer_t` structure

25 Initialize the `pmix_data_buffer_t` fields

PMIx v2.0

```
26 PMIX_DATA_BUFFER_CONSTRUCT(m)
```

27 **IN** m

28 Pointer to the structure to be initialized (pointer to `pmix_data_buffer_t`)

1 **3.2.36.2 Destruct the `pmix_data_buffer_t` structure**

2 Clear the `pmix_data_buffer_t` fields

PMIx v2.0



C

3 **PMIX_DATA_BUFFER_DESTRUCT (m)**



C

4 **IN m**

5 Pointer to the structure to be destructed (pointer to `pmix_data_buffer_t`)

6 **3.2.36.3 Create a `pmix_data_buffer_t` structure**

7 Allocate and initialize a `pmix_data_buffer_t` structure

PMIx v2.0



C

8 **PMIX_DATA_BUFFER_CREATE (m)**



C

9 **INOUT m**

10 Address where the pointer to the `pmix_data_buffer_t` structure shall be stored (handle)

11 **3.2.36.4 Free a `pmix_data_buffer_t`**

12 Release a `pmix_data_buffer_t` structure

PMIx v2.0



C

13 **PMIX_DATA_BUFFER_RELEASE (m)**



C

14 **IN m**

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

16 **3.2.36.5 Load a `pmix_data_buffer_t`**

17 Load data into a `pmix_data_buffer_t` structure

PMIx v2.2



C

18 **PMIX_DATA_BUFFER_LOAD (b, d, s)**



C

19 **IN b**

20 Pointer to the `pmix_data_buffer_t` structure to be loaded (handle)

21 **IN d**

22 Pointer to the data to be loaded into *b* (**void***)

23 **IN s**

24 Number of bytes in *d* (**size_t**)

1 3.2.36.6 Unload a `pmix_data_buffer_t`

2 Unload the data from a `pmix_data_buffer_t` structure

PMIx v2.2

C

3 `PMIX_DATA_BUFFER_UNLOAD(b, d, s)`

C

4 **IN** `b`

Pointer to the `pmix_data_buffer_t` structure to be unloaded (handle)

6 **INOUT** `d`

Pointer to be set to the data region after unloading (**void***)

8 **INOUT** `s`

Variable to be set to the number of bytes in the returned data region (**size_t**)

10 3.2.37 Data Array Structure

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

PMIx v2.0

C

```
12 typedef struct pmix_data_array {  
13     pmix_data_type_t type;  
14     size_t size;  
15     void *array;  
16 } pmix_data_array_t;
```

C

17 3.2.38 Data array support macros

18 The following macros support the `pmix_data_array_t` structure.

19 3.2.38.1 Initialize a `pmix_data_array_t` structure

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

PMIx v2.2

C

21 `PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)`

C

22 **IN** `m`

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

24 **IN** `n`

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

26 **IN** `t`

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

1 3.2.38.2 Destruct a `pmix_data_array_t` structure

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

PMIx v2.2

▼  C 

3 **PMIX_DATA_ARRAY_CONSTRUCT** (m)

▲  C 

4 **IN** m

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

6 3.2.38.3 Create a `pmix_data_array_t` structure

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

PMIx v2.2

▼  C 

9 **PMIX_DATA_ARRAY_CREATE** (m, n, t)

▲  C 

10 **INOUT** m

11 Variable to be set to the address of the structure (pointer to `pmix_data_array_t`)

12 **IN** n

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

14 **IN** t

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

16 3.2.38.4 Free a `pmix_data_array_t` structure

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

PMIx v2.2

▼  C 

18 **PMIX_DATA_ARRAY_FREE** (m)

▲  C 

19 **IN** m

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

1 3.3 Generalized Data Types Used for Packing/Unpacking

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

Advice to PMIx library implementers

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

| | | |
|----|-------------------------------|---|
| 10 | <code>PMIX_UNDEF</code> | Undefined |
| 11 | <code>PMIX_BOOL</code> | Boolean (converted to/from native <code>true/false</code>) (<code>bool</code>) |
| 12 | <code>PMIX_BYTE</code> | A byte of data (<code>uint8_t</code>) |
| 13 | <code>PMIX_STRING</code> | <code>NULL</code> terminated string (<code>char*</code>) |
| 14 | <code>PMIX_SIZE</code> | Size <code>size_t</code> |
| 15 | <code>PMIX_PID</code> | Operating process identifier (PID) (<code>pid_t</code>) |
| 16 | <code>PMIX_INT</code> | Integer (<code>int</code>) |
| 17 | <code>PMIX_INT8</code> | 8-byte integer (<code>int8_t</code>) |
| 18 | <code>PMIX_INT16</code> | 16-byte integer (<code>int16_t</code>) |
| 19 | <code>PMIX_INT32</code> | 32-byte integer (<code>int32_t</code>) |
| 20 | <code>PMIX_INT64</code> | 64-byte integer (<code>int64_t</code>) |
| 21 | <code>PMIX_UINT</code> | Unsigned integer (<code>unsigned int</code>) |
| 22 | <code>PMIX_UINT8</code> | Unsigned 8-byte integer (<code>uint8_t</code>) |
| 23 | <code>PMIX_UINT16</code> | Unsigned 16-byte integer (<code>uint16_t</code>) |
| 24 | <code>PMIX_UINT32</code> | Unsigned 32-byte integer (<code>uint32_t</code>) |
| 25 | <code>PMIX_UINT64</code> | Unsigned 64-byte integer (<code>uint64_t</code>) |
| 26 | <code>PMIX_FLOAT</code> | Float (<code>float</code>) |
| 27 | <code>PMIX_DOUBLE</code> | Double (<code>double</code>) |
| 28 | <code>PMIX_TIMEVAL</code> | Time value (<code>struct timeval</code>) |
| 29 | <code>PMIX_TIME</code> | Time (<code>time_t</code>) |
| 30 | <code>PMIX_STATUS</code> | Status code <code>pmix_status_t</code> |
| 31 | <code>PMIX_VALUE</code> | Value (<code>pmix_value_t</code>) |
| 32 | <code>PMIX_PROC</code> | Process (<code>pmix_proc_t</code>) |
| 33 | <code>PMIX_APP</code> | Application context |
| 34 | <code>PMIX_INFO</code> | Info object |
| 35 | <code>PMIX_PDATA</code> | Pointer to data |
| 36 | <code>PMIX_BUFFER</code> | Buffer |
| 37 | <code>PMIX_BYTE_OBJECT</code> | Byte object (<code>pmix_byte_object_t</code>) |
| 38 | <code>PMIX_KVAL</code> | Key/value pair |

```

1  PMIX_PERSIST Persistence ( pmix\_persistence\_t )
2  PMIX_POINTER Pointer to an object (void*)
3  PMIX_SCOPE Scope ( pmix\_scope\_t )
4  PMIX_DATA_RANGE Range for data ( pmix\_data\_range\_t )
5  PMIX_COMMAND PMIx command code (used internally)
6  PMIX_INFO_DIRECTIVES Directives flag for pmix\_info\_t (
7  pmix\_info\_directives\_t )
8  PMIX_DATA_TYPE Data type code ( pmix\_data\_type\_t )
9  PMIX_PROC_STATE Process state ( pmix\_proc\_state\_t )
10 PMIX_PROC_INFO Process information ( pmix\_proc\_info\_t )
11 PMIX_DATA_ARRAY Data array ( pmix\_data\_array\_t )
12 PMIX_PROC_RANK Process rank ( pmix\_rank\_t )
13 PMIX_QUERY Query structure ( pmix\_query\_t )
14 PMIX_COMPRESSED_STRING String compressed with zlib (char*)
15 PMIX_ALLOC_DIRECTIVE Allocation directive ( pmix\_alloc\_directive\_t )
16 PMIX_IOF_CHANNEL Input/output forwarding channel ( pmix\_iof\_channel\_t )
17 PMIX_ENVAR Environmental variable structure ( pmix\_envar\_t )
18 PMIX_REGATTR Structure supporting attribute registrations ( pmix\_regattr\_t )

```

19 3.4 Reserved attributes

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

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

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

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

33 **PMIX_ATTR_UNDEF** NULL (NULL)
34 Constant representing an undefined attribute.

1 3.4.1 Initialization attributes

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

4 **PMIX_EVENT_BASE** "pmix.evbase" (struct event_base *)
5 Pointer to libevent¹ `event_base` to use in place of the internal progress thread.
6 **PMIX_SERVER_TOOL_SUPPORT** "pmix.srvr.tool" (bool)
7 The host RM wants to declare itself as willing to accept tool connection requests.
8 **PMIX_SERVER_REMOTE_CONNECTIONS** "pmix.srvr.remote" (bool)
9 Allow connections from remote tools. Forces the PMIx server to not exclusively use
10 loopback device.
11 **PMIX_SERVER_SYSTEM_SUPPORT** "pmix.srvr.sys" (bool)
12 The host RM wants to declare itself as being the local system server for PMIx connection
13 requests.
14 **PMIX_SERVER_TMPDIR** "pmix.srvr.tmpdir" (char*)
15 Top-level temporary directory for all client processes connected to this server, and where the
16 PMIx server will place its tool rendezvous point and contact information.
17 **PMIX_SYSTEM_TMPDIR** "pmix.sys.tmpdir" (char*)
18 Temporary directory for this system, and where a PMIx server that declares itself to be a
19 system-level server will place a tool rendezvous point and contact information.
20 **PMIX_SERVER_ENABLE_MONITORING** "pmix.srv.monitor" (bool)
21 Enable PMIx internal monitoring by the PMIx server.
22 **PMIX_SERVER_NAMESPACE** "pmix.srv.namespace" (char*)
23 Name of the namespace to use for this PMIx server.
24 **PMIX_SERVER_RANK** "pmix.srv.rank" (pmix_rank_t)
25 Rank of this PMIx server
26 **PMIX_SERVER_GATEWAY** "pmix.srv.gway" (bool)
27 Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
28 (e.g., logging to email)

29 3.4.2 Tool-related attributes

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

33 **PMIX_TOOL_NAMESPACE** "pmix.tool.namespace" (char*)
34 Name of the namespace to use for this tool.
35 **PMIX_TOOL_RANK** "pmix.tool.rank" (uint32_t)
36 Rank of this tool.
37 **PMIX_SERVER_PIDINFO** "pmix.srvr.pidinfo" (pid_t)
38 PID of the target PMIx server for a tool.
39 **PMIX_CONNECT_TO_SYSTEM** "pmix.cnct.sys" (bool)

¹<http://libevent.org/>

1 The requestor requires that a connection be made only to a local, system-level PMIx server.
2 **PMIX_CONNECT_SYSTEM_FIRST** "pmix.cnct.sys.first" (bool)
3 Preferentially, look for a system-level PMIx server first.
4 **PMIX_SERVER_URI** "pmix.srvr.uri" (char*)
5 uniform resource identifier (URI) of the PMIx server to be contacted.
6 **PMIX_SERVER_HOSTNAME** "pmix.srvr.host" (char*)
7 Host where target PMIx server is located.
8 **PMIX_CONNECT_MAX_RETRIES** "pmix.tool.mretries" (uint32_t)
9 Maximum number of times to try to connect to PMIx server.
10 **PMIX_CONNECT_RETRY_DELAY** "pmix.tool.retry" (uint32_t)
11 Time in seconds between connection attempts to a PMIx server.
12 **PMIX_TOOL_DO_NOT_CONNECT** "pmix.tool.nocon" (bool)
13 The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.
14 **PMIX_RECONNECT_SERVER** "pmix.tool.recon" (bool)
15 Tool is requesting to change server connections
16 **PMIX_LAUNCHER** "pmix.tool.launcher" (bool)
17 Tool is a launcher and needs rendezvous files created

18 3.4.3 Identification attributes

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

21 **PMIX_USERID** "pmix.euid" (uint32_t)
22 Effective user id.
23 **PMIX_GRPID** "pmix.egid" (uint32_t)
24 Effective group id.
25 **PMIX_DSTPATH** "pmix.dstpath" (char*)
26 Path to shared memory data storage (dstore) files.
27 **PMIX_VERSION_INFO** "pmix.version" (char*)
28 PMIx version of contractor.
29 **PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)
30 The requesting process is a PMIx tool.
31 **PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)
32 The requesting process is a PMIx client.
33 **PMIX_PSET_NAME** "pmix.pset.nm" (char*)
34 User-assigned name for the process set containing the given process.

1 3.4.4 Programming model attributes

2 These attributes are associated with programming models.

3 **PMIX_PROGRAMMING_MODEL** "pmix.pgm.model" (char*)
4 Programming model being initialized (e.g., "MPI" or "OpenMP")
5 **PMIX_MODEL_LIBRARY_NAME** "pmix.mdl.name" (char*)
6 Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
7 **PMIX_MODEL_LIBRARY_VERSION** "pmix.mdl.vrs" (char*)
8 Programming model version string (e.g., "2.1.1")
9 **PMIX_THREADING_MODEL** "pmix.threads" (char*)
10 Threading model used (e.g., "pthreads")
11 **PMIX_MODEL_NUM_THREADS** "pmix.mdl.nthrds" (uint64_t)
12 Number of active threads being used by the model
13 **PMIX_MODEL_NUM_CPUS** "pmix.mdl.ncpu" (uint64_t)
14 Number of cpus being used by the model
15 **PMIX_MODEL_CPU_TYPE** "pmix.mdl.cputype" (char*)
16 Granularity - "hwthread", "core", etc.
17 **PMIX_MODEL_PHASE_NAME** "pmix.mdl.phase" (char*)
18 User-assigned name for a phase in the application execution (e.g., "cfd reduction")
19 **PMIX_MODEL_PHASE_TYPE** "pmix.mdl.ptype" (char*)
20 Type of phase being executed (e.g., "matrix multiply")
21 **PMIX_MODEL_AFFINITY_POLICY** "pmix.mdl.tap" (char*)
22 Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread
23 located on cpu close to master thread), "spread" (threads load-balanced across available cpus)

24 3.4.5 UNIX socket rendezvous socket attributes

25 These attributes are used to describe a UNIX socket for rendezvous with the local RM by passing
26 them into the relevant initialization API - thus, they are not typically accessed via the [PMIx_Get](#)
27 API.

28 **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)
29 Disable legacy UNIX socket (usock) support
30 **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)
31 POSIX *mode_t* (9 bits valid)
32 **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)
33 Use only one rendezvous socket, letting priorities and/or environment parameters select the
34 active transport.

1 3.4.6 TCP connection attributes

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

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

6 If provided, directs that the TCP URI be reported and indicates the desired method of
7 reporting: `'-'` for stdout, `'+'` for stderr, or filename.

8 **PMIX_TCP_URI** `"pmix.tcp.uri"` (`char*`)

9 The URI of the PMIx server to connect to, or a file name containing it in the form of
10 `file:<name of file containing it>`.

11 **PMIX_TCP_IF_INCLUDE** `"pmix.tcp.ifinclude"` (`char*`)

12 Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to
13 include when establishing the TCP connection.

14 **PMIX_TCP_IF_EXCLUDE** `"pmix.tcp.ifexclude"` (`char*`)

15 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
16 TCP connection.

17 **PMIX_TCP_IPV4_PORT** `"pmix.tcp.ipv4"` (`int`)

18 The IPv4 port to be used.

19 **PMIX_TCP_IPV6_PORT** `"pmix.tcp.ipv6"` (`int`)

20 The IPv6 port to be used.

21 **PMIX_TCP_DISABLE_IPV4** `"pmix.tcp.disipv4"` (`bool`)

22 Set to `true` to disable IPv4 family of addresses.

23 **PMIX_TCP_DISABLE_IPV6** `"pmix.tcp.disipv6"` (`bool`)

24 Set to `true` to disable IPv6 family of addresses.

25 3.4.7 Global Data Storage (GDS) attributes

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

29 **PMIX_GDS_MODULE** `"pmix.gds.mod"` (`char*`)

30 Comma-delimited string of desired modules.

31 3.4.8 General process-level attributes

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

33 **PMIX_CPUSSET** `"pmix.cpuset"` (`char*`)

34 `hwloc`² bitmap to be applied to the process upon launch.

35 **PMIX_CREDENTIAL** `"pmix.cred"` (`char*`)

36 Security credential assigned to the process.

37 **PMIX_SPAWNED** `"pmix.spawned"` (`bool`)

²<https://www.open-mpi.org/projects/hwloc/>

1 true if this process resulted from a call to [PMIx_Spawn](#).
2 **PMIX_ARCH** "pmix.arch" (uint32_t)
3 Architecture flag.

4 **3.4.9 Scratch directory attributes**

5 These attributes are used to define an application scratch directory and are referenced using the
6 [PMIX_RANK_WILDCARD](#) rank.

7 **PMIX_TMPDIR** "pmix.tmpdir" (char*)
8 Full path to the top-level temporary directory assigned to the session.
9 **PMIX_NSDIR** "pmix.nmdir" (char*)
10 Full path to the temporary directory assigned to the namespace, under [PMIX_TMPDIR](#).
11 **PMIX_PROCDIR** "pmix.pdir" (char*)
12 Full path to the subdirectory under [PMIX_NSDIR](#) assigned to the process.
13 **PMIX_TDIR_RMCLEAN** "pmix.tdir.rmclean" (bool)
14 Resource Manager will clean session directories

15 **3.4.10 Relative Rank Descriptive Attributes**

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

18 **PMIX_CLUSTER_ID** "pmix.clid" (char*)
19 A string name for the cluster this proc is executing on
20 **PMIX_PROCID** "pmix.procid" (pmix_proc_t)
21 Process identifier
22 **PMIX_NAMESPACE** "pmix.namespace" (char*)
23 Namespace of the job.
24 **PMIX_JOBID** "pmix.jobid" (char*)
25 Job identifier assigned by the scheduler.
26 **PMIX_APPNUM** "pmix.appnum" (uint32_t)
27 Application number within the job.
28 **PMIX_RANK** "pmix.rank" (pmix_rank_t)
29 Process rank within the job.
30 **PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)
31 Process rank spanning across all jobs in this session.
32 **PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)
33 Process rank within this application.
34 **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
35 Starting global rank of this job - referenced using [PMIX_RANK_WILDCARD](#).
36 **PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)
37 Local rank on this node within this job.
38 **PMIX_NODE_RANK** "pmix.nrank" (uint16_t)
39 Process rank on this node spanning all jobs.

```

1  PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)
2      Lowest rank on this node within this job - referenced using PMIX\_RANK\_WILDCARD .
3  PMIX_APPLDR "pmix.aldr" (pmix_rank_t)
4      Lowest rank in this application within this job - referenced using PMIX\_RANK\_WILDCARD .
5  PMIX_PROC_PID "pmix.ppid" (pid_t)
6      PID of specified process.
7  PMIX_SESSION_ID "pmix.session.id" (uint32_t)
8      Session identifier - referenced using PMIX\_RANK\_WILDCARD .
9  PMIX_NODE_LIST "pmix.nlist" (char*)
10     Comma-delimited list of nodes running processes for the specified namespace - referenced
11     using PMIX\_RANK\_WILDCARD .
12  PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)
13     Comma-delimited list of all nodes in this allocation regardless of whether or not they
14     currently host processes - referenced using PMIX\_RANK\_WILDCARD .
15  PMIX_HOSTNAME "pmix.hname" (char*)
16     Name of the host where the specified process is running.
17  PMIX_NODEID "pmix.nodeid" (uint32_t)
18     Node identifier where the specified process is located, expressed as the node's index
19     (beginning at zero) in the array resulting from expansion of the PMIX\_NODE\_MAP regular
20     expression for the job
21  PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
22     Comma-delimited list of ranks on this node within the specified namespace - referenced
23     using PMIX\_RANK\_WILDCARD .
24  PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)
25     Array of pmix\_proc\_t of all processes on the specified node - referenced using
26     PMIX\_RANK\_WILDCARD .
27  PMIX_LOCAL_CPUSSETS "pmix.lcpus" (char*)
28     Colon-delimited cpusets of local peers within the specified namespace - referenced using
29     PMIX\_RANK\_WILDCARD .
30  PMIX_PROC_URI "pmix.puri" (char*)
31     URI containing contact information for a given process.
32  PMIX_LOCALITY "pmix.loc" (uint16_t)
33     Relative locality of the specified process to the requestor.
34  PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
35     Process identifier of the parent process of the calling process.
36  PMIX_EXIT_CODE "pmix.exit.code" (int)
37     Exit code returned when process terminated

```

38 3.4.11 Information retrieval attributes

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

```

41  PMIX_SESSION_INFO "pmix.ssn.info" (bool)

```

1 Return information about the specified session. If information about a session other than the
2 one containing the requesting process is desired, then the attribute array must contain a
3 [PMIX_SESSION_ID](#) attribute identifying the desired target.

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

5 Return information about the specified job or namespace. If information about a job or
6 namespace other than the one containing the requesting process is desired, then the attribute
7 array must contain a [PMIX_JOBID](#) or [PMIX_NAMESPACE](#) attribute identifying the desired
8 target. Similarly, if information is requested about a job or namespace in a session other than
9 the one containing the requesting process, then an attribute identifying the target session
10 must be provided.

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

12 Return information about the specified application. If information about an application other
13 than the one containing the requesting process is desired, then the attribute array must
14 contain a [PMIX_APPNUM](#) attribute identifying the desired target. Similarly, if information
15 is requested about an application in a job or session other than the one containing the
16 requesting process, then attributes identifying the target job and/or session must be provided.

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

18 Return information about the specified node. If information about a node other than the one
19 containing the requesting process is desired, then the attribute array must contain either the
20 [PMIX_NODEID](#) or [PMIX_HOSTNAME](#) attribute identifying the desired target.

21 3.4.12 Information storage attributes

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

24 **PMIX_SESSION_INFO_ARRAY** "pmix.ssn.arr" (pmix_data_array_t)

25 Provide an array of [pmix_info_t](#) containing session-level information. The
26 [PMIX_SESSION_ID](#) attribute is required to be included in the array.

27 **PMIX_JOB_INFO_ARRAY** "pmix.job.arr" (pmix_data_array_t)

28 Provide an array of [pmix_info_t](#) containing job-level information. The
29 [PMIX_SESSION_ID](#) attribute of the [session](#) containing the [job](#) is required to be
30 included in the array whenever the PMIx server library may host multiple sessions (e.g.,
31 when executing with a host RM daemon). As information is registered one job (aka
32 namespace) at a time via the [PMIx_server_register_namespace](#) API, there is no
33 requirement that the array contain either the [PMIX_NAMESPACE](#) or [PMIX_JOBID](#) attributes
34 when used in that context (though either or both of them may be included). At least one of
35 the job identifiers must be provided in all other contexts where the job being referenced is
36 ambiguous.

37 **PMIX_APP_INFO_ARRAY** "pmix.app.arr" (pmix_data_array_t)

38 Provide an array of [pmix_info_t](#) containing app-level information. The [PMIX_NAMESPACE](#)
39 or [PMIX_JOBID](#) attributes of the [job](#) containing the application, plus its [PMIX_APPNUM](#)
40 attribute, are must to be included in the array when the array is *not* included as part of a call
41 to [PMIx_server_register_namespace](#) - i.e., when the job containing the application is
42 ambiguous. The job identification is otherwise optional.

1 **PMIX_NODE_INFO_ARRAY** "pmix.node.arr" (pmix_data_array_t)

2 Provide an array of **pmix_info_t** containing node-level information. At a minimum,
3 either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute is required to be included in the
4 array, though both may be included.

5 Note that these assemblages can be used hierarchically:

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

Advice to PMIx library implementers

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

3.4.13 Size information attributes

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

20 **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)

21 Number of allocated slots in a session - each slot may or may not be occupied by an
22 executing process. Note that this attribute is the equivalent to the combination of
23 **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it is
24 included in the Standard for historical reasons.

25 **PMIX_JOB_SIZE** "pmix.job.size" (uint32_t)

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

31 **PMIX_JOB_NUM_APPS** "pmix.job.napps" (uint32_t)

32 Number of applications in this job.

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

34 Number of processes in this application.

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

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

2 **PMIX_NODE_SIZE** "pmix.node.size" (uint32_t)

3 Number of processes across all jobs on this node.

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

5 Maximum number of processes that can be executed in this context (session, namespace,
6 application, or node). Typically, this is a constraint imposed by a scheduler or by user
7 settings in a hostfile or other resource description.

8 **PMIX_NUM_SLOTS** "pmix.num.slots" (uint32_t)

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

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

13 Number of nodes in this session, or that are currently executing processes from the
14 associated namespace or application.

15 **3.4.14 Memory information attributes**

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

18 **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)

19 Total available physical memory on this node.

20 **PMIX_DAEMON_MEMORY** "pmix.dmn.mem" (float)

21 Megabytes of memory currently used by the RM daemon.

22 **PMIX_CLIENT_AVG_MEMORY** "pmix.cl.mem.avg" (float)

23 Average Megabytes of memory used by client processes.

24 **3.4.15 Topology information attributes**

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

27 **PMIX_NET_TOPO** "pmix.ntopo" (char*)

28 eXtensible Markup Language (XML) representation of the network topology.

29 **PMIX_LOCAL_TOPO** "pmix.ltopo" (char*)

30 XML representation of local node topology.

31 **PMIX_TOPOLOGY** "pmix.topo" (hwloc_topology_t)

32 Pointer to the PMIx client's internal hwloc topology object.

33 **PMIX_TOPOLOGY_XML** "pmix.topo.xml" (char*)

34 XML-based description of topology

35 **PMIX_TOPOLOGY_FILE** "pmix.topo.file" (char*)

36 Full path to file containing XML topology description

37 **PMIX_TOPOLOGY_SIGNATURE** "pmix.toposig" (char*)

38 Topology signature string.

39 **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***)

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

5 **PMIX_COLLECTIVE_ALGO_REQD** "pmix.calreqd" (bool)

6 If true, indicates that the requested choice of algorithm is mandatory.

7 **PMIX_NOTIFY_COMPLETION** "pmix.notecomp" (bool)

8 Notify the parent process upon termination of child job.

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

10 Value for calls to publish/lookup/unpublish or for monitoring event notifications.

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

12 Value for calls to [PMIx_Publish](#).

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

14 Scope of the data to be found in a [PMIx_Get](#) call.

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

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

18 **PMIX_EMBED_BARRIER** "pmix.embed.barrier" (bool)

19 Execute a blocking fence operation before executing the specified operation. For example,
20 [PMIx_Finalize](#) does not include an internal barrier operation by default. This attribute
21 would direct [PMIx_Finalize](#) to execute a barrier as part of the finalize operation.

22 **PMIX_JOB_TERM_STATUS** "pmix.job.term.status" (pmix_status_t)

23 Status to be returned upon job termination.

24 **PMIX_PROC_STATE_STATUS** "pmix.proc.state" (pmix_proc_state_t)

25 Process state

26 3.4.17 Server-to-PMIx library attributes

27 Attributes used by the host environment to pass data to its PMIx server library. The data will then
28 be parsed and provided to the local PMIx clients. These attributes are all referenced using
29 [PMIX_RANK_WILDCARD](#) except where noted.

30 **PMIX_REGISTER_NODATA** "pmix.reg.nodata" (bool)

31 Registration is for this namespace only, do not copy job data - this attribute is not accessed
32 using the [PMIx_Get](#)

33 **PMIX_PROC_DATA** "pmix.pdata" (pmix_data_array_t)

34 Array of process data. Starts with rank, then contains more data.

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

36 Regular expression of nodes - see [11.1.3.1](#) for an explanation of its generation.

37 **PMIX_PROC_MAP** "pmix.pmap" (char*)

38 Regular expression describing processes on each node - see [11.1.3.1](#) for an explanation of its
39 generation.

40 **PMIX_ANL_MAP** "pmix.anlmap" (char*)

41 Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.

1 **PMIX_APP_MAP_TYPE** "pmix.apmap.type" (char*)
2 Type of mapping used to layout the application (e.g., **cyclic**).
3 **PMIX_APP_MAP_REGEX** "pmix.apmap.regex" (char*)
4 Regular expression describing the result of the process mapping.

5 **3.4.18 Server-to-Client attributes**

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

8 **PMIX_PROC_BLOB** "pmix.pblob" (pmix_byte_object_t)
9 Packed blob of process data.
10 **PMIX_MAP_BLOB** "pmix.mblob" (pmix_byte_object_t)
11 Packed blob of process location.

12 **3.4.19 Event handler registration and notification attributes**

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

15 **PMIX_EVENT_HDLR_NAME** "pmix.evname" (char*)
16 String name identifying this handler.
17 **PMIX_EVENT_HDLR_FIRST** "pmix.evfirst" (bool)
18 Invoke this event handler before any other handlers.
19 **PMIX_EVENT_HDLR_LAST** "pmix.evlast" (bool)
20 Invoke this event handler after all other handlers have been called.
21 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY** "pmix.evfirstcat" (bool)
22 Invoke this event handler before any other handlers in this category.
23 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY** "pmix.evlastcat" (bool)
24 Invoke this event handler after all other handlers in this category have been called.
25 **PMIX_EVENT_HDLR_BEFORE** "pmix.evbefore" (char*)
26 Put this event handler immediately before the one specified in the (char*) value.
27 **PMIX_EVENT_HDLR_AFTER** "pmix.evafter" (char*)
28 Put this event handler immediately after the one specified in the (char*) value.
29 **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (bool)
30 Prepend this handler to the precedence list within its category.
31 **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (bool)
32 Append this handler to the precedence list within its category.
33 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrang" (pmix_data_array_t*)
34 Array of **pmix_proc_t** defining range of event notification.
35 **PMIX_EVENT_AFFECTED_PROC** "pmix.evproc" (pmix_proc_t)
36 The single process that was affected.
37 **PMIX_EVENT_AFFECTED_PROCS** "pmix.evaffected" (pmix_data_array_t*)
38 Array of **pmix_proc_t** defining affected processes.
39 **PMIX_EVENT_NON_DEFAULT** "pmix.evnond" (bool)

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

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

3 Object to be returned whenever the registered callback function **cbfunc** is invoked. The
4 object will only be returned to the process that registered it.

5 **PMIX_EVENT_DO_NOT_CACHE** "pmix.evnocache" (bool)

6 Instruct the PMIx server not to cache the event.

7 **PMIX_EVENT_SILENT_TERMINATION** "pmix.evsilentterm" (bool)

8 Do not generate an event when this job normally terminates.

9 **PMIX_EVENT_PROXY** "pmix.evproxy" (pmix_proc_t*)

10 PMIx server that sourced the event

11 **PMIX_EVENT_TEXT_MESSAGE** "pmix.evtext" (char*)

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

13 3.4.20 Fault tolerance attributes

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

16 **PMIX_EVENT_TERMINATE_SESSION** "pmix.evterm.sess" (bool)

17 The RM intends to terminate this session.

18 **PMIX_EVENT_TERMINATE_JOB** "pmix.evterm.job" (bool)

19 The RM intends to terminate this job.

20 **PMIX_EVENT_TERMINATE_NODE** "pmix.evterm.node" (bool)

21 The RM intends to terminate all processes on this node.

22 **PMIX_EVENT_TERMINATE_PROC** "pmix.evterm.proc" (bool)

23 The RM intends to terminate just this process.

24 **PMIX_EVENT_ACTION_TIMEOUT** "pmix.evtimeout" (int)

25 The time in seconds before the RM will execute error response.

26 **PMIX_EVENT_NO_TERMINATION** "pmix.evnoterm" (bool)

27 Indicates that the handler has satisfactorily handled the event and believes termination of the
28 application is not required.

29 **PMIX_EVENT_WANT_TERMINATION** "pmix.evterm" (bool)

30 Indicates that the handler has determined that the application should be terminated

31 3.4.21 Spawn attributes

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

39 **PMIX_PERSONALITY** "pmix.pers" (char*)

1 Name of personality to use.

2 **PMIX_HOST** "pmix.host" (char*)

3 Comma-delimited list of hosts to use for spawned processes.

4 **PMIX_HOSTFILE** "pmix.hostfile" (char*)

5 Hostfile to use for spawned processes.

6 **PMIX_ADD_HOST** "pmix.addhost" (char*)

7 Comma-delimited list of hosts to add to the allocation.

8 **PMIX_ADD_HOSTFILE** "pmix.addhostfile" (char*)

9 Hostfile listing hosts to add to existing allocation.

10 **PMIX_PREFIX** "pmix.prefix" (char*)

11 Prefix to use for starting spawned processes.

12 **PMIX_WDIR** "pmix.wdir" (char*)

13 Working directory for spawned processes.

14 **PMIX_MAPPER** "pmix.mapper" (char*)

15 Mapping mechanism to use for placing spawned processes - when accessed using

16 **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping

17 mechanism used for the provided namespace.

18 **PMIX_DISPLAY_MAP** "pmix.dispmap" (bool)

19 Display process mapping upon spawn.

20 **PMIX_PPR** "pmix.ppr" (char*)

21 Number of processes to spawn on each identified resource.

22 **PMIX_MAPBY** "pmix.mapby" (char*)

23 Process mapping policy - when accessed using **PMIx_Get** , use the

24 **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the

25 provided namespace

26 **PMIX_RANKBY** "pmix.rankby" (char*)

27 Process ranking policy - when accessed using **PMIx_Get** , use the

28 **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the

29 provided namespace

30 **PMIX_BINDTO** "pmix.bindto" (char*)

31 Process binding policy - when accessed using **PMIx_Get** , use the

32 **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the

33 provided namespace

34 **PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)

35 Preload binaries onto nodes.

36 **PMIX_PRELOAD_FILES** "pmix.preloadfiles" (char*)

37 Comma-delimited list of files to pre-position on nodes.

38 **PMIX_NON_PMI** "pmix.nonpmi" (bool)

39 Spawned processes will not call **PMIx_Init** .

40 **PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)

41 Spawned process rank that is to receive **stdin**.

42 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)

43 Forward this process's **stdin** to the designated process.

1 **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)
 2 Forward **stdout** from spawned processes to this process.

3 **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)
 4 Forward **stderr** from spawned processes to this process.

5 **PMIX_FWD_STDDIAG** "pmix.fwd.stddiag" (bool)
 6 If a diagnostic channel exists, forward any output on it from the spawned processes to this
 7 process (typically used by a tool)

8 **PMIX_DEBUGGER_DAEMONS** "pmix.debugger" (bool)
 9 Spawned application consists of debugger daemons.

10 **PMIX_COSPAWN_APP** "pmix.cospawn" (bool)
 11 Designated application is to be spawned as a disconnected job. Meaning that it is not part of
 12 the "comm_world" of the parent process.

13 **PMIX_SET_SESSION_CWD** "pmix.ssn cwd" (bool)
 14 Set the application's current working directory to the session working directory assigned by
 15 the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for
 16 the rank to discover the session working directory assigned to the provided namespace

17 **PMIX_TAG_OUTPUT** "pmix.tagout" (bool)
 18 Tag application output with the identity of the source process.

19 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
 20 Timestamp output from applications.

21 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
 22 Merge **stdout** and **stderr** streams from application processes.

23 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)
 24 Output application output to the specified file.

25 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)
 26 Mark the **argv** with the rank of the process.

27 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)
 28 Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the
 29 **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the
 30 provided namespace

31 **PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)
 32 Do not place processes on the head node.

33 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)
 34 Do not oversubscribe the cpus.

35 **PMIX_REPORT_BINDINGS** "pmix.repbinding" (bool)
 36 Report bindings of the individual processes.

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

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

43 **PMIX_JOB_CONTINUOUS** "pmix.continuous" (bool)

1 Application is continuous, all failed processes should be immediately restarted.
2 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)
3 Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the
4 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
5 namespace
6 **PMIX_SPAWN_TOOL** "pmix.spwn.tool" (bool)
7 Indicate that the job being spawned is a tool

8 3.4.22 Query attributes

9 Attributes used to describe **PMIx_Query_info_nb** behavior - these are values passed to the
10 **PMIx_Query_info_nb** API and therefore are not passed to the **PMIx_Get** API.
11 **PMIX_QUERY_REFRESH_CACHE** "pmix.qry.rfsh" (bool)
12 Retrieve updated information from server.
13 **PMIX_QUERY_NAMESPACES** "pmix.qry.ns" (char*)
14 Request a comma-delimited list of active namespaces.
15 **PMIX_QUERY_JOB_STATUS** "pmix.qry.jst" (pmix_status_t)
16 Status of a specified, currently executing job.
17 **PMIX_QUERY_QUEUE_LIST** "pmix.qry.qlst" (char*)
18 Request a comma-delimited list of scheduler queues.
19 **PMIX_QUERY_QUEUE_STATUS** "pmix.qry.qst" (TBD)
20 Status of a specified scheduler queue.
21 **PMIX_QUERY_PROC_TABLE** "pmix.qry.ptable" (char*)
22 Input namespace of the job whose information is being requested returns (
23 **pmix_data_array_t**) an array of **pmix_proc_info_t** .
24 **PMIX_QUERY_LOCAL_PROC_TABLE** "pmix.qry.lptable" (char*)
25 Input namespace of the job whose information is being requested returns (
26 **pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same
27 node.
28 **PMIX_QUERY_AUTHORIZATIONS** "pmix.qry.auths" (bool)
29 Return operations the PMIx tool is authorized to perform.
30 **PMIX_QUERY_SPAWN_SUPPORT** "pmix.qry.spawn" (bool)
31 Return a comma-delimited list of supported spawn attributes.
32 **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
33 Return a comma-delimited list of supported debug attributes.
34 **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)
35 Return information on memory usage for the processes indicated in the qualifiers.
36 **PMIX_QUERY_LOCAL_ONLY** "pmix.qry.local" (bool)
37 Constrain the query to local information only.
38 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
39 Report only average values for sampled information.
40 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)
41 Report minimum and maximum values.

1 **PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (char*)
 2 String identifier of the allocation whose status is being requested.
 3 **PMIX_TIME_REMAINING** "pmix.time.remaining" (char*)
 4 Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
 5 **PMIX_QUERY_ATTRIBUTE_SUPPORT** "pmix.qry.attrs" (bool)
 6 Query list of supported attributes for specified APIs
 7 **PMIX_QUERY_NUM_PSETS** "pmix.qry.psetnum" (size_t)
 8 Return the number of psets defined in the specified range (defaults to session).
 9 **PMIX_QUERY_PSET_NAMES** "pmix.qry.psets" (char*)
 10 Return a comma-delimited list of the names of the psets defined in the specified range
 11 (defaults to session).

12 3.4.23 Log attributes

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

15 **PMIX_LOG_SOURCE** "pmix.log.source" (pmix_proc_t*)
 16 ID of source of the log request
 17 **PMIX_LOG_STDERR** "pmix.log.stderr" (char*)
 18 Log string to stderr.
 19 **PMIX_LOG_STDOUT** "pmix.log.stdout" (char*)
 20 Log string to stdout.
 21 **PMIX_LOG_SYSLOG** "pmix.log.syslog" (char*)
 22 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,
 23 otherwise to local syslog
 24 **PMIX_LOG_LOCAL_SYSLOG** "pmix.log.lsys" (char*)
 25 Log data to local syslog. Defaults to **ERROR** priority.
 26 **PMIX_LOG_GLOBAL_SYSLOG** "pmix.log.gsys" (char*)
 27 Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.
 28 **PMIX_LOG_SYSLOG_PRI** "pmix.log.syspri" (int)
 29 Syslog priority level
 30 **PMIX_LOG_TIMESTAMP** "pmix.log.tstmp" (time_t)
 31 Timestamp for log report
 32 **PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstmp" (bool)
 33 Generate timestamp for log
 34 **PMIX_LOG_TAG_OUTPUT** "pmix.log.tag" (bool)
 35 Label the output stream with the channel name (e.g., "stdout")
 36 **PMIX_LOG_TIMESTAMP_OUTPUT** "pmix.log.tsout" (bool)
 37 Print timestamp in output string
 38 **PMIX_LOG_XML_OUTPUT** "pmix.log.xml" (bool)
 39 Print the output stream in XML format
 40 **PMIX_LOG_ONCE** "pmix.log.once" (bool)

1 Only log this once with whichever channel can first support it, taking the channels in priority
2 order

3 **PMIX_LOG_MSG** "pmix.log.msg" (**pmix_byte_object_t**)
4 Message blob to be sent somewhere.
5 **PMIX_LOG_EMAIL** "pmix.log.email" (**pmix_data_array_t**)
6 Log via email based on [pmix_info_t](#) containing directives.
7 **PMIX_LOG_EMAIL_ADDR** "pmix.log.emaddr" (**char***)
8 Comma-delimited list of email addresses that are to receive the message.
9 **PMIX_LOG_EMAIL_SENDER_ADDR** "pmix.log.emfaddr" (**char***)
10 Return email address of sender
11 **PMIX_LOG_EMAIL_SUBJECT** "pmix.log.emsub" (**char***)
12 Subject line for email.
13 **PMIX_LOG_EMAIL_MSG** "pmix.log.emmsg" (**char***)
14 Message to be included in email.
15 **PMIX_LOG_EMAIL_SERVER** "pmix.log.esrvr" (**char***)
16 Hostname (or IP address) of estmp server
17 **PMIX_LOG_EMAIL_SRVR_PORT** "pmix.log.esrvrprt" (**int32_t**)
18 Port the email server is listening to
19 **PMIX_LOG_GLOBAL_DATASTORE** "pmix.log.gstore" (**bool**)
20 Store the log data in a global data store (e.g., database)
21 **PMIX_LOG_JOB_RECORD** "pmix.log.jrec" (**bool**)
22 Log the provided information to the host environment's job record

23 3.4.24 Debugger attributes

24 Attributes used to assist debuggers - these are values that can be passed to the [PMIx_Spawn](#) or
25 [PMIx_Init](#) APIs. Some may be accessed using the [PMIx_Get](#) API with the
26 [PMIX_RANK_WILDCARD](#) rank.

27 **PMIX_DEBUG_STOP_ON_EXEC** "pmix.dbg.exec" (**bool**)
28 Passed to [PMIx_Spawn](#) to indicate that the specified application is being spawned under
29 debugger, and that the launcher is to pause the resulting application processes on first
30 instruction for debugger attach.
31 **PMIX_DEBUG_STOP_IN_INIT** "pmix.dbg.init" (**bool**)
32 Passed to [PMIx_Spawn](#) to indicate that the specified application is being spawned under
33 debugger, and that the PMIx client library is to pause the resulting application processes
34 during [PMIx_Init](#) until debugger attach and release.
35 **PMIX_DEBUG_WAIT_FOR_NOTIFY** "pmix.dbg.notify" (**bool**)
36 Passed to [PMIx_Spawn](#) to indicate that the specified application is being spawned under
37 debugger, and that the resulting application processes are to pause at some
38 application-determined location until debugger attach and release.
39 **PMIX_DEBUG_JOB** "pmix.dbg.job" (**char***)
40 Namespace of the job to be debugged - provided to the debugger upon launch.
41 **PMIX_DEBUG_WAITING_FOR_NOTIFY** "pmix.dbg.waiting" (**bool**)

1 Job to be debugged is waiting for a release - this is not a value accessed using the
2 **PMIx_Get** API.
3 **PMIX_DEBUG_JOB_DIRECTIVES** "pmix.dbg.jdirs" (pmix_data_array_t*)
4 Array of job-level directives
5 **PMIX_DEBUG_APP_DIRECTIVES** "pmix.dbg.adirs" (pmix_data_array_t*)
6 Array of app-level directives

7 **3.4.25 Resource manager attributes**

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

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

16 **3.4.26 Environment variable attributes**

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

19 **PMIX_SET_ENVAR** "pmix.envar.set" (pmix_envar_t*)
20 Set the envar to the given value, overwriting any pre-existing one
21 **PMIX_UNSET_ENVAR** "pmix.envar.unset" (char*)
22 Unset the environment variable specified in the string.
23 **PMIX_ADD_ENVAR** "pmix.envar.add" (pmix_envar_t*)
24 Add the environment variable, but do not overwrite any pre-existing one
25 **PMIX_PREPEND_ENVAR** "pmix.envar.prepnd" (pmix_envar_t*)
26 Prepend the given value to the specified environmental value using the given separator
27 character, creating the variable if it doesn't already exist
28 **PMIX_APPEND_ENVAR** "pmix.envar.appnd" (pmix_envar_t*)
29 Append the given value to the specified environmental value using the given separator
30 character, creating the variable if it doesn't already exist

31 **3.4.27 Job Allocation attributes**

32 Attributes used to describe the job allocation - these are values passed to the
33 **PMIx_Allocation_request_nb** API and are not accessed using the **PMIx_Get** API

34 **PMIX_ALLOC_ID** "pmix.alloc.id" (char*)
35 Provide a string identifier for this allocation request which can later be used to query status
36 of the request.
37 **PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (uint64_t)

1 The number of nodes.

2 **PMIX_ALLOC_NODE_LIST** "pmix.alloc.nlist" (char*)

3 Regular expression of the specific nodes.

4 **PMIX_ALLOC_NUM_CPUS** "pmix.alloc.ncpus" (uint64_t)

5 Number of cpus.

6 **PMIX_ALLOC_NUM_CPU_LIST** "pmix.alloc.ncpulist" (char*)

7 Regular expression of the number of cpus for each node.

8 **PMIX_ALLOC_CPU_LIST** "pmix.alloc.cpulist" (char*)

9 Regular expression of the specific cpus indicating the cpus involved.

10 **PMIX_ALLOC_MEM_SIZE** "pmix.alloc.msize" (float)

11 Number of Megabytes.

12 **PMIX_ALLOC_NETWORK** "pmix.alloc.net" (array)

13 Array of [pmix_info_t](#) describing requested network resources. This must include at
14 least: [PMIX_ALLOC_NETWORK_ID](#), [PMIX_ALLOC_NETWORK_TYPE](#), and
15 [PMIX_ALLOC_NETWORK_ENDPTS](#), plus whatever other descriptors are desired.

16 **PMIX_ALLOC_NETWORK_ID** "pmix.alloc.netid" (char*)

17 The key to be used when accessing this requested network allocation. The allocation will be
18 returned/stored as a [pmix_data_array_t](#) of [pmix_info_t](#) indexed by this key and
19 containing at least one entry with the same key and the allocated resource description. The
20 type of the included value depends upon the network support. For example, a TCP allocation
21 might consist of a comma-delimited string of socket ranges such as
22 "32000-32100,33005,38123-38146". Additional entries will consist of any provided
23 resource request directives, along with their assigned values. Examples include:
24 [PMIX_ALLOC_NETWORK_TYPE](#) - the type of resources provided;
25 [PMIX_ALLOC_NETWORK_PLANE](#) - if applicable, what plane the resources were assigned
26 from; [PMIX_ALLOC_NETWORK_QOS](#) - the assigned QoS; [PMIX_ALLOC_BANDWIDTH](#) -
27 the allocated bandwidth; [PMIX_ALLOC_NETWORK_SEC_KEY](#) - a security key for the
28 requested network allocation. NOTE: the assigned values may differ from those requested,
29 especially if [PMIX_INFO_REQD](#) was not set in the request.

30 **PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (float)

31 Mbits/sec.

32 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)

33 Quality of service level.

34 **PMIX_ALLOC_TIME** "pmix.alloc.time" (uint32_t)

35 Time in seconds.

36 **PMIX_ALLOC_NETWORK_TYPE** "pmix.alloc.nettype" (char*)

37 Type of desired transport (e.g., "tcp", "udp")

38 **PMIX_ALLOC_NETWORK_PLANE** "pmix.alloc.netplane" (char*)

39 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

40 **PMIX_ALLOC_NETWORK_ENDPTS** "pmix.alloc.endpts" (size_t)

41 Number of endpoints to allocate per process

42 **PMIX_ALLOC_NETWORK_ENDPTS_NODE** "pmix.alloc.endpts.nd" (size_t)

43 Number of endpoints to allocate per node

1 **PMIX_ALLOC_NETWORK_SEC_KEY** "pmix.alloc.nsec" (pmix_byte_object_t)
2 Network security key

3 **3.4.28 Job control attributes**

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

6 **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
7 Provide a string identifier for this request.
8 **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
9 Pause the specified processes.
10 **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
11 Resume ("un-pause") the specified processes.
12 **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
13 Cancel the specified request (**NULL** implies cancel all requests from this requestor).
14 **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)
15 Forcibly terminate the specified processes and cleanup.
16 **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)
17 Restart the specified processes using the given checkpoint ID.
18 **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)
19 Checkpoint the specified processes and assign the given ID to it.
20 **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)
21 Use event notification to trigger a process checkpoint.
22 **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)
23 Use the given signal to trigger a process checkpoint.
24 **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)
25 Time in seconds to wait for a checkpoint to complete.
26 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**
27 "pmix.jctrl.ckmethod" (pmix_data_array_t)
28 Array of **pmix_info_t** declaring each method and value supported by this application.
29 **PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)
30 Send given signal to specified processes.
31 **PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)
32 Regular expression identifying nodes that are to be provisioned.
33 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvning" (char*)
34 Name of the image that is to be provisioned.
35 **PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)
36 Indicate that the job can be pre-empted.
37 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)
38 Politely terminate the specified processes.
39 **PMIX_REGISTER_CLEANUP** "pmix.reg.cleanup" (char*)
40 Comma-delimited list of files to be removed upon process termination
41 **PMIX_REGISTER_CLEANUP_DIR** "pmix.reg.cleanupdir" (char*)

1 Comma-delimited list of directories to be removed upon process termination
2 **PMIX_CLEANUP_RECURSIVE** "pmix.clnup.recurse" (bool)
3 Recursively cleanup all subdirectories under the specified one(s)
4 **PMIX_CLEANUP_EMPTY** "pmix.clnup.empty" (bool)
5 Only remove empty subdirectories
6 **PMIX_CLEANUP_IGNORE** "pmix.clnup.ignore" (char*)
7 Comma-delimited list of filenames that are not to be removed
8 **PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)
9 When recursively cleaning subdirectories, do not remove the top-level directory (the one
10 given in the cleanup request)

11 3.4.29 Monitoring attributes

12 Attributes used to control monitoring of an executing application- these are values passed to the
13 [PMIx_Process_monitor_nb](#) API and are not accessed using the [PMIx_Get](#) API.

14 **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
15 Provide a string identifier for this request.
16 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
17 Identifier to be canceled (NULL means cancel all monitoring for this process).
18 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
19 The application desires to control the response to a monitoring event.
20 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)
21 Register to have the PMIx server monitor the requestor for heartbeats.
22 **PMIX_SEND_HEARTBEAT** "pmix.monitor.beat" (void)
23 Send heartbeat to local PMIx server.
24 **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)
25 Time in seconds before declaring heartbeat missed.
26 **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)
27 Number of heartbeats that can be missed before generating the event.
28 **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)
29 Register to monitor file for signs of life.
30 **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)
31 Monitor size of given file is growing to determine if the application is running.
32 **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)
33 Monitor time since last access of given file to determine if the application is running.
34 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)
35 Monitor time since last modified of given file to determine if the application is running.
36 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)
37 Time in seconds between checking the file.
38 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)
39 Number of file checks that can be missed before generating the event.

1 3.4.30 Security attributes

2 *PMIx v3.0* Attributes for managing security credentials

3 **PMIX_CRED_TYPE** "pmix.sec.ctype" (char*)

4 When passed in **PMIx_Get_credential**, a prioritized, comma-delimited list of desired
5 credential types for use in environments where multiple authentication mechanisms may be
6 available. When returned in a callback function, a string identifier of the credential type.

7 **PMIX_CRYPT_KEY** "pmix.sec.key" (pmix_byte_object_t)

8 Blob containing crypto key

9 3.4.31 IO Forwarding attributes

10 *PMIx v3.0* Attributes used to control IO forwarding behavior

11 **PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (uint32_t)

12 The requested size of the server cache in bytes for each specified channel. By default, the
13 server is allowed (but not required) to drop all bytes received beyond the max size.

14 **PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (bool)

15 In an overflow situation, drop the oldest bytes to make room in the cache.

16 **PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (bool)

17 In an overflow situation, drop any new bytes received until room becomes available in the
18 cache (default).

19 **PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (uint32_t)

20 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
21 IO arrives. The library will execute the callback whenever the specified number of bytes
22 becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
23 respective channel.

24 **PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (uint32_t)

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

27 **PMIX_IOF_COMPLETE** "pmix.iof.cmp" (bool)

28 Indicates whether or not the specified IO channel has been closed by the source.

29 **PMIX_IOF_TAG_OUTPUT** "pmix.iof.tag" (bool)

30 Tag output with the channel it comes from.

31 **PMIX_IOF_TIMESTAMP_OUTPUT** "pmix.iof.ts" (bool)

32 Timestamp output

33 **PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)

34 Format output in XML

1 3.4.32 Application setup attributes

2 *PMIx v3.0* Attributes for controlling contents of application setup data

3 **PMIX_SETUP_APP_ENVARS** "pmix.setup.env" (bool)
4 Harvest and include relevant environmental variables
5 **PMIX_SETUP_APP_NONENVARS** "pmix.setup.nenv" (bool)
6 Include all relevant data other than environmental variables
7 **PMIX_SETUP_APP_ALL** "pmix.setup.all" (bool)
8 Include all relevant data

9 3.4.33 Attribute support level attributes

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

26 3.4.34 Descriptive attributes

27 **PMIX_MAX_VALUE** "pmix.descr.maxval" (varies)
28 Used in `pmix_regattr_t` to describe the maximum valid value for the associated
29 attribute.
30 **PMIX_MIN_VALUE** "pmix.descr.minval" (varies)
31 Used in `pmix_regattr_t` to describe the minimum valid value for the associated
32 attribute.
33 **PMIX_ENUM_VALUE** "pmix.descr.enum" (char*)
34 Used in `pmix_regattr_t` to describe accepted values for the associated attribute.
35 Numerical values shall be presented in a form convertible to the attribute's declared data
36 type. Named values (i.e., values defined by constant names via a typical C-language enum
37 declaration) must be provided as their numerical equivalent.

1 3.4.35 Process group attributes

2 *PMIx v4.0* Attributes for controlling the PMIx Group APIs

3 **PMIX_GROUP_ID** "pmix.grp.id" (char*)

4 User-provided group identifier

5 **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)

6 This process is the leader of the group

7 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

8 Participation is optional - do not return an error if any of the specified processes terminate
9 without having joined. The default is false

10 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

11 Notify remaining members when another member terminates without first leaving the group.
12 The default is false

13 **PMIX_GROUP_INVITE_DECLINE** "pmix.grp.decline" (bool)

14 Notify the inviting process that this process does not wish to participate in the proposed
15 group The default is true

16 **PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)

17 Adjust internal tracking for terminated processes. Default is false

18 **PMIX_GROUP_MEMBERSHIP** "pmix.grp.mbrs" (pmix_data_array_t*)

19 Array of group member ID's

20 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

21 Notify remaining members when another member terminates without first leaving the group.
22 The default is false

23 **PMIX_GROUP_CONTEXT_ID** "pmix.grp.ctxid" (size_t)

24 Context ID assigned to group

25 **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)

26 Group operation only involves local processes. PMIx implementations are *required* to
27 automatically scan an array of group members for local vs remote processes - if only local
28 processes are detected, the implementation need not execute a global collective for the
29 operation unless a context ID has been requested from the host environment. This can result
30 in significant time savings. This attribute can be used to optimize the operation by indicating
31 whether or not only local processes are represented, thus allowing the implementation to
32 bypass the scan. The default is false

33 **PMIX_GROUP_ENDPT_DATA** "pmix.grp.endpt" (pmix_byte_object_t)

34 Data collected to be shared during group construction

35 3.5 Callback Functions

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

1 3.5.1 Release Callback Function

2 Summary

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

6 Format

PMIx v1.0

C

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

C

9 INOUT `cbdata`

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

11 Description

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

14 3.5.2 Modex Callback Function

15 Summary

16 The `pmix_modex_cbfunc_t` is used by the `pmix_server_fencefn_t` and
17 `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex business card
18 exchange (BCX) data.

PMIx v1.0

C

```
19 typedef void (*pmix_modex_cbfunc_t)  
20     (pmix_status_t status,  
21      const char *data, size_t ndata,  
22      void *cbdata,  
23      pmix_release_cbfunc_t release_fn,  
24      void *release_cbdata)
```

C

25 IN `status`

26 Status associated with the operation (handle)

27 IN `data`

28 Data to be passed (pointer)

29 IN `ndata`

30 size of the data (`size_t`)

31 IN `cbdata`

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

1 **IN** `release_fn`
2 Callback for releasing *data* (function pointer)
3 **IN** `release_cbdata`
4 Pointer to be passed to *release_fn* (memory reference)

5 **Description**

6 A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data
7 in response to “fence” and “get” operations. The returned blob contains the data collected from
8 each server participating in the operation.

9 **3.5.3 Spawn Callback Function**

10 **Summary**

11 The `pmix_spawn_cbfunc_t` is used on the PMIx client side by `PMIx_Spawn_nb` and on
12 the PMIx server side by `pmix_server_spawn_fn_t`.

PMIx v1.0

```
13 typedef void (*pmix_spawn_cbfunc_t)  
14     (pmix_status_t status,  
15     pmix_namespace_t nspc, void *cbdata);
```

16 **IN** `status`
17 Status associated with the operation (handle)
18 **IN** `nspc`
19 Namespace string (`pmix_namespace_t`)
20 **IN** `cbdata`
21 Callback data passed to original API call (memory reference)

22 **Description**

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

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

1 3.5.4 Op Callback Function

2 Summary

3 The [pmix_op_cbfunc_t](#) is used by operations that simply return a status.

PMIx v1.0

```
▼ C ▶  
4 typedef void (*pmix_op_cbfunc_t)  
5     (pmix_status_t status, void *cbdata);  
▲ C ▶
```

6 **IN status**

7 Status associated with the operation (handle)

8 **IN cbdata**

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

10 Description

11 Used by a wide range of PMIx API's including [PMIx_Fence_nb](#),
12 [pmix_server_client_connected_fn_t](#), [PMIx_server_register_namespace](#). This
13 callback function is used to return a status to an often nonblocking operation.

14 3.5.5 Lookup Callback Function

15 Summary

16 The [pmix_lookup_cbfunc_t](#) is used by [PMIx_Lookup_nb](#) to return data.

PMIx v1.0

```
▼ C ▶  
17 typedef void (*pmix_lookup_cbfunc_t)  
18     (pmix_status_t status,  
19      pmix_pdata_t data[], size_t ndata,  
20      void *cbdata);  
▲ C ▶
```

21 **IN status**

22 Status associated with the operation (handle)

23 **IN data**

24 Array of data returned ([pmix_pdata_t](#))

25 **IN ndata**

26 Number of elements in the *data* array (**size_t**)

27 **IN cbdata**

28 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

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

IN status

Status associated with the operation (handle)

IN kv

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

IN cbdata

Callback data passed to original API call (memory reference)

Description

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

3.5.7 Info Callback Function

Summary

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

PMIx v2.0

```
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

1 **IN status**
2 Status associated with the operation (`pmix_status_t`)
3 **IN info**
4 Array of `pmix_info_t` returned by the operation (pointer)
5 **IN ninfo**
6 Number of elements in the *info* array (`size_t`)
7 **IN cbdata**
8 Callback data passed to original API call (memory reference)
9 **IN release_fn**
10 Function to be called when done with the *info* data (function pointer)
11 **IN release_cbdata**
12 Callback data to be passed to *release_fn* (memory reference)

Description

13 The *status* indicates if requested data was found or not. An array of `pmix_info_t` will contain
14 the key/value pairs.
15

16 3.5.8 Event Handler Registration Callback Function

17 The `pmix_evhdlr_reg_cbfunc_t` callback function.

Advice to users

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

PMIx v2.0

C

```
22 typedef void (*pmix_evhdlr_reg_cbfunc_t)  
23     (pmix_status_t status,  
24      size_t evhdlr_ref,  
25      void *cbdata)
```

C

26 **IN status**
27 Status indicates if the request was successful or not (`pmix_status_t`)
28 **IN evhdlr_ref**
29 Reference assigned to the event handler by PMIx — this reference * must be used to
30 deregister the err handler (`size_t`)
31 **IN cbdata**
32 Callback data passed to original API call (memory reference)

1 **Description**

2 Define a callback function for calls to [PMIx_Register_event_handler](#)

3 **3.5.9 Notification Handler Completion Callback Function**

4 **Summary**

5 The [pmix_event_notification_cbfnc_fn_t](#) is called by event handlers to indicate
6 completion of their operations.

PMIx v2.0

C

```

7       typedef void (*pmix_event_notification_cbfnc_fn_t)
8                    (pmix_status_t status,
9                    pmix_info_t *results, size_t nresults,
10                    pmix_op_cbfnc_t cbfunc, void *thiscbdata,
11                    void *notification_cbdata);

```

C

- 12 **IN status**
13 Status returned by the event handler’s operation ([pmix_status_t](#))
- 14 **IN results**
15 Results from this event handler’s operation on the event ([pmix_info_t](#))
- 16 **IN nresults**
17 Number of elements in the results array ([size_t](#))
- 18 **IN cbfunc**
19 [pmix_op_cbfnc_t](#) function to be executed when PMIx completes processing the
20 callback (function reference)
- 21 **IN thiscbdata**
22 Callback data that was passed in to the handler (memory reference)
- 23 **IN cbdata**
24 Callback data to be returned when PMIx executes cbfunc (memory reference)

25 **Description**

26 Define a callback by which an event handler can notify the PMIx library that it has completed its
27 response to the notification. The handler is *required* to execute this callback so the library can
28 determine if additional handlers need to be called. The handler shall return
29 [PMIX_ERR_EVENT_COMPLETE](#) if no further action is required. The return status of each event
30 handler and any returned [pmix_info_t](#) structures will be added to the *results* array of
31 [pmix_info_t](#) passed to any subsequent event handlers to help guide their operation.

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

1 3.5.10 Notification Function

2 Summary

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

Advice to users

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

PMIx v2.0

```
7 typedef void (*pmix_notification_fn_t)
8     (size_t evhdlr_registration_id,
9      pmix_status_t status,
10     const pmix_proc_t *source,
11     pmix_info_t info[], size_t ninfo,
12     pmix_info_t results[], size_t nresults,
13     pmix_event_notification_cbfnc_fn_t cbfunc,
14     void *cbdata);
```

15 **IN** `evhdlr_registration_id`

Registration number of the handler being called (`size_t`)

17 **IN** `status`

Status associated with the operation (`pmix_status_t`)

19 **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`

22 **IN** `info`

Information describing the event (`pmix_info_t`). This argument will be `NULL` if no additional information was provided by the event generator.

25 **IN** `ninfo`

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

27 **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.

31 **IN** `nresults`

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

33 **IN** `cbfunc`

`pmix_event_notification_cbfnc_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.

1
PMIx v2.0

Format

C

```
2 typedef void (*pmix_setup_application_cbfunc_t) (  
3     pmix_status_t status,  
4     pmix_info_t info[], size_t ninfo,  
5     void *provided_cbdata,  
6     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

- 7 **IN status**
8 returned status of the request ([pmix_status_t](#))
- 9 **IN info**
10 Array of info structures (array of handles)
- 11 **IN ninfo**
12 Number of elements in the *info* array (integer)
- 13 **IN provided_cbdata**
14 Data originally passed to call to [PMIx_server_setup_application](#) (memory
15 reference)
- 16 **IN cbfunc**
17 [pmix_op_cbfunc_t](#) function to be called when processing completed (function reference)
- 18 **IN cbdata**
19 Data to be passed to the *cbfunc* callback function (memory reference)

Description

21 Define a function to be called by the PMIx server library for return of application-specific setup
22 data in response to a request from the host RM. The returned *info* array is owned by the PMIx
23 server library and will be free'd when the provided *cbfunc* is called.

24 3.5.12 Server Direct Modex Response Callback Function

25 The [PMIx_server_dmodex_request](#) callback function.

26 Summary

27 Provide a function by which the local PMIx server library can return connection and other data
28 posted by local application processes to the host resource manager.

29 Format

PMIx v1.0

C

```
30 typedef void (*pmix_dmodex_response_fn_t) (pmix_status_t status,  
31     char *data, size_t sz,  
32     void *cbdata);
```

C

1 **IN status**
2 Returned status of the request (`pmix_status_t`)
3 **IN data**
4 Pointer to a data "blob" containing the requested information (handle)
5 **IN sz**
6 Number of bytes in the *data* blob (integer)
7 **IN cbdata**
8 Data passed into the initial call to `PMIx_server_dmodex_request` (memory reference)

Description

9
10 Define a function to be called by the PMIx server library for return of information posted by a local
11 application process (via `PMIx_Put` with subsequent `PMIx_Commit`) in response to a request
12 from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd
13 upon return from the function.

14 3.5.13 PMIx Client Connection Callback Function

15 Summary

16 Callback function for incoming connection request from a local client

17 Format

PMIx v1.0

C

```
18 typedef void (*pmix_connection_cbfunc_t) (  
19                                     int incoming_sd, void *cbdata)
```

C

20 **IN incoming_sd**
21 (integer)
22 **IN cbdata**
23 (memory reference)

24 Description

25 Callback function for incoming connection requests from local clients - only used by host
26 environments that wish to directly handle socket connection requests.

27 3.5.14 PMIx Tool Connection Callback Function

28 Summary

29 Callback function for incoming tool connections.

1
PMIx v2.0

Format

C

```
2 typedef void (*pmix_tool_connection_cbfunc_t) (  
3             pmix_status_t status,  
4             pmix_proc_t *proc, void *cbdata)
```

C

- 5 **IN** **status**
- 6 **pmix_status_t** value (handle)
- 7 **IN** **proc**
- 8 **pmix_proc_t** structure containing the identifier assigned to the tool (handle)
- 9 **IN** **cbdata**
- 10 Data to be passed (memory reference)

Description

11 Callback function for incoming tool connections. The host environment shall provide a
12 namespace/rank identifier for the connecting tool.
13

Advice to PMIx server hosts

14 It is assumed that **rank=0** will be the normal assignment, but allow for the future possibility of a
15 parallel set of tools connecting, and thus each process requiring a unique rank.

16 3.5.15 Credential callback function

17 Summary

18 Callback function to return a requested security credential

1
PMIx v3.0

Format

C

```
2 typedef void (*pmix_credential_cbfunc_t) (  
3     pmix_status_t status,  
4     pmix_byte_object_t *credential,  
5     pmix_info_t info[], size_t ninfo,  
6     void *cbdata)
```

C

- 7 **IN status**
8 `pmix_status_t` value (handle)
- 9 **IN credential**
10 `pmix_byte_object_t` structure containing the security credential (handle)
- 11 **IN info**
12 Array of provided by the system to pass any additional information about the credential - e.g.,
13 the identity of the issuing agent. (handle)
- 14 **IN ninfo**
15 Number of elements in *info* (`size_t`)
- 16 **IN cbdata**
17 Object passed in original request (memory reference)

Description

19 Define a callback function to return a requested security credential. Information provided by the
20 issuing agent can subsequently be used by the application for a variety of purposes. Examples
21 include:

- 22 • checking identified authorizations to determine what requests/operations are feasible as a means
23 to steering workflows
- 24 • compare the credential type to that of the local SMS for compatibility

Advice to users

25 The credential is opaque and therefore understandable only by a service compatible with the issuer.
26 The *info* array is owned by the PMIx library and is not to be released or altered by the receiving
27 party.

28 3.5.16 Credential validation callback function

29 Summary

30 Callback function for security credential validation

1
PMIx v3.0

Format

C

```

2 typedef void (*pmix_validation_cbfunc_t) (
3         pmix_status_t status,
4         pmix_info_t info[], size_t ninfo,
5         void *cbdata);

```

C

- 6 **IN status**
7 `pmix_status_t` value (handle)
- 8 **IN info**
9 Array of `pmix_info_t` provided by the system to pass any additional information about
10 the authentication - e.g., the effective userid and group id of the certificate holder, and any
11 related authorizations (handle)
- 12 **IN ninfo**
13 Number of elements in *info* (`size_t`)
- 14 **IN cbdata**
15 Object passed in original request (memory reference)

Description

16 Define a validation callback function to indicate if a provided credential is valid, and any
17 corresponding information regarding authorizations and other security matters.
18

Advice to users

19 The precise contents of the array will depend on the host environment and its associated security
20 system. At the minimum, it is expected (but not required) that the array will contain entries for the
21 `PMIX_USERID` and `PMIX_GRPID` of the client described in the credential. The *info* array is
22 owned by the PMIx library and is not to be released or altered by the receiving party.

23 3.5.17 IOF delivery function

24 Summary

25 Callback function for delivering forwarded IO to a process

1
PMIx v3.0

Format

C

```

2 typedef void (*pmix_iof_cbfunc_t) (
3     size_t iofhdlr, pmix_iof_channel_t channel,
4     pmix_proc_t *source, char *payload,
5     pmix_info_t info[], size_t ninfo);

```

C

- 6 **IN iofhdlr**
Registration number of the handler being invoked (**size_t**)
- 7
- 8 **IN channel**
bitmask identifying the channel the data arrived on (**pmix_iof_channel_t**)
- 9
- 10 **IN source**
Pointer to a **pmix_proc_t** identifying the namespace/rank of the process that generated the data (**char***)
- 11
- 12
- 13 **IN payload**
Pointer to character array containing the data.
- 14
- 15 **IN info**
Array of **pmix_info_t** provided by the source containing metadata about the payload.
This could include **PMIX_IOF_COMPLETE** (handle)
- 16
- 17
- 18 **IN ninfo**
Number of elements in *info* (**size_t**)
- 19

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.

26 3.5.18 IOF and Event registration function

27 Summary

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

1 **Format**

PMIx v3.0

```

2 typedef void (*pmix_hdlr_reg_cbfunc_t) (pmix_status_t status,
3                                         size_t refid,
4                                         void *cbdata);

```

5 **IN status**

6 **PMIX_SUCCESS** or an appropriate error constant (`pmix_status_t`)

7 **IN refid**

8 reference identifier assigned to the handler by PMIx, used to deregister the handler (`size_t`)

9 **IN cbdata**

10 object provided to the registration call (pointer)

11 **Description**

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

13 3.6 Constant String Functions

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

16 **Summary**

17 String representation of a `pmix_status_t`.

PMIx v1.0

```

18 const char*
19 PMIx_Error_string (pmix_status_t status);

```

20 **Summary**

21 String representation of a `pmix_proc_state_t`.

PMIx v2.0

```

22 const char*
23 PMIx_Proc_state_string (pmix_proc_state_t state);

```


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

PMIx v2.0 ▼  ▼

3 **const char***
4 **PMIx_Scope_string**(pmix_scope_t scope);

▲  ▲

5 **Summary**
6 String representation of a [pmix_persistence_t](#) .

PMIx v2.0 ▼  ▼

7 **const char***
8 **PMIx_Persistence_string**(pmix_persistence_t persist);

▲  ▲

9 **Summary**
10 String representation of a [pmix_data_range_t](#) .

PMIx v2.0 ▼  ▼

11 **const char***
12 **PMIx_Data_range_string**(pmix_data_range_t range);

▲  ▲

13 **Summary**
14 String representation of a [pmix_info_directives_t](#) .

PMIx v2.0 ▼  ▼

15 **const char***
16 **PMIx_Info_directives_string**(pmix_info_directives_t directives);

▲  ▲

17 **Summary**
18 String representation of a [pmix_data_type_t](#) .

PMIx v2.0 ▼  ▼

19 **const char***
20 **PMIx_Data_type_string**(pmix_data_type_t type);

▲  ▲

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

PMIx v2.0 ▼ C ▼

3 `const char*`
4 `PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);`
▲ C ▲

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

PMIx v3.0 ▼ C ▼

7 `const char*`
8 `PMIx_IOF_channel_string(pmix_iof_channel_t channel);`
▲ C ▲

CHAPTER 4

Initialization and Finalization

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

4 There are three sets of initialization and finalization functions depending upon the role of the
5 process in the PMIx universe. Each of these functional sets are described in this chapter. Note that
6 a process can only call *one* of the init/finalize functional pairs - e.g., a process that calls the client
7 initialization function cannot also call the tool or server initialization functions, and must call the
8 corresponding client finalize.

Advice to users

9 Processes that initialize as a server or tool automatically are given access to all client APIs. Server
10 initialization includes setting up the infrastructure to support local clients - thus, it necessarily
11 includes overhead and an increased memory footprint. Tool initialization automatically searches for
12 a server to which it can connect — if declared as a *launcher*, the PMIx library sets up the required
13 “hooks” for other tools (e.g., debuggers) to attach to it.

4.1 Query

15 The API defined in this section can be used by any PMIx process, regardless of their role in the
16 PMIx universe.

4.1.1 PMIx_Initialized

Format

PMIx v1.0

```
int PMIx_Initialized(void)
```

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





Rationale

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

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




4 **4.1.2 PMIx_Get_version**

5 **Summary**
6 Get the PMIx version information.

7 **Format**
8 *PMIx v1.0*  
9 `const char* PMIx_Get_version(void)`
10  

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

12 **4.2 Client Initialization and Finalization**

13 Initialization and finalization routines for PMIx clients.
14  **Advice to users** 
15 The PMIx *ad hoc* v1.0 Standard defined the `PMIx_Init` function, but modified the function
16 signature in the v1.2 version. The *ad hoc* v1.0 version is not included in this document to avoid
confusion.


17 **4.2.1 PMIx_Init**

18 **Summary**
19 Initialize the PMIx client library

1
PMIx v1.2

Format

```

pmix_status_t
PMIx_Init (pmix_proc_t *proc,
           pmix_info_t info[], size_t ninfo)

```

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.

1 **PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)
 2 The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be
 3 supported for specifying the port to be used.

4 **PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)
 5 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be
 6 supported for specifying the port to be used.

7 **PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)
 8 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,
 9 this attribute may be supported for disabling it.

10 **PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)
 11 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,
 12 this attribute may be supported for disabling it.

13 **PMIX_EVENT_BASE** "pmix.evbase" (struct event_base *)
 14 Pointer to libevent¹ **event_base** to use in place of the internal progress thread.

15 **PMIX_GDS_MODULE** "pmix.gds.mod" (char*)
 16 Comma-delimited string of desired modules. This attribute is specific to the PRI and
 17 controls only the selection of GDS module for internal use by the process. Module selection
 18 for interacting with the server is performed dynamically during the connection process.



19 Description

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

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

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

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

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

¹<http://libevent.org/>

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

3 4.2.2 `PMIx_Finalize`

4 Summary

5 Finalize the PMIx client library.

6 Format

PMIx v1.0

```
▼ _____ C _____ ▼  
7 pmix_status_t  
8 PMIx_Finalize(const pmix_info_t info[], size_t ninfo)  
▲ _____ C _____ ▲
```

9 **IN** `info`

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

11 **IN** `ninfo`

12 Number of element in the `info` array (`size_t`)

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

▼ ----- Optional Attributes ----- ▼

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

15 **PMIX_EMBED_BARRIER** "`pmix.embed.barrier`" (`bool`)

16 Execute a blocking fence operation before executing the specified operation. For example,
17 `PMIx_Finalize` does not include an internal barrier operation by default. This attribute
18 would direct `PMIx_Finalize` to execute a barrier as part of the finalize operation.

▲ ----- ▲

19 Description

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

23 4.3 Tool Initialization and Finalization

24 Initialization and finalization routines for PMIx tools.

25 4.3.1 `PMIx_tool_init`

26 Summary

27 Initialize the PMIx library for operating as a tool.

1
PMIx v2.0

Format

```

pmix_status_t
PMIx_tool_init(pmix_proc_t *proc,
               pmix_info_t info[], size_t ninfo)

```

INOUT proc

`pmix_proc_t` structure (handle)

IN info

Array of `pmix_info_t` structures (array of handles)

IN ninfo

Number of 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_NAMESPACE "pmix.tool.namespace" (`char*`)

Name of the namespace to use for this tool.

PMIX_TOOL_RANK "pmix.tool.rank" (`uint32_t`)

Rank of this tool.

PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (`bool`)

The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

PMIX_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`)

1 Time in seconds between connection attempts to a PMIx server.

2 **PMIX_CONNECT_MAX_RETRIES** "pmix.tool.mretries" (uint32_t)

3 Maximum number of times to try to connect to PMIx server.

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

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

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

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

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

12 Comma-delimited list of devices and/or CIDR notation to include when establishing the
13 TCP connection. If the library supports TCP socket connections, this attribute may be
14 supported for specifying the interfaces to be used.

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

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

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

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

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

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

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

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

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

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

31 **PMIX_EVENT_BASE** "pmix.evbase" (struct event_base *)

32 Pointer to libevent² **event_base** to use in place of the internal progress thread.

33 **PMIX_GDS_MODULE** "pmix.gds.mod" (char*)

34 Comma-delimited string of desired modules. This attribute is specific to the PRI and
35 controls only the selection of GDS module for internal use by the process. Module selection
36 for interacting with the server is performed dynamically during the connection process.

²<http://libevent.org/>



1 Description

2 Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided
3 `pmix_proc_t` struct. The *info* array is used to pass user requests pertaining to the init and
4 subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives are
5 desired.

6 If called with the **PMIX_TOOL_DO_NOT_CONNECT** attribute, the PMIx tool library will fully
7 initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later
8 point in time, if desired. In all other cases, the PMIx tool library will attempt to connect to
9 according to the following precedence chain:

- 10 • if **PMIX_SERVER_URI** or **PMIX_TCP_URI** is given, then connection will be attempted to the
11 server at the specified URI. Note that it is an error for both of these attributes to be specified.
12 **PMIX_SERVER_URI** is the preferred method as it is more generalized — **PMIX_TCP_URI** is
13 provided for those cases where the user specifically wants to use a TCP transport for the
14 connection and wants to error out if it isn't available or cannot succeed. The PMIx library will
15 return an error if connection fails — it will not proceed to check for other connection options as
16 the user specified a particular one to use
- 17 • if **PMIX_SERVER_PIDINFO** was provided, then the tool will search under the directory
18 provided by the `PMIX_SERVER_TMPDIR` environmental variable for a rendezvous file created
19 by the process corresponding to that PID. The PMIx library will return an error if the rendezvous
20 file cannot be found, or the connection is refused by the server
- 21 • if **PMIX_CONNECT_TO_SYSTEM** is given, then the tool will search for a system-level
22 rendezvous file created by a PMIx server in the directory specified by the
23 `PMIX_SYSTEM_TMPDIR` environmental variable. If found, then the tool will attempt to
24 connect to it. An error is returned if the rendezvous file cannot be found or the connection is
25 refused.
- 26 • if **PMIX_CONNECT_SYSTEM_FIRST** is given, then the tool will search for a system-level
27 rendezvous file created by a PMIx server in the directory specified by the
28 `PMIX_SYSTEM_TMPDIR` environmental variable. If found, then the tool will attempt to
29 connect to it. In this case, no error will be returned if the rendezvous file is not found or
30 connection is refused — the PMIx library will silently continue to the next option
- 31 • by default, the tool will search the directory tree under the directory provided by the
32 `PMIX_SERVER_TMPDIR` environmental variable for rendezvous files of PMIx servers,
33 attempting to connect to each it finds until one accepts the connection. If no rendezvous files are
34 found, or all contacted servers refuse connection, then the PMIx library will return an error.

35 If successful, the function will return **PMIX_SUCCESS** and will fill the provided structure (if
36 provided) with the server-assigned namespace and rank of the tool. Note that each connection
37 attempt in the above precedence chain will retry (with delay between each retry) a number of times
38 according to the values of the corresponding attributes. Default is no retries.

1 Note that the PMIx tool library is referenced counted, and so multiple calls to `PMIx_tool_init`
2 are allowed. Thus, one way to obtain the namespace and rank of the process is to simply call
3 `PMIx_tool_init` with a non-NULL parameter.

4 4.3.2 `PMIx_tool_finalize`

5 **Summary**

6 Finalize the PMIx library for a tool connection.

7 **Format**

PMIx v2.0

▼ `PMIx_tool_finalize` C

8 `pmix_status_t`

9 `PMIx_tool_finalize(void)`

▲ `PMIx_tool_finalize` C ▲

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

11 **Description**

12 Finalize the PMIx tool library, closing the connection to the server. An error code will be returned
13 if, for some reason, the connection cannot be cleanly terminated — in this case, the connection is
14 dropped.

15 4.3.3 `PMIx_tool_connect_to_server`

16 **Summary**

17 Switch connection from the current PMIx server to another one, or initialize a connection to a
18 specified server.

19 **Format**

PMIx v3.0

▼ `PMIx_tool_connect_to_server` C

20 `pmix_status_t`

21 `PMIx_tool_connect_to_server(pmix_proc_t *proc,`
22 `pmix_info_t info[], size_t ninfo)`

▲ `PMIx_tool_connect_to_server` C ▲

23 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_NAMESPACE "pmix.srv.namespace" (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 namespace/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.

1 4.4 Server Initialization and Finalization

2 Initialization and finalization routines for PMIx servers.

3 4.4.1 PMIx_server_init

4 Summary

5 Initialize the PMIx server.

6 Format

PMIx v1.0

```
▼----- C -----▼  
7 pmix_status_t  
8 PMIx_server_init(pmix_server_module_t *module,  
9                 pmix_info_t info[], size_t ninfo)  
▲----- C -----▲
```

10 INOUT module

11 `pmix_server_module_t` structure (handle)

12 IN info

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

14 IN ninfo

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

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

▼----- Required Attributes -----▼

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

18 **PMIX_SERVER_NAMESPACE** "pmix.srv.namespace" (`char*`)

19 Name of the namespace to use for this PMIx server.

20 **PMIX_SERVER_RANK** "pmix.srv.rank" (`pmix_rank_t`)

21 Rank of this PMIx server

22 **PMIX_SERVER_TMPDIR** "pmix.srv.tmpdir" (`char*`)

23 Top-level temporary directory for all client processes connected to this server, and where the
24 PMIx server will place its tool rendezvous point and contact information.

25 **PMIX_SYSTEM_TMPDIR** "pmix.sys.tmpdir" (`char*`)

26 Temporary directory for this system, and where a PMIx server that declares itself to be a
27 system-level server will place a tool rendezvous point and contact information.

28 **PMIX_SERVER_TOOL_SUPPORT** "pmix.srvr.tool" (`bool`)

29 The host RM wants to declare itself as willing to accept tool connection requests.

30 **PMIX_SERVER_SYSTEM_SUPPORT** "pmix.srvr.sys" (`bool`)

31 The host RM wants to declare itself as being the local system server for PMIx connection
32 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.

1 **PMIX_EVENT_BASE** "pmix.evbase" (struct event_base *)
 2 Pointer to libevent³ event_base to use in place of the internal progress thread.
 3 **PMIX_GDS_MODULE** "pmix.gds.mod" (char*)
 4 Comma-delimited string of desired modules. This attribute is specific to the PRI and
 5 controls only the selection of GDS module for internal use by the process. Module selection
 6 for interacting with the server is performed dynamically during the connection process.



7 **Description**

8 Initialize the PMIx server support library, and provide a pointer to a pmix_server_module_t
 9 structure containing the caller's callback functions. The array of pmix_info_t structs is used to
 10 pass additional info that may be required by the server when initializing. For example, it may
 11 include the PMIX_SERVER_TOOL_SUPPORT key, thereby indicating that the daemon is willing
 12 to accept connection requests from tools.

▼ **Advice to PMIx server hosts** ▼

13 Providing a value of NULL for the module argument is permitted, as is passing an empty module
 14 structure. Doing so indicates that the host environment will not provide support for multi-node
 15 operations such as PMIx_Fence, but does intend to support local clients access to information.



16 **4.4.2 PMIx_server_finalize**

17 **Summary**

18 Finalize the PMIx server library.

19 **Format**

PMIx v1.0

▼ C ▼
 20 pmix_status_t
 21 PMIx_server_finalize(void)
 ▲ C ▲

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

23 **Description**

24 Finalize the PMIx server support library, terminating all connections to attached tools and any local
 25 clients. All memory usage is released.

³<http://libevent.org/>

CHAPTER 5

Key/Value Management

1 Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of
2 the PMIx community is to eliminate collective operations, it is recognized that the traditional
3 method of publishing/exchanging data must be supported until that objective can be met. This
4 method relies on processes to discover and publish their local information which is collected by the
5 local PMIx server library. Global exchange of the published information is then executed via a
6 collective operation performed by the host SMS servers.

7 Keys are required to be unique within a specific level of information as defined in 3.4.11. For
8 example, a value for `PMIX_NUM_NODES` can be specified for each of the `session`, `job`, and
9 `application` levels. However, subsequently specifying another value for that attribute in the
10 `session` level will overwrite the prior value.

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

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

IN scope

Distribution scope of the provided value (handle)

IN key

key (`pmix_key_t`)

IN value

Reference to a `pmix_value_t` structure (handle)

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

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 29. 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.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
4         const pmix_info_t info[], size_t ninfo,
5         pmix_value_t **val)
```

C

6 **IN** **proc**
7 process reference (handle)
8 **IN** **key**
9 key to retrieve ([pmix_key_t](#))
10 **IN** **info**
11 Array of info structures (array of handles)
12 **IN** **ninfo**
13 Number of element in the *info* array (integer)
14 **OUT** **val**
15 value (handle)

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

Required Attributes

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

- 18 **PMIX_OPTIONAL** "pmix.optional" (bool)
19 Look only in the client's local data store for the requested value - do not request data from
20 the PMIx server if not found.
- 21 **PMIX_IMMEDIATE** "pmix.immediate" (bool)
22 Specified operation should immediately return an error from the PMIx server if the requested
23 data cannot be found - do not request it from the host RM.
- 24 **PMIX_DATA_SCOPE** "pmix.scope" ([pmix_scope_t](#))
25 Scope of the data to be found in a [PMIx_Get](#) call.
- 26 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)
27 Return information about the specified session. If information about a session other than the
28 one containing the requesting process is desired, then the attribute array must contain a
29 [PMIX_SESSION_ID](#) attribute identifying the desired target.
- 30 **PMIX_JOB_INFO** "pmix.job.info" (bool)

1 Return information about the specified job or namespace. If information about a job or
2 namespace other than the one containing the requesting process is desired, then the attribute
3 array must contain a `PMIX_JOBID` or `PMIX_NAMESPACE` attribute identifying the desired
4 target. Similarly, if information is requested about a job or namespace in a session other than
5 the one containing the requesting process, then an attribute identifying the target session
6 must be provided.

7 **PMIX_APP_INFO** "`pmix.app.info`" (bool)

8 Return information about the specified application. If information about an application other
9 than the one containing the requesting process is desired, then the attribute array must
10 contain a `PMIX_APPNUM` attribute identifying the desired target. Similarly, if information is
11 requested about an application in a job or session other than the one containing the requesting
12 process, then attributes identifying the target job and/or session must be provided.

13 **PMIX_NODE_INFO** "`pmix.node.info`" (bool)

14 Return information about the specified node. If information about a node other than the one
15 containing the requesting process is desired, then the attribute array must contain either the
16 `PMIX_NODEID` or `PMIX_HOSTNAME` attribute identifying the desired target.



Optional Attributes



17 The following attributes are optional for host environments:

18 **PMIX_TIMEOUT** "`pmix.timeout`" (int)

19 Time in seconds before the specified operation should time out (0 indicating infinite) in
20 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
21 the target process from ever exposing its data.



Advice to PMIx library implementers



22 We recommend that implementation of the `PMIX_TIMEOUT` attribute be left to the host
23 environment due to race condition considerations between delivery of the data by the host
24 environment versus internal timeout in the PMIx server library. Implementers that choose to
25 support `PMIX_TIMEOUT` directly in the PMIx server library must take care to resolve the race
26 condition and should avoid passing `PMIX_TIMEOUT` to the host environment so that multiple
27 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.

1
PMIx v1.0

Format

C

```

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

```

C

- 6 **IN proc**
process reference (handle)
- 7
- 8 **IN key**
key to retrieve (string)
- 9
- 10 **IN info**
Array of info structures (array of handles)
- 11
- 12 **IN ninfo**
Number of elements in the *info* array (integer)
- 13
- 14 **IN cbfunc**
Callback function (function reference)
- 15
- 16 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 17

18 Returns one of the following:

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

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

- 28 • **PMIX_SUCCESS** The requested data has been returned
- 29 • **PMIX_ERR_NOT_FOUND** The requested data was not available
- 30 • a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

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

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

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

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

6 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)
7 Return information about the specified session. If information about a session other than the
8 one containing the requesting process is desired, then the attribute array must contain a
9 **PMIX_SESSION_ID** attribute identifying the desired target.

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

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

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

▲-----▲

▼-----▼ **Optional Attributes** -----▼

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

28 **PMIX_TIMEOUT** "pmix.timeout" (int)
29 Time in seconds before the specified operation should time out (0 indicating infinite) in
30 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
31 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.

1
PMIx v1.0

Format

C

```
2 pmix_status_t  
3 PMIx_Store_internal(const pmix_proc_t *proc,  
4                     const pmix_key_t key,  
5                     pmix_value_t *val);
```

C

```
6 IN  proc  
7     process reference (handle)  
8 IN  key  
9     key to retrieve (string)  
10 IN  val  
11     Value to store (handle)
```

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

Description

13 Store some data locally for retrieval by other areas of the proc. This is data that has only internal
14 scope - it will never be “pushed” externally.
15

16 5.1.5 Accessing information: examples

17 This section provides examples illustrating methods for accessing information at various levels.
18 The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate
19 how **PMIx_Get** can be used to obtain information on a **session**, **job**, **application**,
20 process, and node.

21 5.1.5.1 Session-level information

22 The **PMIx_Get** API does not include an argument for specifying the **session** associated with
23 the information being requested. Information regarding the session containing the requestor can be
24 obtained by the following methods:

- 25 • for session-level attributes (e.g., **PMIX_UNIV_SIZE**), specifying the requestor’s namespace
26 and a rank of **PMIX_RANK_WILDCARD**; or
- 27 • for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_SESSION_INFO**
28 attribute to indicate that the session-level information for that attribute is being requested

29 Example requests are shown below:

C

```

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

```

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

```

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

```

C

1 5.1.5.2 Job-level information

2 Information regarding a job can be obtained by the following methods:

- 3 • for job-level attributes (e.g., `PMIX_JOB_SIZE` or `PMIX_JOB_NUM_APPS`), specifying the
4 namespace of the job and a rank of `PMIX_RANK_WILDCARD` for the *proc* argument to
5 `PMIx_Get` ; or
- 6 • for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_JOB_INFO`
7 attribute to indicate that the job-level information for that attribute is being requested

8 Example requests are shown below:

```
9 pmix_info_t info;  
10 pmix_value_t *value;  
11 pmix_status_t rc;  
12 pmix_proc_t myproc, wildcard;  
13  
14 /* initialize the client library */  
15 PMIx_Init(&myproc, NULL, 0);  
16  
17 /* get the #apps in our job */  
18 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);  
19 rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);  
20  
21 /* get the #nodes in our job */  
22 PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);  
23 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

24 5.1.5.3 Application-level information

25 Information regarding an application can be obtained by the following methods:

- 26 • for application-level attributes (e.g., `PMIX_APP_SIZE`), specifying the namespace and rank of
27 a process within that application;
- 28 • for application-level attributes (e.g., `PMIX_APP_SIZE`), including the `PMIX_APPNUM`
29 attribute specifying the application whose information is being requested. In this case, the
30 namespace field of the *proc* argument is used to reference the *job* containing the application -
31 the *rank* field is ignored;
- 32 • or application-level attributes (e.g., `PMIX_APP_SIZE`), including the `PMIX_APPNUM` and
33 `PMIX_NAMESPACE` or `PMIX_JOBID` attributes specifying the job/application whose information
34 is being requested. In this case, the *proc* argument is ignored;
- 35 • for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_APP_INFO`
36 attribute to indicate that the application-level information for that attribute is being requested

1 Example requests are shown below:

```
2 pmix_info_t info;
3 pmix_value_t *value;
4 pmix_status_t rc;
5 pmix_proc_t myproc, otherproc;
6 uint32_t appsize, appnum;
7
8 /* initialize the client library */
9 PMIx_Init(&myproc, NULL, 0);
10
11 /* get the #processes in our application */
12 rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
13 appsize = value->data.uint32;
14
15 /* get the #nodes in an application containing "otherproc".
16  * Note that the rank of a process in the other application
17  * must be obtained first - a simple method is shown here */
18
19 /* assume for this example that we are in the first application
20  * and we want the #nodes in the second application - use the
21  * rank of the first process in that application, remembering
22  * that ranks start at zero */
23 PMIX_PROC_LOAD(&otherproc, myproc.namespace, appsize);
24
25 PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
26 rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);
27
28 /* alternatively, we can directly ask for the #nodes in
29  * the second application in our job, again remembering that
30  * application numbers start with zero */
31 appnum = 1;
32 PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
33 PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
34 rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
35
```

36 5.1.5.4 Process-level information

37 Process-level information is accessed by providing the namespace and rank of the target process. In
38 the absence of any directive as to the level of information being requested, the PMIx library will
39 always return the process-level value.

1 5.1.5.5 Node-level information

2 Information regarding a node within the system can be obtained by the following methods:

- 3 • for node-level attributes (e.g., `PMIX_NODE_SIZE`), specifying the namespace and rank of a
4 process executing on the target node;
- 5 • for node-level attributes (e.g., `PMIX_NODE_SIZE`), including the `PMIX_NODEID` or
6 `PMIX_HOSTNAME` attribute specifying the node whose information is being requested. In this
7 case, the *proc* argument's values are ignored; or
- 8 • for non-specific attributes (e.g., `PMIX_MAX_PROCS`), including the `PMIX_NODE_INFO`
9 attribute to indicate that the node-level information for that attribute is being requested

10 Example requests are shown below:

```
11 pmix_info_t info[2];  
12 pmix_value_t *value;  
13 pmix_status_t rc;  
14 pmix_proc_t myproc, otherproc;  
15 uint32_t nodeid;  
16  
17 /* initialize the client library */  
18 PMIx_Init(&myproc, NULL, 0);  
19  
20 /* get the #procs on our node */  
21 rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);  
22  
23 /* get the #slots on another node */  
24 PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);  
25 PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);  
26 rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);  
27
```

28 An explanation of the use of `PMIx_Get` versus `PMIx_Query_info_nb` is provided in 7.1.3.1.

29 5.2 Exchanging Key/Value Pairs

30 The APIs defined in this section push key/value pairs from the client to the local PMIx server, and
31 circulate the data between PMIx servers for subsequent retrieval by the local clients.

1 5.2.1 `PMIx_Commit`

2 **Summary**

3 Push all previously `PMIx_Put` values to the local PMIx server.

4 **Format**

PMIx v1.0

C

5 `pmix_status_t PMIx_Commit(void)`

C

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

7 **Description**

8 This is an asynchronous operation. The PRI will immediately return to the caller while the data is
9 transmitted to the local server in the background.

Advice to users

10 The local PMIx server will cache the information locally - i.e., the committed data will not be
11 circulated during `PMIx_Commit`. Availability of the data upon completion of `PMIx_Commit` is
12 therefore implementation-dependent.

13 5.2.2 `PMIx_Fence`

14 **Summary**

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

1
PMIx v1.0

Format

C

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

C

- 5 **IN procs**
Array of `pmix_proc_t` structures (array of handles)
- 7 **IN nprocs**
Number of element in the `procs` array (integer)
- 9 **IN info**
Array of info structures (array of handles)
- 11 **IN ninfo**
Number of element in the `info` array (integer)

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

Required Attributes

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

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

Optional Attributes

17 The following attributes are optional for host environments:

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

22 **PMIX_COLLECTIVE_ALGO** "pmix.calgo" (char*)
23 Comma-delimited list of algorithms to use for the collective operation. PMIx does not
24 impose any requirements on a host environment's collective algorithms. Thus, the
25 acceptable values for this attribute will be environment-dependent - users are encouraged to
26 check their host environment for supported values.

27 **PMIX_COLLECTIVE_ALGO_REQD** "pmix.calreqd" (bool)
28 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.

1
PMIx v1.0

Format

C

```

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

```

C

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

18 Returns one of the following:

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

Required Attributes

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

- 28 **PMIX_COLLECT_DATA** "pmix.collect" (bool)
29 Collect 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.

1 5.3 Publish and Lookup Data

2 The APIs defined in this section publish data from one client that can be later exchanged and looked
3 up by another client.

▼ Advice to PMIx library implementers ▼

4 PMIx libraries that support any of the functions in this section are required to support *all* of them.

▲

▼ Advice to PMIx server hosts ▼

5 Host environments that support any of the functions in this section are required to support *all* of
6 them.

▲

7 5.3.1 PMIx_Publish

8 Summary

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

10 Format

PMIx v1.0

▼ C ▼

11 `pmix_status_t`

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

▲ C ▲

13 **IN** `info`

14 Array of info structures (array of handles)

15 **IN** `ninfo`

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

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

▼ Required Attributes ▼

18 PMIx libraries are not required to directly support any attributes for this function. However, any
19 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
20 *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process that
21 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_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.

1 5.3.2 PMIx_Publish_nb

2 Summary

3 Nonblocking [PMIx_Publish](#) routine.

4 Format

PMIx v1.0

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

8 IN info

9 Array of info structures (array of handles)

10 IN ninfo

11 Number of element in the *info* array (integer)

12 IN cbfunc

13 Callback function [pmix_op_cbfunc_t](#) (function reference)

14 IN cbdata

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

16 Returns one of the following:

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

Required Attributes

24 PMIx libraries are not required to directly support any attributes for this function. However, any
25 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
26 *required* to add the [PMIX_USERID](#) and the [PMIX_GRPID](#) attributes of the client process that
27 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**.

1
PMIx v1.0

Format

```

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

```

INOUT data

Array of publishable data structures (array of 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_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`.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Lookup_nb(char **keys,
4                 const pmix_info_t info[], size_t ninfo,
5                 pmix_lookup_cbfunc_t cbfunc, void *cbdata)
```

C

- 6 **IN keys**
Array to be provided to the callback (array of strings)
- 8 **IN info**
Array of info structures (array of handles)
- 10 **IN ninfo**
Number of element in the *info* array (integer)
- 12 **IN cbfunc**
Callback function (handle)
- 14 **IN cbdata**
Callback data to be provided to the callback function (pointer)

16 Returns one of the following:

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

Required Attributes

21 PMIx libraries are not required to directly support any attributes for this function. However, any
22 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
23 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is
24 requesting the info.

Optional Attributes

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

- 26 **PMIX_TIMEOUT** "pmix.timeout" (int)
27 Time in seconds before the specified operation should time out (0 indicating infinite) in
28 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
29 the target process from ever exposing its data.
- 30 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)
31 Value for calls to publish/lookup/unpublish or for monitoring event notifications.
- 32 **PMIX_WAIT** "pmix.wait" (int)

1 Caller requests that the PMIx server wait until at least the specified number of values are
2 found (0 indicates all and is the default).

Advice to PMIx library implementers

3 We recommend that implementation of the `PMIX_TIMEOUT` attribute be left to the host
4 environment due to race condition considerations between completion of the operation versus
5 internal timeout in the PMIx server library. Implementers that choose to support `PMIX_TIMEOUT`
6 directly in the PMIx server library must take care to resolve the race condition and should avoid
7 passing `PMIX_TIMEOUT` to the host environment so that multiple competing timeouts are not
8 created.

Description

9 Non-blocking form of the `PMIx_Lookup` function. Data for the provided NULL-terminated *keys*
10 array will be returned in the provided callback function. As with `PMIx_Lookup`, the default
11 behavior is to not wait for data to be published. The *info* array can be used to modify the behavior
12 as previously described by `PMIx_Lookup`. Both the *info* and *keys* arrays must be maintained until
13 the callback is provided.
14

5.3.5 PMIx_Unpublish

Summary

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

Format

17 *PMIx v1.0*

```
18 pmix_status_t  
19 PMIx_Unpublish(char **keys,  
20                 const pmix_info_t info[], size_t ninfo)
```

IN info

22 Array of info structures (array of handles)

IN ninfo

23 Number of element in the *info* array (integer)

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

Required Attributes

25 PMIx libraries are not required to directly support any attributes for this function. However, any
26 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
27 *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process that is
28 requesting the operation.
29
30

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_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**.

1
PMIx v1.0

Format

C

```

2 pmix_status_t
3 PMIx_Unpublish_nb(char **keys,
4                   const pmix_info_t info[], size_t ninfo,
5                   pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN keys**
(array of strings)
- 8 **IN info**
Array of info structures (array of handles)
- 10 **IN ninfo**
Number of element in the *info* array (integer)
- 12 **IN cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)
- 14 **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

1 This chapter defines functionality used by clients to create and destroy/abort processes in the PMIx
2 universe.

3 6.1 Abort

4 PMIx provides a dedicated API by which an application can request that specified processes be
5 aborted by the system.

6 6.1.1 PMIx_Abort

7 Summary

8 Abort the specified processes

9 Format

PMIx v1.0

C

```
10 pmix_status_t  
11 PMIx_Abort(int status, const char msg[],  
12           pmix_proc_t procs[], size_t nprocs)
```

C

13 **IN** **status**

14 Error code to return to invoking environment (integer)

15 **IN** **msg**

16 String message to be returned to user (string)

17 **IN** **procs**

18 Array of [pmix_proc_t](#) structures (array of handles)

19 **IN** **nprocs**

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

21 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.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
4             const pmix_app_t apps[], size_t napps,
5             char nspace[])
```

C

6 **IN** `job_info`
7 Array of info structures (array of handles)
8 **IN** `ninfo`
9 Number of elements in the `job_info` array (integer)
10 **IN** `apps`
11 Array of `pmix_app_t` structures (array of handles)
12 **IN** `napps`
13 Number of elements in the `apps` array (integer)
14 **OUT** `nspace`
15 Namespace of the new job (string)

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

Required Attributes

17 PMIx libraries are not required to directly support any attributes for this function. However, any
18 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
19 required to add the following attributes to those provided before passing the request to the host:

20 **PMIX_SPAWNED** "pmix.spawned" (bool)
21 `true` if this process resulted from a call to `PMIx_Spawn`.
22 **PMIX_PARENT_ID** "pmix.parent" (pmix_proc_t)
23 Process identifier of the parent process of the calling process.
24 **PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)
25 The requesting process is a PMIx client.
26 **PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)
27 The requesting process is a PMIx tool.

28
29 Host environments that implement support for `PMIx_Spawn` are required to pass the
30 `PMIX_SPAWNED` and `PMIX_PARENT_ID` attributes to all PMIx servers launching new child
31 processes so those values can be returned to clients upon connection to the PMIx server. In
32 addition, they are required to support the following attributes when present in either the `job_info` or
33 the `info` array of an element of the `apps` array:

34 **PMIX_WDIR** "pmix.wdir" (char*)

1 Working directory for spawned processes.

2 **PMIX_SET_SESSION_CWD** "pmix.ssn cwd" (bool)

3 Set the application's current working directory to the session working directory assigned by
4 the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for
5 the rank to discover the session working directory assigned to the provided namespace

6 **PMIX_PREFIX** "pmix.prefix" (char*)

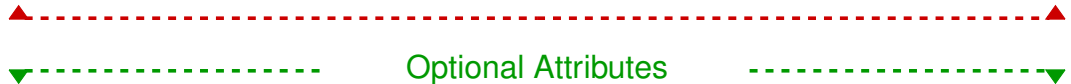
7 Prefix to use for starting spawned processes.

8 **PMIX_HOST** "pmix.host" (char*)

9 Comma-delimited list of hosts to use for spawned processes.

10 **PMIX_HOSTFILE** "pmix.hostfile" (char*)

11 Hostfile to use for spawned processes.



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

13 **PMIX_ADD_HOSTFILE** "pmix.addhostfile" (char*)

14 Hostfile listing hosts to add to existing allocation.

15 **PMIX_ADD_HOST** "pmix.addhost" (char*)

16 Comma-delimited list of hosts to add to the allocation.

17 **PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)

18 Preload binaries onto nodes.

19 **PMIX_PRELOAD_FILES** "pmix.preloadfiles" (char*)

20 Comma-delimited list of files to pre-position on nodes.

21 **PMIX_PERSONALITY** "pmix.pers" (char*)

22 Name of personality to use.

23 **PMIX_MAPPER** "pmix.mapper" (char*)

24 Mapping mechanism to use for placing spawned processes - when accessed using
25 **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping
26 mechanism used for the provided namespace.

27 **PMIX_DISPLAY_MAP** "pmix.dispmap" (bool)

28 Display process mapping upon spawn.

29 **PMIX_PPR** "pmix.ppr" (char*)

30 Number of processes to spawn on each identified resource.

31 **PMIX_MAPBY** "pmix.mapby" (char*)

1 Process mapping policy - when accessed using `PMIx_Get` , use the
2 `PMIX_RANK_WILDCARD` value for the rank to discover the mapping policy used for the
3 provided namespace

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

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

12 **PMIX_NON_PMI** "pmix.nonpmi" (bool)
13 Spawned processes will not call `PMIx_Init` .

14 **PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)
15 Spawned process rank that is to receive `stdin`.

16 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)
17 Forward this process's `stdin` to the designated process.

18 **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)
19 Forward `stdout` from spawned processes to this process.

20 **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)
21 Forward `stderr` from spawned processes to this process.

22 **PMIX_DEBUGGER_DAEMONS** "pmix.debugger" (bool)
23 Spawned application consists of debugger daemons.

24 **PMIX_TAG_OUTPUT** "pmix.tagout" (bool)
25 Tag application output with the identity of the source process.

26 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
27 Timestamp output from applications.

28 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
29 Merge `stdout` and `stderr` streams from application processes.

30 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)
31 Output application output to the specified file.

32 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)
33 Mark the `argv` with the rank of the process.

34 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)

1 Number of cpus to assign to each rank - when accessed using `PMIx_Get` , use the
2 `PMIX_RANK_WILDCARD` value for the rank to discover the cpus/process assigned to the
3 provided namespace

4 `PMIX_NO_PROCS_ON_HEAD` "pmix.nolocal" (bool)
5 Do not place processes on the head node.

6 `PMIX_NO_OVERSUBSCRIBE` "pmix.noover" (bool)
7 Do not oversubscribe the cpus.

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

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

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

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

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



22 **Description**

23 Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspc*
24 parameter. A `NULL` value in that location indicates that the caller doesn't wish to have the
25 namespace returned. The *nspc* array must be at least of size one more than `PMIX_MAX_NSLEN` .

26 By default, the spawned processes will be PMIx “connected” to the parent process upon successful
27 launch (see `PMIx_Connect` description for details). Note that this only means that (a) the parent
28 process will be given a copy of the new job’s information so it can query job-level info without
29 incurring any communication penalties, (b) newly spawned child processes will receive a copy of
30 the parent processes job-level info, and (c) both the parent process and members of the child job
31 will receive notification of errors from processes in their combined assemblage.



Advice to users

32 Behavior of individual resource managers may differ, but it is expected that failure of any
33 application process to start will result in termination/cleanup of all processes in the newly spawned
34 job and return of an error code to the caller.



1 6.2.2 PMIx_Spawn_nb

2 Summary

3 Nonblocking version of the [PMIx_Spawn](#) routine.

4 Format

PMIx v1.0

C

5 `pmix_status_t`

```
6 PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,  
7               const pmix_app_t apps[], size_t napps,  
8               pmix_spawn_cbfunc_t cbfunc, void *cbdata)
```

C

9 **IN** `job_info`

10 Array of info structures (array of handles)

11 **IN** `ninfo`

12 Number of elements in the `job_info` array (integer)

13 **IN** `apps`

14 Array of [pmix_app_t](#) structures (array of handles)

15 **IN** `cbfunc`

16 Callback function [pmix_spawn_cbfunc_t](#) (function reference)

17 **IN** `cbdata`

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

19 Returns one of the following:

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

Required Attributes

24 PMIx libraries are not required to directly support any attributes for this function. However, any
25 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
26 required to add the following attributes to those provided before passing the request to the host:

27 **PMIX_SPAWNED** "`pmix.spawned`" (bool)

28 `true` if this process resulted from a call to [PMIx_Spawn](#).

29 **PMIX_PARENT_ID** "`pmix.parent`" (`pmix_proc_t`)

30 Process identifier of the parent process of the calling process.

31 **PMIX_REQUESTOR_IS_CLIENT** "`pmix.req.client`" (bool)

32 The requesting process is a PMIx client.

33 **PMIX_REQUESTOR_IS_TOOL** "`pmix.req.tool`" (bool)

1 The requesting process is a PMIx tool.

2
3 Host environments that implement support for `PMIx_Spawn` are required to pass the
4 `PMIX_SPAWNED` and `PMIX_PARENT_ID` attributes to all PMIx servers launching new child
5 processes so those values can be returned to clients upon connection to the PMIx server. In
6 addition, they are required to support the following attributes when present in either the `job_info` or
7 the `info` array of an element of the `apps` array:

8 `PMIX_WDIR` "pmix.wdir" (char*)

9 Working directory for spawned processes.

10 `PMIX_SET_SESSION_CWD` "pmix.ssn cwd" (bool)

11 Set the application's current working directory to the session working directory assigned by
12 the RM - when accessed using `PMIx_Get`, use the `PMIX_RANK_WILDCARD` value for
13 the rank to discover the session working directory assigned to the provided namespace

14 `PMIX_PREFIX` "pmix.prefix" (char*)

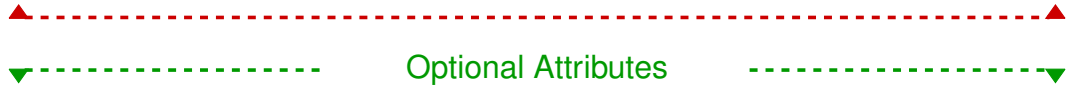
15 Prefix to use for starting spawned processes.

16 `PMIX_HOST` "pmix.host" (char*)

17 Comma-delimited list of hosts to use for spawned processes.

18 `PMIX_HOSTFILE` "pmix.hostfile" (char*)

19 Hostfile to use for spawned processes.



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

21 `PMIX_ADD_HOSTFILE` "pmix.addhostfile" (char*)

22 Hostfile listing hosts to add to existing allocation.

23 `PMIX_ADD_HOST` "pmix.addhost" (char*)

24 Comma-delimited list of hosts to add to the allocation.

25 `PMIX_PRELOAD_BIN` "pmix.preloadbin" (bool)

26 Preload binaries onto nodes.

27 `PMIX_PRELOAD_FILES` "pmix.preloadfiles" (char*)

28 Comma-delimited list of files to pre-position on nodes.

29 `PMIX_PERSONALITY` "pmix.pers" (char*)

30 Name of personality to use.

31 `PMIX_MAPPER` "pmix.mapper" (char*)

1 Mapping mechanism to use for placing spawned processes - when accessed using
2 [PMIx_Get](#) , use the [PMIX_RANK_WILDCARD](#) value for the rank to discover the mapping
3 mechanism used for the provided namespace.

4 [PMIX_DISPLAY_MAP](#) "pmix.dispmap" (bool)

5 Display process mapping upon spawn.

6 [PMIX_PPR](#) "pmix.ppr" (char*)

7 Number of processes to spawn on each identified resource.

8 [PMIX_MAPBY](#) "pmix.mapby" (char*)

9 Process mapping policy - when accessed using [PMIx_Get](#) , use the
10 [PMIX_RANK_WILDCARD](#) value for the rank to discover the mapping policy used for the
11 provided namespace

12 [PMIX_RANKBY](#) "pmix.rankby" (char*)

13 Process ranking policy - when accessed using [PMIx_Get](#) , use the
14 [PMIX_RANK_WILDCARD](#) value for the rank to discover the ranking algorithm used for the
15 provided namespace

16 [PMIX_BINDTO](#) "pmix.bindto" (char*)

17 Process binding policy - when accessed using [PMIx_Get](#) , use the
18 [PMIX_RANK_WILDCARD](#) value for the rank to discover the binding policy used for the
19 provided namespace

20 [PMIX_NON_PMI](#) "pmix.nonpmi" (bool)

21 Spawned processes will not call [PMIx_Init](#) .

22 [PMIX_STDIN_TGT](#) "pmix.stdin" (uint32_t)

23 Spawned process rank that is to receive `stdin`.

24 [PMIX_FWD_STDIN](#) "pmix.fwd.stdin" (bool)

25 Forward this process's `stdin` to the designated process.

26 [PMIX_FWD_STDOUT](#) "pmix.fwd.stdout" (bool)

27 Forward `stdout` from spawned processes to this process.

28 [PMIX_FWD_STDERR](#) "pmix.fwd.stderr" (bool)

29 Forward `stderr` from spawned processes to this process.

30 [PMIX_DEBUGGER_DAEMONS](#) "pmix.debugger" (bool)

31 Spawned application consists of debugger daemons.

32 [PMIX_TAG_OUTPUT](#) "pmix.tagout" (bool)

33 Tag application output with the identity of the source process.

34 [PMIX_TIMESTAMP_OUTPUT](#) "pmix.tsout" (bool)

35 Timestamp output from applications.

36 [PMIX_MERGE_STDERR_STDOUT](#) "pmix.mergeerrout" (bool)

1 Merge `stdout` and `stderr` streams from application processes.

2 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)

3 Output application output to the specified file.

4 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)

5 Mark the `argv` with the rank of the process.

6 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)

7 Number of cpus to assign to each rank - when accessed using `PMIx_Get`, use the
8 **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the
9 provided namespace

10 **PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)

11 Do not place processes on the head node.

12 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)

13 Do not oversubscribe the cpus.

14 **PMIX_REPORT_BINDINGS** "pmix.repbinding" (bool)

15 Report bindings of the individual processes.

16 **PMIX_CPU_LIST** "pmix.cpulist" (char*)

17 List of cpus to use for this job - when accessed using `PMIx_Get`, use the
18 **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided
19 namespace

20 **PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)

21 Application supports recoverable operations.

22 **PMIX_JOB_CONTINUOUS** "pmix.continuous" (bool)

23 Application is continuous, all failed processes should be immediately restarted.

24 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)

25 Maximum number of times to restart a job - when accessed using `PMIx_Get`, use the
26 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
27 namespace



28 Description

29 Nonblocking version of the `PMIx_Spawn` routine. The provided callback function will be
30 executed upon successful start of *all* specified application processes.

Advice to users

31 Behavior of individual resource managers may differ, but it is expected that failure of any
32 application process to start will result in termination/cleanup of all processes in the newly spawned
33 job and return of an error code to the caller.

1 6.3 Connecting and Disconnecting Processes

2 This section defines functions to connect and disconnect processes in two or more separate PMIx
3 namespaces. The PMIx definition of *connected* solely implies the following:

- 4 • job-level information for each namespace involved in the operation is to be made available to all
5 processes in the connected assemblage
- 6 • any data posted by a process in the connected assemblage (via calls to **PMIx_Put** committed via
7 **PMIx_Commit**) prior to execution of the **PMIx_Connect** operation is to be made accessible
8 to all processes in the assemblage - any data posted after execution of the *connect* operation must
9 be exchanged via a separate **PMIx_Fence** operation spanning the connected processes
- 10 • the host environment should treat the failure of any process in the assemblage as a reportable
11 event, taking action on the assemblage as if it were a single application. For example, if the
12 environment defaults (in the absence of any application directives) to terminating an application
13 upon failure of any process in that application, then the environment should terminate all
14 processes in the connected assemblage upon failure of any member.

▼ Advice to PMIx server hosts ▼

15 The host environment may choose to assign a new namespace to the connected assemblage and/or
16 assign new ranks for its members for its own internal tracking purposes. However, it is not required
17 to communicate such assignments to the participants (e.g., in response to an appropriate call to
18 **PMIx_Query_info_nb**). The host environment is required to generate a
19 **PMIX_ERR_INVALID_TERMINATION** event should any process in the assemblage terminate or
20 call **PMIx_Finalize** without first *disconnecting* from the assemblage.

▼ Advice to users ▼

21 Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation.
22 While not explicitly prohibited, users are advised that a PMIx implementation or host environment
23 may return an error in such cases.

24 Neither the PMIx implementation nor host environment are required to provide any tracking
25 support for the assemblage. Thus, the application is responsible for maintaining the membership
26 list of the assemblage.

27 6.3.1 PMIx_Connect

28 Summary

29 Connect namespaces.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,
4              const pmix_info_t info[], size_t ninfo)
```

C

- 5 **IN** `procs`
- 6 Array of proc structures (array of handles)
- 7 **IN** `nprocs`
- 8 Number of elements in the `procs` array (integer)
- 9 **IN** `info`
- 10 Array of info structures (array of handles)
- 11 **IN** `ninfo`
- 12 Number of elements in the `info` array (integer)

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

Required Attributes

14 PMIx libraries are not required to directly support any attributes for this function. However, any
15 provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

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

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

21 **PMIX_COLLECTIVE_ALGO** "`pmix.calgo`" (**char***)
22 Comma-delimited list of algorithms to use for the collective operation. PMIx does not
23 impose any requirements on a host environment's collective algorithms. Thus, the
24 acceptable values for this attribute will be environment-dependent - users are encouraged to
25 check their host environment for supported values.

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

Advice to PMIx library implementers

1 We recommend that implementation of the `PMIX_TIMEOUT` attribute be left to the host
2 environment due to race condition considerations between completion of the operation versus
3 internal timeout in the PMIx server library. Implementers that choose to support `PMIX_TIMEOUT`
4 directly in the PMIx server library must take care to resolve the race condition and should avoid
5 passing `PMIX_TIMEOUT` to the host environment so that multiple competing timeouts are not
6 created.

Description

7 Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The
8 function will return once all processes identified in *procs* have called either `PMIx_Connect` or
9 its non-blocking version, *and* the host environment has completed any supporting operations
10 required to meet the terms of the PMIx definition of *connected* processes.
11

Advice to users

12 All processes engaged in a given `PMIx_Connect` operation must provide the identical *procs*
13 array as ordering of entries in the array and the method by which those processes are identified
14 (e.g., use of `PMIX_RANK_WILDCARD` versus listing the individual processes) *may* impact the
15 host environment's algorithm for uniquely identifying an operation.

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

22 Processes that combine via `PMIx_Connect` must call `PMIx_Disconnect` prior to finalizing
23 and/or terminating - any process in the assemblage failing to meet this requirement will cause a
24 `PMIX_ERR_INVALID_TERMINATION` event to be generated.

25 A process can only engage in one connect operation involving the identical *procs* array at a time.
26 However, a process can be simultaneously engaged in multiple connect operations, each involving a
27 different *procs* array.

28 As in the case of the `PMIx_Fence` operation, the *info* array can be used to pass user-level
29 directives regarding the algorithm to be used for any collective operation involved in the operation,
30 timeout constraints, and other options available from the host RM.

1 6.3.2 PMIx_Connect_nb

2 Summary

3 Nonblocking [PMIx_Connect_nb](#) routine.

4 Format

PMIx v1.0

C

5 `pmix_status_t`

```
6 PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,  
7                 const pmix_info_t info[], size_t ninfo,  
8                 pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

9 **IN** `procs`

10 Array of proc structures (array of handles)

11 **IN** `nprocs`

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

13 **IN** `info`

14 Array of info structures (array of handles)

15 **IN** `ninfo`

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

17 **IN** `cbfunc`

18 Callback function [pmix_op_cbfunc_t](#) (function reference)

19 **IN** `cbdata`

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

21 Returns one of the following:

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

Required Attributes

29 PMIx libraries are not required to directly support any attributes for this function. However, any
30 provided attributes must be passed to the host 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.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
4                 const pmix_info_t info[], size_t ninfo);
```

C

- 5 **IN procs**
Array of proc structures (array of handles)
- 6
- 7 **IN nprocs**
Number of elements in the *procs* array (integer)
- 8
- 9 **IN info**
Array of info structures (array of handles)
- 10
- 11 **IN ninfo**
Number of element in the *info* array (integer)
- 12

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.

1
PMIx v1.0

Format

C

```

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

```

C

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

18 Returns one of the following:

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

Required Attributes

26 PMIx libraries are not required to directly support any attributes for this function. However, any
27 provided attributes must be passed to the host SMS daemon for processing.

Optional Attributes

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

- 29 **PMIX_TIMEOUT** "pmix.timeout" (int)
30 Time in seconds before the specified operation should time out (0 indicating infinite) in
31 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
32 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)

Returns [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant. In the event the function returns an error, the *regcbfunc* will *not* be called.

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

```
pmix_status_t
PMIx_IOF_deregister(size_t iofhdlr,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

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)

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

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)

Returns one of the following:

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

Required Attributes

8 The following attributes are required for PMIx libraries that support IO forwarding:

9 **PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (**uint32_t**)

10 The requested size of the server cache in bytes for each specified channel. By default, the
11 server is allowed (but not required) to drop all bytes received beyond the max size.

12 **PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (**bool**)

13 In an overflow situation, drop the oldest bytes to make room in the cache.

14 **PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (**bool**)

15 In an overflow situation, drop any new bytes received until room becomes available in the
16 cache (default).

Optional Attributes

17 The following attributes are optional for PMIx libraries that support IO forwarding:

18 **PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (**uint32_t**)

19 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
20 IO arrives. The library will execute the callback whenever the specified number of bytes
21 becomes available. Any remaining buffered data will be “flushed” upon call to deregister the
22 respective channel.

23 **PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (**uint32_t**)

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

1
2
3
4
5
6

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

1 The job management APIs provide an application with the ability to orchestrate its operation in
2 partnership with the SMS. Members of this category include the
3 [PMIx_Allocation_request_nb](#), [PMIx_Job_control_nb](#), and
4 [PMIx_Process_monitor_nb](#) APIs.

5 7.1 Query

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

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

20 PMIx supports this use-case by defining an attribute key ([PMIX_TIME_REMAINING](#)) that can be
21 used with the [PMIx_Query_info_nb](#) interface to obtain the number of seconds remaining in
22 the current job allocation. Note that one could alternatively use the
23 [PMIx_Register_event_handler](#) API to register for an event indicating incipient job
24 termination, and then use the [PMIx_Job_control_nb](#) API to request that the host SMS
25 generate an event a specified amount of time prior to reaching the maximum run time. PMIx
26 provides such alternate methods as a means of maximizing the probability of a host system
27 supporting at least one method by which the application can obtain the desired service.

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

29 7.1.1 [PMIx_Resolve_peers](#)

30 Summary

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

1 **Format**
PMIx v1.0 C

```
2 pmix_status_t  
3 PMIx_Resolve_peers(const char *nodename,  
4                   const pmix_namespace_t nspace,  
5                   pmix_proc_t **procs, size_t *nprocs)
```

- 6 **IN** **nodename**
7 Name of the node to query (string)
- 8 **IN** **nspace**
9 namespace (string)
- 10 **OUT** **procs**
11 Array of process structures (array of handles)
- 12 **OUT** **nprocs**
13 Number of elements in the *procs* array (integer)

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

15 **Description**

16 Given a *nodename*, return the array of processes within the specified *nspace* that are executing on
17 that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified
18 node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will
19 be 0. The caller is responsible for releasing the *procs* array when done with it. The
20 **PMIX_PROC_FREE** macro is provided for this purpose.

21 **7.1.2 PMIx_Resolve_nodes**

22 **Summary**

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

24 **Format**
PMIx v1.0 C

```
25 pmix_status_t  
26 PMIx_Resolve_nodes(const char *nspace, char **nodelist)
```

- 27 **IN** **nspace**
28 Namespace (string)
- 29 **OUT** **nodelist**
30 Comma-delimited list of nodenames (string)

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

1
2
3
4

5
6
7
8
9
10
11

12
13
14
15
16
17
18
19

20
21
22
23
24
25
26
27
28
29
30
31
32

Description

Given a *nospace*, 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_nb

Summary

Query information about the system in general.

Format

PMIx v2.0

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

IN queries

Array of query structures (array of handles)

IN nqueries

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

IN cbfunc

Callback function [pmix_info_cbfunc_t](#) (function reference)

IN cbdata

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

Returns one of the following:

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

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

- **PMIX_SUCCESS** All data 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_NAMESPACE** 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_NAMESPACE "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_NAMESPACE** attribute. Only required when the request is for information on a specific process.

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

2 Query list of supported attributes for specified APIs

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

4 Request attributes supported by the PMIx client library

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

6 Request attributes supported by the PMIx server library

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

8 Request attributes supported by the host environment

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

10 Request attributes supported by the PMIx tool library functions

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

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

21 Host environments that support this operation are required to support the following attributes as
22 qualifiers to the request:

23 **PMIX_PROCID** "pmix.procid" (**pmix_proc_t**)

24 Process identifier Specifies the process ID whose information is being requested - e.g., a
25 query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the
26 request is for information on a specific process.

27 **PMIX_NAMESPACE** "pmix.namespace" (**char***)

28 Namespace of the job. Specifies the namespace of the process whose information is being
29 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
30 be accompanied by the **PMIX_RANK** attribute. Only required when the request is for
31 information on a specific process.

32 **PMIX_RANK** "pmix.rank" (**pmix_rank_t**)

33 Process rank within the job. Specifies the rank of the process whose information is being
34 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
35 be accompanied by the **PMIX_NAMESPACE** attribute. Only required when the request is for
36 information on a specific process.

1 Note that inclusion of the `PMIX_PROCID` directive and either the `PMIX_NAMESPACE` or the
2 `PMIX_RANK` attribute will return a `PMIX_ERR_BAD_PARAM` result, and that the inclusion of a
3 process identifier must apply to all keys in that `pmix_query_t`. Queries for information on
4 multiple specific processes therefore requires submitting multiple `pmix_query_t` structures,
5 each referencing one process.

Optional Attributes

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

7 `PMIX_QUERY_NAMESPACES` "pmix.qry.ns" (char*)

8 Request a comma-delimited list of active namespaces.

9 `PMIX_QUERY_JOB_STATUS` "pmix.qry.jst" (`pmix_status_t`)

10 Status of a specified, currently executing job.

11 `PMIX_QUERY_QUEUE_LIST` "pmix.qry qlst" (char*)

12 Request a comma-delimited list of scheduler queues.

13 `PMIX_QUERY_QUEUE_STATUS` "pmix.qry.qst" (TBD)

14 Status of a specified scheduler queue.

15 `PMIX_QUERY_PROC_TABLE` "pmix.qry.phtable" (char*)

16 Input namespace of the job whose information is being requested returns (
17 `pmix_data_array_t`) an array of `pmix_proc_info_t`.

18 `PMIX_QUERY_LOCAL_PROC_TABLE` "pmix.qry.lhtable" (char*)

19 Input namespace of the job whose information is being requested returns (
20 `pmix_data_array_t`) an array of `pmix_proc_info_t` for processes in job on same
21 node.

22 `PMIX_QUERY_SPAWN_SUPPORT` "pmix.qry.spawn" (bool)

23 Return a comma-delimited list of supported spawn attributes.

24 `PMIX_QUERY_DEBUG_SUPPORT` "pmix.qry.debug" (bool)

25 Return a comma-delimited list of supported debug attributes.

26 `PMIX_QUERY_MEMORY_USAGE` "pmix.qry.mem" (bool)

27 Return information on memory usage for the processes indicated in the qualifiers.

28 `PMIX_QUERY_REPORT_AVG` "pmix.qry.avg" (bool)

29 Report only average values for sampled information.

30 `PMIX_QUERY_REPORT_MINMAX` "pmix.qry.minmax" (bool)

31 Report minimum and maximum values.

32 `PMIX_QUERY_ALLOC_STATUS` "pmix.query.alloc" (char*)

33 String identifier of the allocation whose status is being requested.

34 `PMIX_TIME_REMAINING` "pmix.time.remaining" (char*)

1 Query number of seconds (`uint32_t`) remaining in allocation for the specified namespace.
2

3 **PMIX_SERVER_URI** "pmix.srvr.uri" (`char*`)

4 URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
5 PMIx connection. Defaults to requesting the information for the local PMIx server.

6 **PMIX_PROC_URI** "pmix.puri" (`char*`)

7 URI containing contact information for a given process. Requests the URI of the specified
8 PMIx server's out-of-band connection. Defaults to requesting the information for the local
9 PMIx server.



10 Description

11 Query information about the system in general. This can include a list of active namespaces,
12 network topology, etc. Also can be used to query node-specific info such as the list of peers
13 executing on a given node. We assume that the host RM will exercise appropriate access control on
14 the information.

15 NOTE: There is no blocking form of this API as the structures passed to query info differ from
16 those for receiving the results.

17 The *status* argument to the callback function indicates if requested data was found or not. An array
18 of `pmix_info_t` will contain each key that was provided and the corresponding value that was
19 found. Requests for keys that are not found will return the key paired with a value of type
20 `PMIX_UNDEF`.

Advice to PMIx library implementers

21 Information returned from `PMIx_Query_info_nb` shall be locally cached so that retrieval by
22 subsequent calls to `PMIx_Get` or `PMIx_Query_info_nb` can succeed with minimal overhead.
23 The local cache shall be checked prior to querying the PMIx server and/or the host environment.
24 Queries that include the `PMIX_QUERY_REFRESH_CACHE` attribute shall bypass the local cache
25 and retrieve a new value for the query, refreshing the values in the cache upon return.

1 7.1.3.1 Using `PMIx_Get` vs `PMIx_Query_info_nb`

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

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

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

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

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

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

29 7.1.3.2 Accessing attribute support information

30 Information as to attributes supported by either the PMIx implementation or its host environment
31 can be obtained via the `PMIx_Query_info_nb` API. The
32 `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute must be listed as the first entry in the *keys* field
33 of the `pmix_query_t` structure, followed by the name of the function whose attribute support is
34 being requested - support for multiple functions can be requested simultaneously by simply adding
35 the function names to the array of *keys*. Function names *must* be given as user-level API names -
36 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:

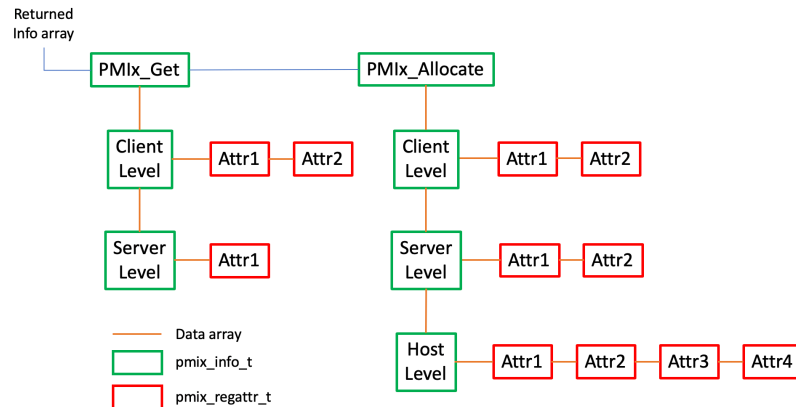


Figure 7.1.: Returned information hierarchy for attribute support request

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

- 1 • *Malleable* environments where the scheduler redirects resources away from executing
- 2 applications for higher priority jobs or load balancing
- 3 • *Resilient* applications that need to request replacement resources in the face of failures
- 4 • *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time,
- 5 but realizes that they underestimated their required time while executing
- 6 PMIx attempts to address this range of use-cases with a flexible API.

7 7.2.1 PMIx_Allocation_request

8 Summary

9 Request an allocation operation from the host resource manager.

10 Format

PMIx v3.0

C

```
11 pmix_status_t
12 PMIx_Allocation_request(pmix_alloc_directive_t directive,
13                        pmix_info_t info[], size_t ninfo);
```

C

- 14 **IN directive**
Allocation directive (handle)
- 15
- 16 **IN info**
Array of `pmix_info_t` structures (array of handles)
- 17
- 18 **IN ninfo**
Number of elements in the *info* array (integer)
- 19

20 Returns one of the following:

- 21 • **PMIX_SUCCESS**, indicating that the request was processed and returned *success*
- 22 • a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

23 PMIx libraries are not required to directly support any attributes for this function. However, any
 24 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
 25 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
 26 the request.

28 Host environments that implement support for this operation are required to support the following
 29 attributes:

30 **PMIX_ALLOC_ID** "pmix.alloc.id" (char*)

1 Provide a string identifier for this allocation request which can later be used to query status
2 of the request.

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

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

7 **PMIX_ALLOC_TIME** "pmix.alloc.time" (uint32_t)
8 Time in seconds.



▼----- Optional Attributes -----▼

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

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

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

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

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

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

22 **PMIX_ALLOC_NETWORK_ID** "pmix.alloc.netid" (char*)
23 The key to be used when accessing this requested network allocation. The allocation will be
24 returned/stored as a **pmix_data_array_t** of **pmix_info_t** indexed by this key and
25 containing at least one entry with the same key and the allocated resource description. The
26 type of the included value depends upon the network support. For example, a TCP allocation
27 might consist of a comma-delimited string of socket ranges such as
28 "32000-32100,33005,38123-38146". Additional entries will consist of any provided
29 resource request directives, along with their assigned values. Examples include:
30 **PMIX_ALLOC_NETWORK_TYPE** - the type of resources provided;
31 **PMIX_ALLOC_NETWORK_PLANE** - if applicable, what plane the resources were assigned
32 from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** -
33 the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the
34 requested network allocation. NOTE: the assigned values may differ from those requested,
35 especially if **PMIX_INFO_REQD** was not set in the request.

36 **PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (float)

1 Mbits/sec.

2 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)

3 Quality of service level.

4 **PMIX_ALLOC_NETWORK_TYPE** "pmix.alloc.nettype" (char*)

5 Type of desired transport (e.g., "tcp", "udp")

6 **PMIX_ALLOC_NETWORK_PLANE** "pmix.alloc.netplane" (char*)

7 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

8 **PMIX_ALLOC_NETWORK_ENDPTS** "pmix.alloc.endpts" (size_t)

9 Number of endpoints to allocate per process

10 **PMIX_ALLOC_NETWORK_ENDPTS_NODE** "pmix.alloc.endpts.nd" (size_t)

11 Number of endpoints to allocate per node

12 **PMIX_ALLOC_NETWORK_SEC_KEY** "pmix.alloc.nsec" (pmix_byte_object_t)

13 Network security key



14 **Description**

15 Request an allocation operation from the host resource manager. Several broad categories are
16 envisioned, including the ability to:

- 17 • Request allocation of additional resources, including memory, bandwidth, and compute. This
18 should be accomplished in a non-blocking manner so that the application can continue to
19 progress while waiting for resources to become available. Note that the new allocation will be
20 disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one
21 allocation will not impact the other.
- 22 • Extend the reservation on currently allocated resources, subject to scheduling availability and
23 priorities. This includes extending the time limit on current resources, and/or requesting
24 additional resources be allocated to the requesting job. Any additional allocated resources will be
25 considered as part of the current allocation, and thus will be released at the same time.
- 26 • Return no-longer-required resources to the scheduler. This includes the "loan" of resources back
27 to the scheduler with a promise to return them upon subsequent request.

28 **7.2.2 PMIx_Allocation_request_nb**

29 **Summary**

30 Request an allocation operation from the host resource manager.

1
PMIx v2.0

Format

C

```

2 pmix_status_t
3 PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
4                             pmix_info_t info[], size_t ninfo,
5                             pmix_info_cbfunc_t cbfunc, void *cbdata);

```

C

- 6 **IN directive**
Allocation directive (handle)
- 8 **IN info**
Array of `pmix_info_t` structures (array of handles)
- 10 **IN ninfo**
Number of elements in the *info* array (integer)
- 12 **IN cbfunc**
Callback function `pmix_info_cbfunc_t` (function reference)
- 14 **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

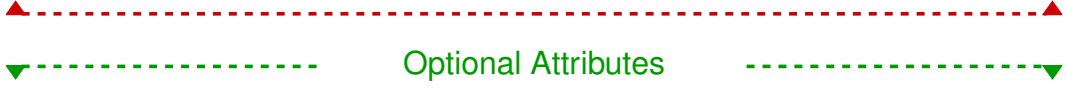
Required Attributes

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_ID** "pmix.alloc.id" (char*)
Provide a string identifier for this allocation request which can later be used to query status of the request.
- PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (uint64_t)

1 The number of nodes.
2 **PMIX_ALLOC_NUM_CPUS** "pmix.alloc.ncpus" (uint64_t)
3 Number of cpus.
4 **PMIX_ALLOC_TIME** "pmix.alloc.time" (uint32_t)
5 Time in seconds.



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

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

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

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

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

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

19 **PMIX_ALLOC_NETWORK_ID** "pmix.alloc.netid" (char*)
20 The key to be used when accessing this requested network allocation. The allocation will be
21 returned/stored as a `pmix_data_array_t` of `pmix_info_t` indexed by this key and
22 containing at least one entry with the same key and the allocated resource description. The
23 type of the included value depends upon the network support. For example, a TCP allocation
24 might consist of a comma-delimited string of socket ranges such as
25 "32000-32100,33005,38123-38146". Additional entries will consist of any provided
26 resource request directives, along with their assigned values. Examples include:
27 **PMIX_ALLOC_NETWORK_TYPE** - the type of resources provided;
28 **PMIX_ALLOC_NETWORK_PLANE** - if applicable, what plane the resources were assigned
29 from; **PMIX_ALLOC_NETWORK_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** -
30 the allocated bandwidth; **PMIX_ALLOC_NETWORK_SEC_KEY** - a security key for the
31 requested network allocation. NOTE: the assigned values may differ from those requested,
32 especially if **PMIX_INFO_REQD** was not set in the request.

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

35 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)

1 Quality of service level.

2 **PMIX_ALLOC_NETWORK_TYPE** "pmix.alloc.nettype" (char*)

3 Type of desired transport (e.g., "tcp", "udp")

4 **PMIX_ALLOC_NETWORK_PLANE** "pmix.alloc.netplane" (char*)

5 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

6 **PMIX_ALLOC_NETWORK_ENDPTS** "pmix.alloc.endpts" (size_t)

7 Number of endpoints to allocate per process

8 **PMIX_ALLOC_NETWORK_ENDPTS_NODE** "pmix.alloc.endpts.nd" (size_t)

9 Number of endpoints to allocate per node

10 **PMIX_ALLOC_NETWORK_SEC_KEY** "pmix.alloc.nsec" (pmix_byte_object_t)

11 Network security key



12 **Description**

13 Non-blocking form of the [PMIx_Allocation_request](#) API.

14 **7.3 Job Control**

15 This section defines APIs that enable the application and host environment to coordinate the
16 response to failures and other events. This can include requesting termination of the entire job or a
17 subset of processes within a job, but can also be used in combination with other PMIx capabilities
18 (e.g., allocation support and event notification) for more nuanced responses. For example, an
19 application notified of an incipient over-temperature condition on a node could use the
20 [PMIx_Allocation_request_nb](#) interface to request replacement nodes while
21 simultaneously using the [PMIx_Job_control_nb](#) interface to direct that a checkpoint event be
22 delivered to all processes in the application. If replacement resources are not available, the
23 application might use the [PMIx_Job_control_nb](#) interface to request that the job continue at
24 a lower power setting, perhaps sufficient to avoid the over-temperature failure.

25 The job control APIs can also be used by an application to register itself as available for preemption
26 when operating in an environment such as a cloud or where incentives, financial or otherwise, are
27 provided to jobs willing to be preempted. Registration can include attributes indicating how many
28 resources are being offered for preemption (e.g., all or only some portion), whether the application
29 will require time to prepare for preemption, etc. Jobs that request a warning will receive an event
30 notifying them of an impending preemption (possibly including information as to the resources that
31 will be taken away, how much time the application will be given prior to being preempted, whether
32 the preemption will be a suspension or full termination, etc.) so they have an opportunity to save
33 their work. Once the application is ready, it calls the provided event completion callback function to
34 indicate that the SMS is free to suspend or terminate it, and can include directives regarding any
35 desired restart.

1 7.3.1 PMIx_Job_control

2 Summary

3 Request a job control action.

4 Format

PMIx v3.0

```
5 pmix_status_t
6 PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
7                 const pmix_info_t directives[], size_t ndirs)
```

8 IN targets

9 Array of proc structures (array of handles)

10 IN ntargets

11 Number of element in the *targets* array (integer)

12 IN directives

13 Array of info structures (array of handles)

14 IN ndirs

15 Number of element in the *directives* array (integer)

16 IN cbfunc

17 Callback function [pmix_info_cbfunc_t](#) (function reference)

18 IN cbdata

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

20 Returns one of the following:

- 21 • **PMIX_SUCCESS**, indicating that the request was processed by the host environment and
22 returned *success*
- 23 • a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

24 PMIx libraries are not required to directly support any attributes for this function. However, any
25 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
26 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
27 the request.

29 Host environments that implement support for this operation are required to support the following
30 attributes:

31 **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)

32 Provide a string identifier for this request.

33 **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)

1 Pause the specified processes.

2 **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)

3 Resume ("un-pause") the specified processes.

4 **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)

5 Forcibly terminate the specified processes and cleanup.

6 **PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)

7 Send given signal to specified processes.

8 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)

9 Politely terminate the specified processes.

10 **PMIX_REGISTER_CLEANUP** "pmix.reg.cleanup" (char*)

11 Comma-delimited list of files to be removed upon process termination

12 **PMIX_REGISTER_CLEANUP_DIR** "pmix.reg.cleanupdir" (char*)

13 Comma-delimited list of directories to be removed upon process termination

14 **PMIX_CLEANUP_RECURSIVE** "pmix.clnup.recurse" (bool)

15 Recursively cleanup all subdirectories under the specified one(s)

16 **PMIX_CLEANUP_EMPTY** "pmix.clnup.empty" (bool)

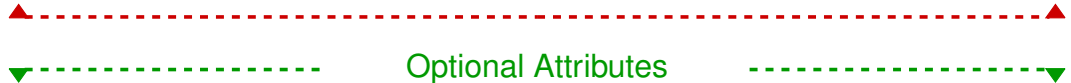
17 Only remove empty subdirectories

18 **PMIX_CLEANUP_IGNORE** "pmix.clnup.ignore" (char*)

19 Comma-delimited list of filenames that are not to be removed

20 **PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)

21 When recursively cleaning subdirectories, do not remove the top-level directory (the one
22 given in the cleanup request)



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

24 **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)

25 Cancel the specified request (**NULL** implies cancel all requests from this requestor).

26 **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)

27 Restart the specified processes using the given checkpoint ID.

28 **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)

29 Checkpoint the specified processes and assign the given ID to it.

30 **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)

31 Use event notification to trigger a process checkpoint.

32 **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)

1 Use the given signal to trigger a process checkpoint.

2 **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)

3 Time in seconds to wait for a checkpoint to complete.

4 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**

5 "pmix.jctrl.ckmethod" (pmix_data_array_t)

6 Array of **pmix_info_t** declaring each method and value supported by this application.

7 **PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)

8 Regular expression identifying nodes that are to be provisioned.

9 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvning" (char*)

10 Name of the image that is to be provisioned.

11 **PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

12 Indicate that the job can be pre-empted.



13 Description

14 Request a job control action. The *targets* array identifies the processes to which the requested job
15 control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's
16 namespace. The use of **PMIX_RANK_WILDARD** can also be used to indicate that all processes in
17 the given namespace are to be included.

18 The directives are provided as **pmix_info_t** structures in the *directives* array. The callback
19 function provides a *status* to indicate whether or not the request was granted, and to provide some
20 information as to the reason for any denial in the **pmix_info_cbfunc_t** array of
21 **pmix_info_t** structures.

22 7.3.2 PMIx_Job_control_nb

23 Summary

24 Request a job control action.

25 Format

PMIx v2.0

C

26 **pmix_status_t**

27 **PMIx_Job_control_nb**(const pmix_proc_t targets[], size_t ntargets,
28 const pmix_info_t directives[], size_t ndirs,
29 pmix_info_cbfunc_t cbfunc, void *cbdata)

1 **IN targets**
 2 Array of proc structures (array of handles)
 3 **IN ntargets**
 4 Number of element in the *targets* array (integer)
 5 **IN directives**
 6 Array of info structures (array of handles)
 7 **IN ndirs**
 8 Number of element in the *directives* array (integer)
 9 **IN cbfunc**
 10 Callback function `pmix_info_cbfunc_t` (function reference)
 11 **IN cbdata**
 12 Data to be passed to the callback function (memory reference)

13 Returns one of the following:

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

Required Attributes

21 PMIx libraries are not required to directly support any attributes for this function. However, any
 22 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
 23 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
 24 the request.

25

26 Host environments that implement support for this operation are required to support the following
 27 attributes:

28 **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
 29 Provide a string identifier for this request.

30 **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
 31 Pause the specified processes.

32 **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
 33 Resume ("un-pause") the specified processes.

34 **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)

1 Forcibly terminate the specified processes and cleanup.

2 **PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)

3 Send given signal to specified processes.

4 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)

5 Politely terminate the specified processes.

6 **PMIX_REGISTER_CLEANUP** "pmix.reg.cleanup" (char*)

7 Comma-delimited list of files to be removed upon process termination

8 **PMIX_REGISTER_CLEANUP_DIR** "pmix.reg.cleanupdir" (char*)

9 Comma-delimited list of directories to be removed upon process termination

10 **PMIX_CLEANUP_RECURSIVE** "pmix.clnup.recurse" (bool)

11 Recursively cleanup all subdirectories under the specified one(s)

12 **PMIX_CLEANUP_EMPTY** "pmix.clnup.empty" (bool)

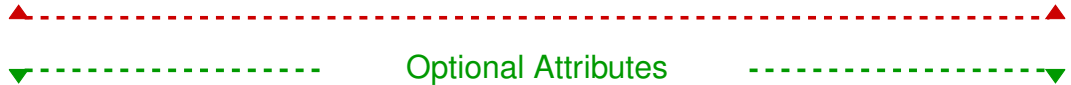
13 Only remove empty subdirectories

14 **PMIX_CLEANUP_IGNORE** "pmix.clnup.ignore" (char*)

15 Comma-delimited list of filenames that are not to be removed

16 **PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)

17 When recursively cleaning subdirectories, do not remove the top-level directory (the one
18 given in the cleanup request)



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

20 **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)

21 Cancel the specified request (**NULL** implies cancel all requests from this requestor).

22 **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)

23 Restart the specified processes using the given checkpoint ID.

24 **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)

25 Checkpoint the specified processes and assign the given ID to it.

26 **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)

27 Use event notification to trigger a process checkpoint.

28 **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)

29 Use the given signal to trigger a process checkpoint.

30 **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)

31 Time in seconds to wait for a checkpoint to complete.

32 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**

33 "pmix.jctrl.ckmethod" (pmix_data_array_t)

1 Array of `pmix_info_t` declaring each method and value supported by this application.

2 **PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)

3 Regular expression identifying nodes that are to be provisioned.

4 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvning" (char*)

5 Name of the image that is to be provisioned.

6 **PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

7 Indicate that the job can be pre-empted.



8 **Description**

9 Non-blocking form of the `PMIx_Job_control` API. The *targets* array identifies the processes
10 to which the requested job control action is to be applied. A `NULL` value can be used to indicate all
11 processes in the caller's namespace. The use of `PMIX_RANK_WILDARD` can also be used to
12 indicate that all processes in the given namespace are to be included.

13 The directives are provided as `pmix_info_t` structures in the *directives* array. The callback
14 function provides a *status* to indicate whether or not the request was granted, and to provide some
15 information as to the reason for any denial in the `pmix_info_cbfnc_t` array of
16 `pmix_info_t` structures.

17 **7.4 Process and Job Monitoring**

18 In addition to external faults, a common problem encountered in HPC applications is a failure to
19 make progress due to some internal conflict in the computation. These situations can result in a
20 significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the
21 job. Various watchdog methods have been developed for detecting this situation, including
22 requiring a periodic "heartbeat" from the application and monitoring a specified file for changes in
23 size and/or modification time.

24 At the request of SMS vendors and members, a monitoring support interface has been included in
25 the PMIx v2 standard. The defined API allows applications to request monitoring, directing what is
26 to be monitored, the frequency of the associated check, whether or not the application is to be
27 notified (via the event notification subsystem) of stall detection, and other characteristics of the
28 operation. In addition, heartbeat and file monitoring methods have been included in the PRI but are
29 active only when requested.

30 **7.4.1 PMIx_Process_monitor**

31 **Summary**

32 Request that application processes be monitored.

1
PMIx v3.0

Format

C

```
2 pmix_status_t
3 PMIx_Process_monitor(const pmix_info_t *monitor, pmix_status_t error,
4                      const pmix_info_t directives[], size_t ndirs)
```

C

- 5 **IN monitor**
- 6 info (handle)
- 7 **IN error**
- 8 status (integer)
- 9 **IN directives**
- 10 Array of info structures (array of handles)
- 11 **IN ndirs**
- 12 Number of elements in the *directives* array (integer)

13 Returns one of the following:

- 14 • **PMIX_SUCCESS**, indicating that the request was processed and returned *success*
- 15 • a PMIx error constant indicating either an error in the input or that the request was refused

Optional Attributes

16 The following attributes may be implemented by a PMIx library or by the host environment. If
17 supported by the PMIx server library, then the library must not pass the supported attributes to the
18 host environment. All attributes not directly supported by the server library must be passed to the
19 host environment if it supports this operation, and the library is *required* to add the
20 **PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

- 21 **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
- 22 Provide a string identifier for this request.
- 23 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
- 24 Identifier to be canceled (NULL means cancel all monitoring for this process).
- 25 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
- 26 The application desires to control the response to a monitoring event.
- 27 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)
- 28 Register to have the PMIx server monitor the requestor for heartbeats.
- 29 **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)
- 30 Time in seconds before declaring heartbeat missed.
- 31 **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrops" (uint32_t)
- 32 Number of heartbeats that can be missed before generating the event.
- 33 **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)

1 Register to monitor file for signs of life.

2 **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)

3 Monitor size of given file is growing to determine if the application is running.

4 **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)

5 Monitor time since last access of given file to determine if the application is running.

6 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)

7 Monitor time since last modified of given file to determine if the application is running.

8 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)

9 Time in seconds between checking the file.

10 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)

11 Number of file checks that can be missed before generating the event.



12 Description

13 Request that application processes be monitored via several possible methods. For example, that
14 the server monitor this process for periodic heartbeats as an indication that the process has not
15 become “wedged”. When a monitor detects the specified alarm condition, it will generate an event
16 notification using the provided error code and passing along any available relevant information. It
17 is up to the caller to register a corresponding event handler.

18 The *monitor* argument is an attribute indicating the type of monitor being requested. For example,
19 **PMIX_MONITOR_FILE** to indicate that the requestor is asking that a file be monitored.

20 The *error* argument is the status code to be used when generating an event notification alerting that
21 the monitor has been triggered. The range of the notification defaults to
22 **PMIX_RANGE_NAMESPACE**. This can be changed by providing a **PMIX_RANGE** directive.

23 The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency
24 of checking to be done

25 7.4.2 PMIx_Process_monitor_nb

26 Summary

27 Request that application processes be monitored.

1
PMIx v2.0

Format

C

```
2 pmix_status_t
3 PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,
4                          const pmix_info_t directives[], size_t ndirs,
5                          pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

6 **IN monitor**
7 info (handle)
8 **IN error**
9 status (integer)
10 **IN directives**
11 Array of info structures (array of handles)
12 **IN ndirs**
13 Number of elements in the *directives* array (integer)
14 **IN cbfunc**
15 Callback function `pmix_info_cbfunc_t` (function reference)
16 **IN cbdata**
17 Data to be passed to the callback function (memory reference)

18 Returns one of the following:

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

Optional Attributes

26 The following attributes may be implemented by a PMIx library or by the host environment. If
27 supported by the PMIx server library, then the library must not pass the supported attributes to the
28 host environment. All attributes not directly supported by the server library must be passed to the
29 host environment if it supports this operation, and the library is *required* to add the
30 **PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

31 **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
32 Provide a string identifier for this request.

33 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
34 Identifier to be canceled (NULL means cancel all monitoring for this process).

35 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)

1 The application desires to control the response to a monitoring event.

2 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)

3 Register to have the PMIx server monitor the requestor for heartbeats.

4 **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)

5 Time in seconds before declaring heartbeat missed.

6 **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)

7 Number of heartbeats that can be missed before generating the event.

8 **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)

9 Register to monitor file for signs of life.

10 **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)

11 Monitor size of given file is growing to determine if the application is running.

12 **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)

13 Monitor time since last access of given file to determine if the application is running.

14 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)

15 Monitor time since last modified of given file to determine if the application is running.

16 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)

17 Time in seconds between checking the file.

18 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)

19 Number of file checks that can be missed before generating the event.



20 Description

21 Non-blocking form of the [PMIx_Process_monitor](#) API. The *cbfunc* function provides a
22 *status* to indicate whether or not the request was granted, and to provide some information as to the
23 reason for any denial in the [pmix_info_cbfunc_t](#) array of [pmix_info_t](#) structures.

24 7.4.3 PMIx_Heartbeat

25 Summary

26 Send a heartbeat to the PMIx server library

27 Format

PMIx v2.0

28 **PMIx_Heartbeat** (void)

29 Description

30 A simplified macro wrapping [PMIx_Process_monitor_nb](#) that sends a heartbeat to the
31 PMIx server library.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

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.tstamp" (time_t)
Timestamp for log report
- PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstamp" (bool)
Generate timestamp for log
- PMIX_LOG_TAG_OUTPUT** "pmix.log.tag" (bool)
Label the output stream with the channel name (e.g., "stdout")
- PMIX_LOG_TIMESTAMP_OUTPUT** "pmix.log.tsout" (bool)
Print timestamp in output string
- PMIX_LOG_XML_OUTPUT** "pmix.log.xml" (bool)
Print the output stream in XML format

1 **PMIX_LOG_EMAIL** "pmix.log.email" (pmix_data_array_t)
2 Log via email based on **pmix_info_t** containing directives.
3 **PMIX_LOG_EMAIL_ADDR** "pmix.log.emaddr" (char*)
4 Comma-delimited list of email addresses that are to receive the message.
5 **PMIX_LOG_EMAIL_SUBJECT** "pmix.log.emsub" (char*)
6 Subject line for email.
7 **PMIX_LOG_EMAIL_MSG** "pmix.log.emmsg" (char*)
8 Message to be included in email.
9 **PMIX_LOG_JOB_RECORD** "pmix.log.jrec" (bool)
10 Log the provided information to the host environment's job record
11 **PMIX_LOG_GLOBAL_DATASTORE** "pmix.log.gstore" (bool)
12 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.



21 7.5.2 PMIx_Log_nb

22 Summary

23 Log data to a data service.

1
PMIx v2.0

Format

C

```

2 pmix_status_t
3 PMIx_Log_nb(const pmix_info_t data[], size_t ndata,
4             const pmix_info_t directives[], size_t ndirs,
5             pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN data**
Array of info structures (array of handles)
- 7
- 8 **IN ndata**
Number of elements in the *data* array (**size_t**)
- 9
- 10 **IN directives**
Array of info structures (array of handles)
- 11
- 12 **IN ndirs**
Number of elements in the *directives* array (**size_t**)
- 13
- 14 **IN cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)
- 15
- 16 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 17

18 Return codes are one of the following:

- 19 **PMIX_SUCCESS** The logging request is valid and is being processed. The resulting status from
- 20 the operation will be provided in the callback function. Note that the library must not invoke
- 21 the callback function prior to returning from the API.
- 22 **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
- 23 returned *success* - the *cbfunc* will *not* be called
- 24 **PMIX_ERR_BAD_PARAM** The logging request contains at least one incorrect entry that prevents
- 25 it from being processed. The callback function will not be called.
- 26 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function. The
- 27 callback function will not be called.

Required Attributes

28 If the PMIx library does not itself perform this operation, then it is required to pass any attributes
29 provided by the client to the host environment for processing. In addition, it must include the
30 following attributes in the passed *info* array:

- 31 **PMIX_USERID** "pmix.euid" (**uint32_t**)
32 Effective user id.
- 33 **PMIX_GRPID** "pmix.egid" (**uint32_t**)
34 Effective group id.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

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.tstamp" (time_t)
Timestamp for log report
- PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstamp" (bool)
Generate timestamp for log
- PMIX_LOG_TAG_OUTPUT** "pmix.log.tag" (bool)
Label the output stream with the channel name (e.g., "stdout")
- PMIX_LOG_TIMESTAMP_OUTPUT** "pmix.log.tsout" (bool)
Print timestamp in output string
- PMIX_LOG_XML_OUTPUT** "pmix.log.xml" (bool)
Print the output stream in XML format

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

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

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

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

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

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



Description

13 Log data subject to the services offered by the host environment. The data to be logged is provided
14 in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The
15 callback function will be executed when the log operation has been completed. The *data* and
16 *directives* arrays must be maintained until the callback is provided.

Advice to users

18 It is strongly recommended that the **PMIx_Log_nb** API not be used by applications for streaming
19 data as it is not a “performant” transport and can perturb the application since it involves the local
20 PMIx server and host SMS daemon. Note that a return of **PMIX_SUCCESS** only denotes that the
21 data was successfully handed to the appropriate system call (for local channels) or the host
22 environment and does not indicate receipt at the final destination.



CHAPTER 8

Event Notification

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

4 8.1 Notification and Management

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

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

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

▼ Advice to PMIx library implementers ▼

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

Advice to users

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

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

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

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

24 Users can specify the callback order of a handler within its category at the time of registration.
25 Ordering can be specified either by providing the relevant returned event handler registration ID or
26 using event handler names, if the user specified an event handler name when registering the
27 corresponding event. Thus, users can specify that a given handler be executed before or after
28 another handler should both handlers appear in an event chain (the ordering is ignored if the other
29 handler isn't included). Note that ordering does not imply immediate relationships. For example,
30 multiple handlers registered to be serviced after event handler *A* will all be executed after *A*, but are
31 not guaranteed to be executed in any particular order amongst themselves.

32 In addition, one event handler can be declared as the *first* handler to be executed in the chain. This
33 handler will *always* be called prior to any other handler, regardless of category, provided the
34 incoming event matches both the specified range and event code. Only one handler can be so
35 designated — attempts to designate additional handlers as *first* will return an error. Deregistration
36 of the declared *first* handler will re-open the position for subsequent assignment.

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

Advice to users

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

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

17 8.1.1 PMIx_Register_event_handler

18 Summary

19 Register an event handler

20 Format

PMIx v2.0

C

```
21 void  
22 PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,  
23                             pmix_info_t info[], size_t ninfo,  
24                             pmix_notification_fn_t evhdlr,  
25                             pmix_evhdlr_reg_cbfunc_t cbfunc,  
26                             void *cbdata);
```

C

27 **IN codes**
28 Array of status codes (array of [pmix_status_t](#))
29 **IN ncodes**
30 Number of elements in the *codes* array (**size_t**)
31 **IN info**
32 Array of info structures (array of handles)

1 **IN** **ninfo**
 2 Number of elements in the *info* array (**size_t**)
 3 **IN** **evhdlr**
 4 Event handler to be called **pmix_notification_fn_t** (function reference)
 5 **IN** **cbfunc**
 6 Callback function **pmix_evhdlr_reg_cbfunc_t** (function reference)
 7 **IN** **cbdata**
 8 Data to be passed to the cbfunc callback function (memory reference)

9 Upon completion, the callback will receive a status based on the following table:

- 10 **PMIX_SUCCESS** The event handler was successfully registered - the event handler identifier is
 11 returned in the callback.
 12 **PMIX_ERR_BAD_PARAM** One or more of the directives provided in the *info* array was
 13 unrecognized.
 14 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification,
 15 or the host SMS does not support notification of the specified event code.

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

▼----- Required Attributes -----▼

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

- 20 **PMIX_EVENT_HDLR_NAME** "pmix.evname" (**char***)
 21 String name identifying this handler.
 22 **PMIX_EVENT_HDLR_FIRST** "pmix.evfirst" (**bool**)
 23 Invoke this event handler before any other handlers.
 24 **PMIX_EVENT_HDLR_LAST** "pmix.evlast" (**bool**)
 25 Invoke this event handler after all other handlers have been called.
 26 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY** "pmix.evfirstcat" (**bool**)
 27 Invoke this event handler before any other handlers in this category.
 28 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY** "pmix.evlastcat" (**bool**)
 29 Invoke this event handler after all other handlers in this category have been called.
 30 **PMIX_EVENT_HDLR_BEFORE** "pmix.evbefore" (**char***)
 31 Put this event handler immediately before the one specified in the (**char***) value.
 32 **PMIX_EVENT_HDLR_AFTER** "pmix.evafter" (**char***)
 33 Put this event handler immediately after the one specified in the (**char***) value.
 34 **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (**bool**)
 35 Prepend this handler to the precedence list within its category.

1 **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (bool)
 2 Append this handler to the precedence list within its category.

3 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrangle" (pmix_data_array_t*)
 4 Array of pmix_proc_t defining range of event notification.

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

7 **PMIX_EVENT_RETURN_OBJECT** "pmix.evobject" (void *)
 8 Object to be returned whenever the registered callback function **cbfunc** is invoked. The
 9 object will only be returned to the process that registered it.

11 Host environments that implement support for PMIx event notification are required to support the
 12 following attributes:

13 **PMIX_EVENT_AFFECTED_PROC** "pmix.evproc" (pmix_proc_t)
 14 The single process that was affected.

15 **PMIX_EVENT_AFFECTED_PROCS** "pmix.evaffected" (pmix_data_array_t*)
 16 Array of pmix_proc_t defining affected processes.



▼----- Optional Attributes -----▼

17 Host environments that support PMIx event notification *may* offer notifications for environmental
 18 events impacting the job and for SMS events relating to the job. The following attributes are
 19 optional for host environments that support this operation:

20 **PMIX_EVENT_TERMINATE_SESSION** "pmix.evterm.sess" (bool)
 21 The RM intends to terminate this session.

22 **PMIX_EVENT_TERMINATE_JOB** "pmix.evterm.job" (bool)
 23 The RM intends to terminate this job.

24 **PMIX_EVENT_TERMINATE_NODE** "pmix.evterm.node" (bool)
 25 The RM intends to terminate all processes on this node.

26 **PMIX_EVENT_TERMINATE_PROC** "pmix.evterm.proc" (bool)
 27 The RM intends to terminate just this process.

28 **PMIX_EVENT_ACTION_TIMEOUT** "pmix.evtimeout" (int)
 29 The time in seconds before the RM will execute error response.

30 **PMIX_EVENT_SILENT_TERMINATION** "pmix.evsilentterm" (bool)
 31 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.

1
PMIx v2.0

Format

C

```
2 void
3 PMIx_Deregister_event_handler(size_t evhdlr_ref,
4                               pmix_op_cbfunc_t cbfunc,
5                               void *cbdata);
```

C

- 6 **IN evhdlr_ref**
7 Event handler ID returned by registration (**size_t**)
- 8 **IN cbfunc**
9 Callback function to be executed upon completion of operation **pmix_op_cbfunc_t**
10 (function reference)
- 11 **IN cbdata**
12 Data to be passed to the cbfunc callback function (memory reference)

13 Returns one of the following:

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

21 If the provided cbfunc is called to confirm removal of the designated handler, the returned status
22 code will be one of the following:

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

26 Description

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

30 8.1.3 PMIx_Notify_event

31 Summary

32 Report an event for notification via any registered event handler.

1
PMIx v2.0

Format

C

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

C

- 8 **IN status**
9 Status code of the event ([pmix_status_t](#))
- 10 **IN source**
11 Pointer to a [pmix_proc_t](#) identifying the original reporter of the event (handle)
- 12 **IN range**
13 Range across which this notification shall be delivered ([pmix_data_range_t](#))
- 14 **IN info**
15 Array of [pmix_info_t](#) structures containing any further info provided by the originator of
16 the event (array of handles)
- 17 **IN ninfo**
18 Number of elements in the *info* array ([size_t](#))
- 19 **IN cbfunc**
20 Callback function to be executed upon completion of operation [pmix_op_cbfunc_t](#)
21 (function reference)
- 22 **IN cbdata**
23 Data to be passed to the cbfunc callback function (memory reference)

24 Returns one of the following:

- 25 **PMIX_SUCCESS** The notification request is valid and is being processed. The callback function
26 will be called when the process-local operation is complete and will provide the resulting
27 status of that operation. Note that this does *not* reflect the success or failure of delivering the
28 event to any recipients. The callback function must not be executed prior to returning from the
29 API.
- 30 **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
31 returned *success* - the *cbfunc* will *not* be called
- 32 **PMIX_ERR_BAD_PARAM** The request contains at least one incorrect entry that prevents it from
33 being processed. The callback function will *not* be called.
- 34 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification,
35 or in the case of a PMIx server calling the API, the range extended beyond the local node and
36 the host SMS environment does not support event notification. The callback function will *not*
37 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.evrage" (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

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

8 9.1 Support Macros

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

10 9.1.1 PMIX_DATA_BUFFER_CREATE

11 **Summary**

12 Allocate memory for a `pmix_data_buffer_t` object and initialize it

13 **Format**

PMIx v2.0

```
13    ▼ _____ C _____ ▼  
14    PMIX_DATA_BUFFER_CREATE (buffer);  
14    ▲ _____ C _____ ▲
```

15 **OUT** `buffer`

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

17 **Description**

18 This macro uses *calloc* to allocate memory for the buffer and initialize all fields in it

19 9.1.2 PMIX_DATA_BUFFER_RELEASE

20 **Summary**

21 Free a `pmix_data_buffer_t` object and the data it contains

1 **Format**

PMIx v2.0 ▼ **C** _____ ▼

2 **PMIX_DATA_BUFFER_RELEASE(buffer);**

▲ _____ **C** _____ ▲

3 **IN buffer**

4 Pointer to the **pmix_data_buffer_t** to be released (handle)

5 **Description**

6 Free's the data contained in the buffer, and then free's the buffer itself

7 **9.1.3 PMIX_DATA_BUFFER_CONSTRUCT**

8 **Summary**

9 Initialize a statically declared **pmix_data_buffer_t** object

10 **Format**

PMIx v2.0 ▼ **C** _____ ▼

11 **PMIX_DATA_BUFFER_CONSTRUCT(buffer);**

▲ _____ **C** _____ ▲

12 **IN buffer**

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

14 **Description**

15 Initialize a pre-allocated buffer object

16 **9.1.4 PMIX_DATA_BUFFER_DESTRUCT**

17 **Summary**

18 Release the data contained in a **pmix_data_buffer_t** object

19 **Format**

PMIx v2.0 ▼ **C** _____ ▼

20 **PMIX_DATA_BUFFER_DESTRUCT(buffer);**

▲ _____ **C** _____ ▲

21 **IN buffer**

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

23 **Description**

24 Free's the data contained in a **pmix_data_buffer_t** object

1 9.1.5 PMIX_DATA_BUFFER_LOAD

2 Summary

3 Load a blob into a `pmix_data_buffer_t` object

4 Format

PMIx v2.0

C

```
5 PMIX_DATA_BUFFER_LOAD(buffer, data, size);
```

C

6 **IN** `buffer`

7 Pointer to a pre-allocated `pmix_data_buffer_t` (handle)

8 **IN** `data`

9 Pointer to a blob (`char*`)

10 **IN** `size`

11 Number of bytes in the blob `size_t`

12 Description

13 Load the given data into the provided `pmix_data_buffer_t` object, usually done in
14 preparation for unpacking the provided data. Note that the data is *not* copied into the buffer - thus,
15 the blob must not be released until after operations on the buffer have completed.

16 9.1.6 PMIX_DATA_BUFFER_UNLOAD

17 Summary

18 Unload the data from a `pmix_data_buffer_t` object

19 Format

PMIx v2.0

C

```
20 PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

C

21 **IN** `buffer`

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

23 **OUT** `data`

24 Variable to be assigned the pointer to the extracted blob (`void*`)

25 **OUT** `size`

26 Variable to be assigned the number of bytes in the blob `size_t`

27 Description

28 Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to
29 the provided variables, usually done to transmit the blob to a remote process for unpacking. The
30 buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release -
31 thus, the user is responsible for releasing the blob when done with it.

1 9.2 General Routines

2 The following routines are provided to support internode transfers in heterogeneous environments.

3 9.2.1 PMIx_Data_pack

4 Summary

5 Pack one or more values of a specified type into a buffer, usually for transmission to another process

6 Format

PMIx v2.0

C

```
7 pmix_status_t
8 PMIx_Data_pack(const pmix_proc_t *target,
9               pmix_data_buffer_t *buffer,
10              void *src, int32_t num_vals,
11              pmix_data_type_t type);
```

C

12 IN target

13 Pointer to a [pmix_proc_t](#) containing the nspace/rank of the process that will be unpacking
14 the final buffer. A NULL value may be used to indicate that the target is based on the same
15 PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

16 IN buffer

17 Pointer to a [pmix_data_buffer_t](#) where the packed data is to be stored (handle)

18 IN src

19 Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the
20 caller must pass the address of the pointer to the string as the (void*). This allows the caller to
21 pass multiple strings in a single call. (memory reference)

22 IN num_vals

23 Number of elements pointed to by the *src* pointer. A string value is counted as a single value
24 regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,
25 string arrays) should be contiguous, although the data pointed to need not be contiguous
26 across array entries. ([int32_t](#))

27 IN type

28 The type of the data to be packed ([pmix_data_type_t](#))

29 Returns one of the following:

30 [PMIX_SUCCESS](#) The data has been packed as requested

31 [PMIX_ERR_NOT_SUPPORTED](#) The PMIx implementation does not support this function.

32 [PMIX_ERR_BAD_PARAM](#) The provided buffer or src is **NULL**

33 [PMIX_ERR_UNKNOWN_DATA_TYPE](#) The specified data type is not known to this
34 implementation

35 [PMIX_ERR_OUT_OF_RESOURCE](#) Not enough memory to support the operation

36 [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.

9.2.2 PMIx_Data_unpack

Summary

Unpack values from a `pmix_data_buffer_t`

Format

PMIx v2.0

`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);
```

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***)

1 **INOUT** `max_num_values`

2 The number of values to be unpacked — upon completion, the parameter will be set to the
3 actual number of values unpacked. In most cases, this should match the maximum number
4 provided in the parameters — but in no case will it exceed the value of this parameter. Note
5 that unpacking fewer values than are actually available will leave the buffer in an unpackable
6 state — the function will return an error code to warn of this condition. (`int32_t`)

7 **IN** `type`

8 The type of the data to be unpacked — must be one of the PMIx defined data types (
9 `pmix_data_type_t`)

10 Returns one of the following:

11 `PMIX_SUCCESS` The data has been unpacked as requested

12 `PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

13 `PMIX_ERR_BAD_PARAM` The provided buffer or dest is **NULL**

14 `PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this
15 implementation

16 `PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation

17 `PMIX_ERROR` General error

18 **Description**

19 The unpack function unpacks the next value (or values) of a specified type from the given buffer.

20 The buffer must have already been initialized via an `PMIX_DATA_BUFFER_CREATE` or
21 `PMIX_DATA_BUFFER_CONSTRUCT` call (and assumedly filled with some data) — otherwise,
22 the `unpack_value` function will return an error. Providing an unsupported type flag will likewise be
23 reported as an error, as will specifying a data type that *does not* match the type of the next item in
24 the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an
25 error.

26 NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper
27 variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte
28 field in a string array that so happens to have a value that matches the specified data type flag).
29 Therefore, the data type error check is *not* completely safe.

30 Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer.
31 It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the
32 `unpack_ptr`.

33 Warning: The caller is responsible for providing adequate memory storage for the requested data.
34 The user must provide a parameter indicating the maximum number of values that can be unpacked
35 into the allocated memory. If more values exist in the buffer than can fit into the memory storage,
36 then the function will unpack what it can fit into that location and return an error code indicating
37 that the buffer was only partially unpacked.

38 Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may
39 lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with
40 heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than

1 can be handled by the recipient will return an error code generated upon unpacking — these errors
2 cannot be detected during packing.

3 The namespace of the process that packed the buffer is used solely to resolve any data type
4 differences between PMIx versions. The packer must, therefore, be known to the user prior to
5 calling the pack function so that the PMIx library is aware of the version the packer is using. Note
6 that all processes in a given namespace are *required* to use the same PMIx version — thus, the
7 caller must only know at least one process from the packer’s namespace.

8 9.2.3 PMIx_Data_copy

9 Summary

10 Copy a data value from one location to another.

11 Format

PMIx v2.0

```
12 pmix_status_t  
13 PMIx_Data_copy(void **dest, void *src,  
14                pmix_data_type_t type);
```

15 IN dest

16 The address of a pointer into which the address of the resulting data is to be stored. (**void****)

17 IN src

18 A pointer to the memory location from which the data is to be copied (handle)

19 IN type

20 The type of the data to be copied — must be one of the PMIx defined data types. (
21 [pmix_data_type_t](#))

22 Returns one of the following:

23 [PMIX_SUCCESS](#) The data has been copied as requested

24 [PMIX_ERR_NOT_SUPPORTED](#) The PMIx implementation does not support this function.

25 [PMIX_ERR_BAD_PARAM](#) The provided src or dest is **NULL**

26 [PMIX_ERR_UNKNOWN_DATA_TYPE](#) The specified data type is not known to this
27 implementation

28 [PMIX_ERR_OUT_OF_RESOURCE](#) Not enough memory to support the operation

29 [PMIX_ERROR](#) General error

30 Description

31 Since registered data types can be complex structures, the system needs some way to know how to
32 copy the data from one location to another (e.g., for storage in the registry). This function, which
33 can call other copy functions to build up complex data types, defines the method for making a copy
34 of the specified data type.

1 9.2.4 PMIx_Data_print

2 Summary

3 Pretty-print a data value.

4 Format

PMIx v2.0

C

5 `pmix_status_t`

6 `PMIx_Data_print(char **output, char *prefix,`
7 `void *src, pmix_data_type_t type);`

C

8 **IN** `output`

9 The address of a pointer into which the address of the resulting output is to be stored.

10 (`char**`)

11 **IN** `prefix`

12 String to be prepended to the resulting output (`char*`)

13 **IN** `src`

14 A pointer to the memory location of the data value to be printed (handle)

15 **IN** `type`

16 The type of the data value to be printed — must be one of the PMIx defined data types. (

17 `pmix_data_type_t`)

18 Returns one of the following:

19 `PMIX_SUCCESS` The data has been printed as requested

20 `PMIX_ERR_BAD_PARAM` The provided data type is not recognized.

21 `PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

22 Description

23 Since registered data types can be complex structures, the system needs some way to know how to
24 print them (i.e., convert them to a string representation). Primarily for debug purposes.

25 9.2.5 PMIx_Data_copy_payload

26 Summary

27 Copy a payload from one buffer to another

1
PMIx v2.0

Format

C

```
2 pmix_status_t  
3 PMIx_Data_copy_payload(pmix_data_buffer_t *dest,  
4 pmix_data_buffer_t *src);
```

C

5 **IN** `dest`
6 Pointer to the destination `pmix_data_buffer_t` (handle)
7 **IN** `src`
8 Pointer to the source `pmix_data_buffer_t` (handle)

9 Returns one of the following:

10 **PMIX_SUCCESS** The data has been copied as requested
11 **PMIX_ERR_BAD_PARAM** The src and dest `pmix_data_buffer_t` types do not match
12 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

Description

13 This function will append a copy of the payload in one buffer into another buffer. Note that this is
14 *not* a destructive procedure — the source buffer’s payload will remain intact, as will any pre-existing
15 payload in the destination’s buffer. Only the unpacked portion of the source payload will be copied.
16

CHAPTER 10

Security

1 PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes.
2 *Client* processes (i.e., processes started by the host environment) must be preregistered with the
3 PMIx server library via the `PMIx_server_register_client` API before they are spawned.
4 This API requires that you pass the expected uid/gid of the client process.

5 When the client attempts to connect to the PMIx server, the server uses available standard
6 Operating System (OS) methods to determine the effective uid/gid of the process requesting the
7 connection. PMIx implementations shall not rely on any values reported by the client process itself
8 as that would be unsafe. The effective uid/gid reported by the OS is compared to the values
9 provided by the host during registration - if they don't match, the PMIx server is required to drop
10 the connection request. This ensures that the PMIx server does not allow connection from a client
11 that doesn't at least meet some minimal security requirement.

12 Once the requesting client passes the initial test, the PMIx server can, at the choice of the
13 implementor, perform additional security checks. This may involve a variety of methods such as
14 exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server
15 reports the client connection request to the host via the
16 `pmix_server_client_connected_fn_t` interface. The host may then perform any
17 additional checks and operations before responding with either `PMIX_SUCCESS` to indicate that
18 the connection is approved, or a PMIx error constant indicating that the connection request is
19 refused. In this latter case, the PMIx server is required to drop the connection.

20 Tools started by the host environment are classed as a subgroup of client processes and follow the
21 client process procedure. However, tools that are not started by the host environment must be
22 handled differently as registration information is not available prior to the connection request. In
23 these cases, the PMIx server library is required to use available standard OS methods to get the
24 effective uid/gid and report them upwards as part of invoking the
25 `pmix_server_tool_connection_fn_t` interface, deferring initial security screening to
26 the host. It is recognized that this may represent a security risk - for this reason, PMIx server
27 libraries must not enable tool connections by default. Instead, the host has to explicitly enable them
28 via the `PMIX_SERVER_TOOL_SUPPORT` attribute, thus recognizing the associated risk. Once
29 the host has completed its authentication procedure, it again informs the PMIx server of the result.

30 Applications and tools often interact with the host environment in ways that require security beyond
31 just verifying the user's identity - e.g., access to that user's relevant authorizations. This is
32 particularly important when tools connect directly to a system-level PMIx server that may be
33 operating at a privileged level. A variety of system management software packages provide
34 authorization services, but the lack of standardized interfaces makes portability problematic.

1 This section defines two PMIx client-side APIs for this purpose. These are most likely to be used
2 by user-space applications/tools, but are not restricted to that realm.

3 10.1 Obtaining Credentials

4 The API for obtaining a credential is a non-blocking operation since the host environment may have
5 to contact a remote credential service. The definition takes into account the potential that the
6 returned credential could be sent via some mechanism to another application that resides in an
7 environment using a different security mechanism. Thus, provision is made for the system to return
8 additional information (e.g., the identity of the issuing agent) outside of the credential itself and
9 visible to the application.

10 10.1.1 PMIx_Get_credential

11 Summary

12 Request a credential from the PMIx server library or the host environment

13 Format

PMIx v3.0

```
14 pmix_status_t  
15 PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,  
16 pmix_credential_cbfunc_t cbfunc, void *cbdata)  
17 C
```

- 17 **IN info**
18 Array of **pmix_info_t** structures (array of handles)
- 19 **IN ninfo**
20 Number of elements in the *info* array (**size_t**)
- 21 **IN cbfunc**
22 Callback function to return credential (**pmix_credential_cbfunc_t** function
23 reference)
- 24 **IN cbdata**
25 Data to be passed to the callback function (memory reference)

26 Returns one of the following:

- 27 • **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server -
28 result will be returned in the provided *cbfunc*
- 29 • a PMIx error constant indicating either an error in the input or that the request is unsupported -
30 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

1 10.2 Validating Credentials

2 The API for validating a credential is a non-blocking operation since the host environment may
3 have to contact a remote credential service. Provision is made for the system to return additional
4 information regarding possible authorization limitations beyond simple authentication.

5 10.2.1 PMIx_Validate_credential

6 Summary

7 Request validation of a credential by the PMIx server library or the host environment

8 Format

PMIx v3.0

C

9 `pmix_status_t`

```
10 PMIx_Validate_credential(const pmix_byte_object_t *cred,  
11                          const pmix_info_t info[], size_t ninfo,  
12                          pmix_validation_cbfunc_t cbfunc,  
13                          void *cbdata)
```

C

14 **IN cred**

15 Pointer to `pmix_byte_object_t` containing the credential (handle)

16 **IN info**

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

18 **IN ninfo**

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

20 **IN cbfunc**

21 Callback function to return result (`pmix_validation_cbfunc_t` function reference)

22 **IN cbdata**

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

24 Returns one of the following:

- 25 • **PMIX_SUCCESS** , indicating that the request has been communicated to the local PMIx server -
26 result will be returned in the provided *cbfunc*
- 27 • a PMIx error constant indicating either an error in the input or that the request is unsupported -
28 the *cbfunc* will *not* be called

Required Attributes

1 PMIx libraries that choose not to support this operation *must* return
2 **PMIX_ERR_NOT_SUPPORTED** when the function is called.

3 There are no required attributes for this API. Note that implementations may choose to internally
4 execute integration for some security environments (e.g., directly contacting a *munge* server).

5 Implementations that support the operation but cannot directly process the client's request must
6 pass any attributes that are provided by the client to the host environment for processing. In
7 addition, the following attributes are required to be included in the *info* array passed from the PMIx
8 library to the host environment:

9 **PMIX_USERID** "pmix.euid" (uint32_t)
10 Effective user id.

11 **PMIX_GRPID** "pmix.egid" (uint32_t)
12 Effective group id.

Optional Attributes

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

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

Advice to PMIx library implementers

18 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
19 environment due to race condition considerations between completion of the operation versus
20 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
21 directly in the PMIx server library must take care to resolve the race condition and should avoid
22 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
23 created.

Description

24 Request validation of a credential by the PMIx server library or the host environment.
25

CHAPTER 11

Server-Specific Interfaces

1 The RM daemon that hosts the PMIx server library interacts with that library in two distinct
2 manners. First, PMIx provides a set of APIs by which the host can request specific services from its
3 library. This includes generating regular expressions, registering information to be passed to client
4 processes, and requesting information on behalf of a remote process. Note that the host always has
5 access to all PMIx client APIs - the functions listed below are in addition to those available to a
6 PMIx client.

7 Second, the host can provide a set of callback functions by which the PMIx server library can pass
8 requests upward for servicing by the host. These include notifications of client connection and
9 finalize, as well as requests by clients for information and/or services that the PMIx server library
10 does not itself provide.

11.1 Server Support Functions

12 The following APIs allow the RM daemon that hosts the PMIx server library to request specific
13 services from the PMIx library.

11.1.1 PMIx_generate_regex

Summary

Generate a regular expression representation of the input string.

Format

PMIx v1.0

C

`pmix_status_t`

`PMIx_generate_regex(const char *input, char **regex)`

C

IN `input`

String to process (string)

OUT `regex`

Regular expression representation of *input* (string)

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

Description

Given a comma-separated list of *input* values, generate a regular expression that can be passed down to the PMIx client for parsing. The order of the individual values in the *input* string is preserved in the resulting *regex* string. The caller is responsible for free'ing the resulting string.

If values have leading zero's, then that is preserved, as are prefix and suffix strings. For example, an input string of

`“odin009.org,odin010.org,odin011.org,odin012.org,odin[102-107].org”`
will return a regular expression of `“pmix:odin[009-012,102-107].org”`

Advice to users

The returned regular expression will have a `“pmix:”` at the beginning of the string. This informs the PMIx parser that the string was produced using the PRI's regular expression generator, and thus that same plugin should be used for parsing the string

11.1.2 PMIx_generate_ppn

Summary

Generate a regular expression representation of the input string.

Format

PMIx v1.0

```
pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)
```

IN **input**

String to process (string)

OUT **regex**

Regular expression representation of *input* (string)

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

Description

The input is expected to consist of a semicolon-separated list of ranges representing the ranks of processes on each node of the job. Thus, an input of "1-4;2-5;8,10,11,12;6,7,9" would generate a regex of "pmix:2x(3);8,10-12;6-7,9"

Advice to users

The returned regular expression will have a `“pmix:”` at the beginning of the string. This informs the PMIx parser that the string was produced using the PRI's regular expression generator, and thus that same plugin should be used for parsing the string

1 11.1.3 PMIx_server_register_namespace

2 Summary

3 Setup the data about a particular namespace.

4 Format

PMIx v1.0

C

5 `pmix_status_t`

```
6 PMIx_server_register_namespace(const pmix_namespace_t nspace,  
7                                int nlocalprocs,  
8                                pmix_info_t info[], size_t ninfo,  
9                                pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

10 **IN** `nspace`

11 Character array of maximum size `PMIX_MAX_NSLEN` containing the namespace identifier
12 (string)

13 **IN** `nlocalprocs`

14 number of local processes (integer)

15 **IN** `info`

16 Array of info structures (array of handles)

17 **IN** `ninfo`

18 Number of elements in the `info` array (integer)

19 **IN** `cbfunc`

20 Callback function `pmix_op_cbfunc_t` (function reference)

21 **IN** `cbdata`

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

23 Returns one of the following:

- 24 • `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result
25 will be returned in the provided `cbfunc`. Note that the library must not invoke the callback
26 function prior to returning from the API.
- 27 • `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and
28 returned `success` - the `cbfunc` will not be called
- 29 • a PMIx error constant indicating either an error in the input or that the request was immediately
30 processed and failed - the `cbfunc` will not be called

Required Attributes

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

32 **PMIX_REGISTER_NODATA** "`pmix.reg.nodata`" (bool)

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

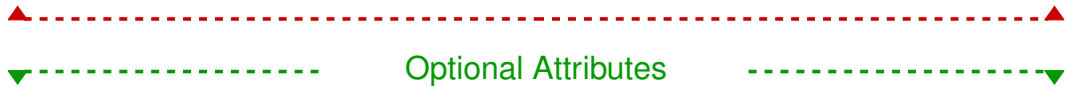
1
2 Host environments are required to provide the following attributes:

- 3 • for the session containing the given namespace:
 - 4 – **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)
5 Number of allocated slots in a session - each slot may or may not be occupied by an
6 executing process. Note that this attribute is the equivalent to the combination of
7 **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it
8 is included in the Standard for historical reasons.
- 9 • for the given namespace:
 - 10 – **PMIX_JOBID** "pmix.jobid" (char*)
11 Job identifier assigned by the scheduler.
 - 12 – **PMIX_JOB_SIZE** "pmix.job.size" (uint32_t)
13 Total number of processes in this job across all contained applications. Note that this
14 value can be different from **PMIX_MAX_PROCS** . For example, users may choose to
15 subdivide an allocation (running several jobs in parallel within it), and dynamic
16 programming models may support adding and removing processes from a running **job**
17 on-the-fly. In the latter case, PMIx events must be used to notify processes within the
18 job that the job size has changed.
 - 19 – **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
20 Maximum number of processes that can be executed in this context (session,
21 namespace, application, or node). Typically, this is a constraint imposed by a scheduler
22 or by user settings in a hostfile or other resource description.
 - 23 – **PMIX_NODE_MAP** "pmix.nmap" (char*)
24 Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.
 - 25 – **PMIX_PROC_MAP** "pmix.pmap" (char*)
26 Regular expression describing processes on each node - see 11.1.3.1 for an explanation
27 of its generation.
- 28 • for its own node:
 - 29 – **PMIX_LOCAL_SIZE** "pmix.local.size" (uint32_t)
30 Number of processes in this job or application on this node.
 - 31 – **PMIX_LOCAL_PEERS** "pmix.lpeers" (char*)
32 Comma-delimited list of ranks on this node within the specified namespace - referenced
33 using **PMIX_RANK_WILDCARD** .
 - 34 – **PMIX_LOCAL_CPUSSETS** "pmix.lcpus" (char*)
35 Colon-delimited cpusets of local peers within the specified namespace - referenced
36 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.



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_NAMESPACE** "pmix.srv.namespace" (char*)

1 Name of the namespace to use for this PMIx server.

- 2 - **PMIX_SERVER_RANK** "pmix.srv.rank" (pmix_rank_t)
3 Rank of this PMIx server
- 4 - **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
5 Starting global rank of this job - referenced using [PMIX_RANK_WILDCARD](#) .
- 6 - **PMIX_ALLOCATED_NODELIST** "pmix.alist" (char*)
7 Comma-delimited list of all nodes in this allocation regardless of whether or not they
8 currently host processes - referenced using [PMIX_RANK_WILDCARD](#) .
- 9 - **PMIX_JOB_NUM_APPS** "pmix.job.napps" (uint32_t)
10 Number of applications in this job.
- 11 - **PMIX_MAPBY** "pmix.mapby" (char*)
12 Process mapping policy - when accessed using [PMIx_Get](#) , use the
13 [PMIX_RANK_WILDCARD](#) value for the rank to discover the mapping policy used for
14 the provided namespace
- 15 - **PMIX_RANKBY** "pmix.rankby" (char*)
16 Process ranking policy - when accessed using [PMIx_Get](#) , use the
17 [PMIX_RANK_WILDCARD](#) value for the rank to discover the ranking algorithm used
18 for the provided namespace
- 19 - **PMIX_BINDTO** "pmix.bindto" (char*)
20 Process binding policy - when accessed using [PMIx_Get](#) , use the
21 [PMIX_RANK_WILDCARD](#) value for the rank to discover the binding policy used for
22 the provided namespace
- 23 • for its own node:
 - 24 - **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)
25 Total available physical memory on this node.
 - 26 - **PMIX_HWLOC_XML_V1** "pmix.hwlocxml1" (char*)
27 XML representation of local topology using HWLOC's v1.x format.
 - 28 - **PMIX_HWLOC_XML_V2** "pmix.hwlocxml2" (char*)
29 XML representation of local topology using HWLOC's v2.x format.
 - 30 - **PMIX_LOCALLDR** "pmix.lldr" (pmix_rank_t)
31 Lowest rank on this node within this job - referenced using [PMIX_RANK_WILDCARD](#) .
 - 32
 - 33 - **PMIX_NODE_SIZE** "pmix.node.size" (uint32_t)
34 Number of processes across all jobs on this node.
 - 35 - **PMIX_LOCAL_PROCS** "pmix.lprocs" (pmix_proc_t array)

1 Array of `pmix_proc_t` of all processes on the specified node - referenced using
2 **`PMIX_RANK_WILDCARD`** .

- 3 • for each process in the given namespace:
 - 4 – **`PMIX_PROCID`** "`pmix.procid`" (`pmix_proc_t`)
5 Process identifier
- 6 – **`PMIX_GLOBAL_RANK`** "`pmix.grank`" (`pmix_rank_t`)
7 Process rank spanning across all jobs in this session.- 8 – **`PMIX_HOSTNAME`** "`pmix.hname`" (`char*`)
9 Name of the host where the specified process is running.

10 Attributes not directly provided by the host environment may be derived by the PMIx server library
11 from other required information and included in the data made available to the server library's
12 clients.



13 **Description**

14 Pass job-related information to the PMIx server library for distribution to local client processes.



Advice to PMIx server hosts

15 Host environments are required to execute this operation prior to starting any local application
16 process within the given namespace.

17 The PMIx server must register all namespaces that will participate in collective operations with
18 local processes. This means that the server must register a namespace even if it will not host any
19 local processes from within that namespace if any local process of another namespace might at
20 some point perform an operation involving one or more processes from the new namespace. This is
21 necessary so that the collective operation can identify the participants and know when it is locally
22 complete.

23 The caller must also provide the number of local processes that will be launched within this
24 namespace. This is required for the PMIx server library to correctly handle collectives as a
25 collective operation call can occur before all the local processes have been started.



Advice to users

26 The number of local processes for any given namespace is generally fixed at the time of application
27 launch. Calls to **`PMIx_Spawn`** result in processes launched in their own namespace, not that of
28 their parent. However, it is possible for processes to *migrate* to another node via a call to
29 **`PMIx_Job_control_nb`** , thus resulting in a change to the number of local processes on both
30 the initial node and the node to which the process moved. It is therefore critical that applications
31 not migrate processes without first ensuring that PMIx-based collective operations are not in
32 progress, and that no such operations be initiated until process migration has completed.



1 11.1.3.1 Assembling the registration information

2 The following description is not intended to represent the actual layout of information in a given
3 PMIx library. Instead, it describes how information provided in the *info* parameter of the
4 `PMIx_server_register_nspace` shall be organized for proper processing by a PMIx server
5 library. The ordering of the various information elements is arbitrary - they are presented in a
6 top-down hierarchical form solely for clarity in reading.

▼ Advice to PMIx server hosts ▼

7 Creating the *info* array of data requires knowing in advance the number of elements required for the
8 array. This can be difficult to compute and somewhat fragile in practice. One method for resolving
9 the problem is to create a linked list of objects, each containing a single `pmix_info_t` structure.
10 Allocation and manipulation of the list can then be accomplished using existing standard methods.
11 Upon completion, the final *info* array can be allocated based on the number of elements on the list,
12 and then the values in the list object `pmix_info_t` structures transferred to the corresponding
13 array element utilizing the `PMIX_INFO_XFER` macro.

14 A common building block used in several areas is the construction of a regular expression
15 identifying the nodes involved in that area - e.g., the nodes in a `session` or `job`. PMIx provides
16 several tools to facilitate this operation, beginning by constructing an argv-like array of node
17 names. This array is then passed to the `PMIx_generate_regex` function to create a regular
18 expression parseable by the PMIx server library, as shown below:

▼ C ▼

```
19 char **nodes = NULL;  
20 char *nodelist;  
21 char *regex;  
22 size_t n;  
23 pmix_status_t rc;  
24 pmix_info_t info;  
25  
26 /* loop over an array of nodes, adding each  
27 * name to the array */  
28 for (n=0; n < num_nodes; n++)  
29     /* filter the nodes to ignore those not included  
30     * in the target range (session, job, etc.). In  
31     * this example, all nodes are accepted */  
32     PMIX_ARGV_APPEND(&nodes, node[n]->name);  
33  
34  
35 /* join into a comma-delimited string */  
36 nodelist = PMIX_ARGV_JOIN(nodes, ',');  
37
```

```

1      /* release the array */
2      PMIX_ARGV_FREE(nodes);
3
4      /* generate regex */
5      rc = PMIx_generate_regex(nodelist, &regex);
6
7      /* release list */
8      free(nodelist);
9
10     /* pass the regex as the value to the PMIX_NODE_MAP key */
11     PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
12     /* release the regex */
13     free(regex);
14

```



15 Changing the filter criteria allows the construction of node maps for any level of information.

16 A similar method is used to construct the map of processes on each node from the namespace being
17 registered. This may be done for each information level of interest (e.g., to identify the process map
18 for the entire **job** or for each **application** in the job) by changing the search criteria. An
19 example is shown below for the case of creating the process map for a **job** :



```

20     char **ndppn;
21     char rank[30];
22     char **ppnarray = NULL;
23     char *ppn;
24     char *localranks;
25     char *regex;
26     size_t n, m;
27     pmix_status_t rc;
28     pmix_info_t info;
29
30     /* loop over an array of nodes */
31     for (n=0; n < num_nodes; n++)
32         /* for each node, construct an array of ranks on that node */
33         ndppn = NULL;
34         for (m=0; m < node[n]->num_procs; m++)
35             /* ignore processes that are not part of the target job */
36             if (!PMIX_CHECK_NAMESPACE(targetjob, node[n]->proc[m].namespace))
37                 continue;
38
39             snprintf(rank, 30, "%d", node[n]->proc[m].rank);
40             PMIX_ARGV_APPEND(&ndppn, rank);

```

```

1
2     /* convert the array into a comma-delimited string of ranks */
3     localranks = PMIX_ARGV_JOIN(ndppn, ',');
4     /* release the local array */
5     PMIX_ARGV_FREE(ndppn);
6     /* add this node's contribution to the overall array */
7     PMIX_ARGV_APPEND(&ppnarray, localranks);
8     /* release the local list */
9     free(localranks);
10
11
12     /* join into a semicolon-delimited string */
13     ppn = PMIX_ARGV_JOIN(ppnarray, ';');
14
15     /* release the array */
16     PMIX_ARGV_FREE(ppnarray);
17
18     /* generate ppn regex */
19     rc = PMIx_generate_ppn(ppn, &regex);
20
21     /* release list */
22     free(ppn);
23
24     /* pass the regex as the value to the PMIX_PROC_MAP key */
25     PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
26     /* release the regex */
27     free(regex);
28

```

C

29 Note that the [PMIX_NODE_MAP](#) and [PMIX_PROC_MAP](#) attributes are linked in that the order of
 30 entries in the process map must match the ordering of nodes in the node map - i.e., there is no
 31 provision in the PMIx process map regular expression generator/parser pair supporting an
 32 out-of-order node or a node that has no corresponding process map entry (e.g., a node with no
 33 processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

- 34 • Session-level information includes all session-specific values. In many cases, only two values (
 35 [PMIX_SESSION_ID](#) and [PMIX_UNIV_SIZE](#)) are included in the registration array. Since
 36 both of these values are session-specific, they can be specified independently - i.e., in their own
 37 [pmix_info_t](#) elements of the *info* array. Alternatively, they can be provided as a
 38 [pmix_data_array_t](#) array of [pmix_info_t](#) using the [PMIX_SESSION_INFO_ARRAY](#)
 39 attribute and identified by including the [PMIX_SESSION_ID](#) attribute in the array - this is
 40 required in cases where non-specific attributes (e.g., [PMIX_NUM_NODES](#) or [PMIX_NODE_MAP](#)

1) are passed to describe aspects of the session. Note that the node map can include nodes not
 2 used by the job being registered as no corresponding process map is specified.

3 The *info* array at this point might look like (where the labels identify the corresponding attribute
 4 - e.g., “Session ID” corresponds to the **PMIX_SESSION_ID** attribute):

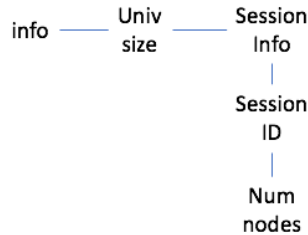


Figure 11.1.: Session-level information elements

5 • Job-level information includes all job-specific values such as **PMIX_JOB_SIZE** ,
 6 **PMIX_JOB_NUM_APPS** , and **PMIX_JOBID** . Since each invocation of
 7 **PMIx_server_register_namespace** describes a single *job* , job-specific values can be
 8 specified independently - i.e., in their own **pmix_info_t** elements of the *info* array.
 9 Alternatively, they can be provided as a **pmix_data_array_t** array of **pmix_info_t**
 10 identified by the **PMIX_JOB_INFO_ARRAY** attribute - this is required in cases where
 11 non-specific attributes (e.g., **PMIX_NODE_MAP**) are passed to describe aspects of the job. Note
 12 that since the invocation only involves a single namespace, there is no need to include the
 13 **PMIX_NAMESPACE** attribute in the array.

14 Upon conclusion of this step, the *info* array might look like:

15 Note that in this example, **PMIX_NUM_NODES** is not required as that information is contained
 16 in the **PMIX_NODE_MAP** attribute. Similarly, **PMIX_JOB_SIZE** is not technically required as
 17 that information is contained in the **PMIX_PROC_MAP** when combined with the corresponding
 18 node map - however, there is no issue with including the job size as a separate entry.

19 The example also illustrates the hierarchical use of the **PMIX_NODE_INFO_ARRAY** attribute.
 20 In this case, we have chosen to pass several job-related values for each node - since those values
 21 are non-unique across the job, they must be passed in a node-info container. Note that the choice
 22 of what information to pass into the PMIx server library versus what information to derive from
 23 other values at time of request is left to the host environment. PMIx implementors in turn may, if
 24 they choose, pre-parse registration data to create expanded views (thus enabling faster response
 25 to requests at the expense of memory footprint) or to compress views into tighter representations
 26 (thus trading minimized footprint for longer response times).

27 • Application-level information includes all application-specific values such as **PMIX_APP_SIZE**
 28 and **PMIX_APPLDR** . If the *job* contains only a single *application* , then the
 29 application-specific values can be specified independently - i.e., in their own **pmix_info_t**

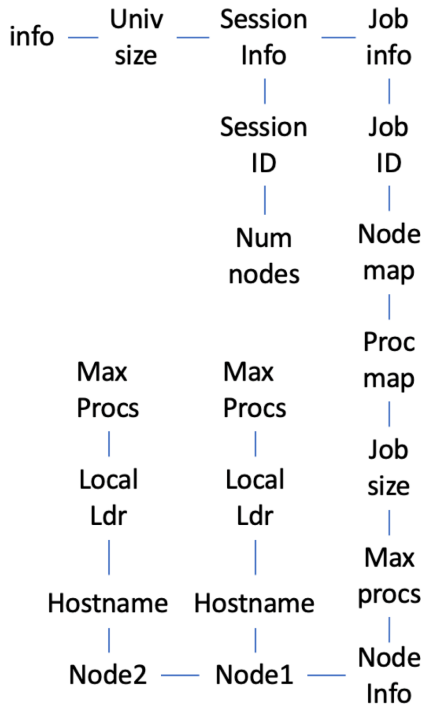


Figure 11.2.: Job-level information elements

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 identified by including the `PMIX_APPNUM` attribute in the array. Use of the array format is must in cases where non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the `PMIX_APP_INFO_ARRAY` format, each identified by its `PMIX_APPNUM` value.

Upon conclusion of this step, the *info* array might look like that shown in 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,

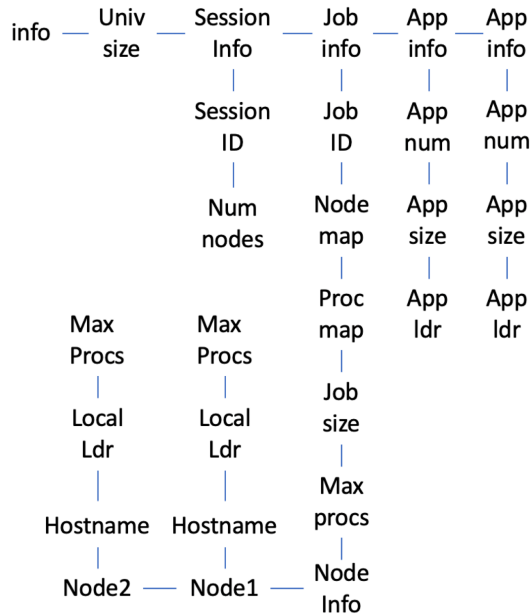


Figure 11.3.: Application-level information elements

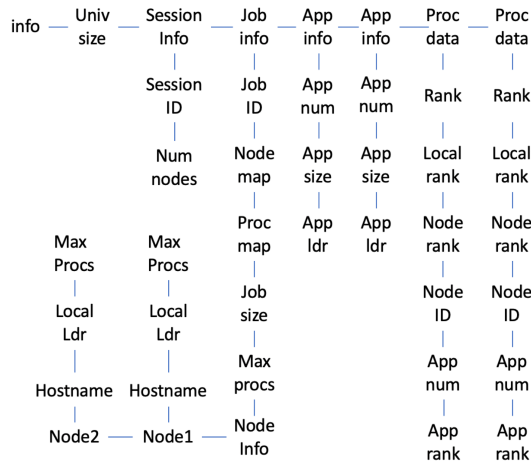


Figure 11.4.: Process-level information elements

- 1 the values included in this level are unique to it and can be specified independently - i.e., in their
- 2 own `pmix_info_t` elements of the `info` array. Alternatively, they can be provided as a
- 3 `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_NODE_INFO_ARRAY`

1 attribute - this is required in cases where non-specific attributes are passed to describe aspects of
2 the node, or where values for multiple nodes are being provided.

3 The node-level information requires two elements that must be constructed in a manner similar to
4 that used for the node map. The **PMIX_LOCAL_PEERS** value is computed based on the
5 processes on the local node, filtered to select those from the job being registered, as shown below
6 using the tools provided by PMIx:

C

```
7 char **ndppn = NULL;
8 char rank[30];
9 char *localranks;
10 size_t m;
11 pmix_info_t info;
12
13 for (m=0; m < mynode->num_procs; m++)
14     /* ignore processes that are not part of the target job */
15     if (!PMIX_CHECK_NAMESPACE(targetjob, mynode->proc[m].nspace))
16         continue;
17
18     snprintf(rank, 30, "%d", mynode->proc[m].rank);
19     PMIX_ARGV_APPEND(&ndppn, rank);
20
21     /* convert the array into a comma-delimited string of ranks */
22     localranks = PMIX_ARGV_JOIN(ndppn, ',');
23     /* release the local array */
24     PMIX_ARGV_FREE(ndppn);
25
26     /* pass the string as the value to the PMIX_LOCAL_PEERS key */
27     PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
28     /* release the list */
29     free(localranks);
30
```

C

31 The **PMIX_LOCAL_CPUSSETS** value is constructed in a similar manner. In the provided
32 example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a
33 comma-delimited string of processor IDs) of the processors assigned to each process has
34 previously been generated and stored on the process description. Thus, the value can be
35 constructed as shown below:

```

1      char **ndcpus = NULL;
2      char *localcpus;
3      size_t m;
4      pmix_info_t info;
5
6      for (m=0; m < mynode->num_procs; m++)
7          /* ignore processes that are not part of the target job */
8          if (!PMIX_CHECK_NAMESPACE(targetjob, mynode->proc[m].nspace))
9              continue;
10
11         PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
12
13         /* convert the array into a colon-delimited string */
14         localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
15         /* release the local array */
16         PMIX_ARGV_FREE(ndcpus);
17
18         /* pass the string as the value to the PMIX_LOCAL_CPUSSETS key */
19         PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSSETS, localcpus, PMIX_STRING);
20         /* release the list */
21         free(localcpus);
22

```

23 Note that for efficiency, these two values can be computed at the same time.
24 The final *info* array might therefore look like the diagram in 11.5:

25 11.1.4 PMIx_server_deregister_namespace

26 Summary

27 Deregister a namespace.

28 Format

PMIx v1.0

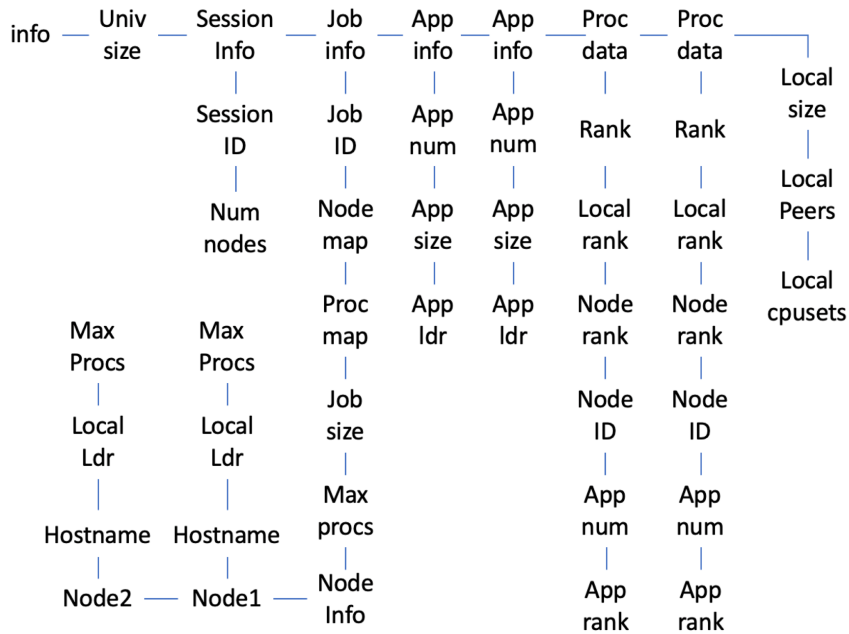


Figure 11.5.: Final information array

```

1 void PMIx_server_deregister_namespace(const pmix_namespace_t nspace,
2                                     pmix_op_cbfunc_t cbfunc, void *cbdata)

```

- 3 **IN nspace**
Namespace (string)
- 4
- 5 **IN cbfunc**
Callback function `pmix_op_cbfunc_t` (function reference)
- 6
- 7 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 8

Description

10 Deregister the specified *namespace* and purge all objects relating to it, including any client information
 11 from that namespace. This is intended to support persistent PMIx servers by providing an
 12 opportunity for the host RM to tell the PMIx server library to release all memory for a completed
 13 job. Note that the library must not invoke the callback function prior to returning from the API.

1 11.1.5 PMIx_server_register_client

2 Summary

3 Register a client process with the PMIx server library.

4 Format

PMIx v1.0

C

5 `pmix_status_t`

```
6 PMIx_server_register_client(const pmix_proc_t *proc,  
7                            uid_t uid, gid_t gid,  
8                            void *server_object,  
9                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

10 **IN** `proc`

11 `pmix_proc_t` structure (handle)

12 **IN** `uid`

13 user id (integer)

14 **IN** `gid`

15 group id (integer)

16 **IN** `server_object`

17 (memory reference)

18 **IN** `cbfunc`

19 Callback function `pmix_op_cbfunc_t` (function reference)

20 **IN** `cbdata`

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

22 Returns one of the following:

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

30 Description

31 Register a client process with the PMIx server library.

32 The host server can also, if it desires, provide an object it wishes to be returned when a server
33 function is called that relates to a specific process. For example, the host server may have an object
34 that tracks the specific client. Passing the object to the library allows the library to provide that
35 object to the host server during subsequent calls related to that client, such as a

1 [pmix_server_client_connected_fn_t](#) function. This allows the host server to access
2 the object without performing a lookup based on the client's namespace and rank.

Advice to PMIx server hosts

3 Host environments are required to execute this operation prior to starting the client process. The
4 expected user ID and group ID of the child process allows the server library to properly authenticate
5 clients as they connect by requiring the two values to match. Accordingly, the detected user and
6 group ID's of the connecting process are not included in the
7 [pmix_server_client_connected_fn_t](#) server module function.

Advice to PMIx library implementers

8 For security purposes, the PMIx server library should check the user and group ID's of a
9 connecting process against those provided for the declared client process identifier via the
10 [PMIx_server_register_client](#) prior to completing the connection.

11 11.1.6 [PMIx_server_deregister_client](#)

12 Summary

13 Deregister a client and purge all data relating to it.

14 Format

PMIx v1.0

```
15 void  
16 PMIx_server_deregister_client(const pmix_proc_t *proc,  
17                               pmix_op_cbfunc_t cbfunc, void *cbdata)
```

18 **IN** `proc`
19 [pmix_proc_t](#) structure (handle)
20 **IN** `cbfunc`
21 Callback function [pmix_op_cbfunc_t](#) (function reference)
22 **IN** `cbdata`
23 Data to be passed to the callback function (memory reference)

24 Description

25 The [PMIx_server_deregister_namespace](#) API will delete all client information for that
26 namespace. The PMIx server library will automatically perform that operation upon disconnect of
27 all local clients. This API is therefore intended primarily for use in exception cases, but can be
28 called in non-exception cases if desired. Note that the library must not invoke the callback function
29 prior to returning from the API.

1 11.1.7 PMIx_server_setup_fork

2 Summary

3 Setup the environment of a child process to be forked by the host.

4 Format

PMIx v1.0

C

5 `pmix_status_t`

6 `PMIx_server_setup_fork(const pmix_proc_t *proc,`
7 `char ***env)`

C

8 **IN** `proc`

9 `pmix_proc_t` structure (handle)

10 **IN** `env`

11 Environment array (array of strings)

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

13 Description

14 Setup the environment of a child process to be forked by the host so it can correctly interact with
15 the PMIx server.

Advice to PMIx server hosts

16 Host environments are required to execute this operation prior to starting the client process.

17 The PMIx client needs some setup information so it can properly connect back to the server. This
18 function will set appropriate environmental variables for this purpose, and will also provide any
19 environmental variables that were specified in the launch command (e.g., via `PMIx_Spawn`) plus
20 other values (e.g., variables required to properly initialize the client's fabric library).

21 11.1.8 PMIx_server_dmodex_request

22 Summary

23 Define a function by which the host server can request modex data from the local PMIx server.

1
PMIx v1.0

Format

C

```
2 pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,  
3 pmix_dmodex_response_fn_t cbfunc,  
4 void *cbdata)
```

C

- 5 **IN** `proc`
- 6 `pmix_proc_t` structure (handle)
- 7 **IN** `cbfunc`
- 8 Callback function `pmix_dmodex_response_fn_t` (function reference)
- 9 **IN** `cbdata`
- 10 Data to be passed to the callback function (memory reference)

11 Returns one of the following:

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

Description

17 Define a function by which the host server can request modex data from the local PMIx server.
18 Traditional wireup procedures revolve around the per-process posting of data (e.g., location and
19 endpoint information) via the **PMIx_Put** and **PMIx_Commit** functions followed by a
20 **PMIx_Fence** barrier that globally exchanges the posted information. However, the barrier
21 operation represents a significant time impact at large scale.

22 PMIx supports an alternative wireup method known as *Direct Modex* that replaces the
23 barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In
24 place of the barrier operation, data posted by each process is cached on the local PMIx server.
25 When a process requests the information posted by a particular peer, it first checks the local cache
26 to see if the data is already available. If not, then the request is passed to the local PMIx server,
27 which subsequently requests that its RM host request the data from the RM daemon on the node
28 where the specified peer process is located. Upon receiving the request, the RM daemon passes the
29 request into its PMIx server library using the **PMIx_server_dmodex_request** function,
30 receiving the response in the provided `cbfunc` once the indicated process has posted its information.
31 The RM daemon then returns the data to the requesting daemon, who subsequently passes the data
32 to its PMIx server library for transfer to the requesting client.

Advice to users

33 While direct modex allows for faster launch times by eliminating the barrier operation, per-peer
34 retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by
35 returning posted information from all processes on a node upon first request - but in general direct
36 modex remains best suited for sparsely connected applications.

1 11.1.9 PMIx_server_setup_application

2 Summary

3 Provide a function by which the resource manager can request application-specific setup data prior
4 to launch of a `job`.

5 Format

PMIx v2.0

C

```
6 pmix_status_t  
7 PMIx_server_setup_application(const pmix_namespace_t nspace,  
8                             pmix_info_t info[], size_t ninfo,  
9                             pmix_setup_application_cbfunc_t cbfunc,  
10                            void *cbdata)
```

C

11 **IN nspace**
12 namespace (string)
13 **IN info**
14 Array of info structures (array of handles)
15 **IN ninfo**
16 Number of elements in the *info* array (integer)
17 **IN cbfunc**
18 Callback function `pmix_setup_application_cbfunc_t` (function reference)
19 **IN cbdata**
20 Data to be passed to the *cbfunc* callback function (memory reference)

21 Returns one of the following:

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

Required Attributes

26 PMIx libraries that support this operation are required to support the following:

27 **PMIX_SETUP_APP_ENVARS** "pmix.setup.env" (bool)
28 Harvest and include relevant environmental variables
29 **PMIX_SETUP_APP_NONENVARS** "pmix.setup.nenv" (bool)
30 Include all relevant data other than environmental variables
31 **PMIX_SETUP_APP_ALL** "pmix.setup.all" (bool)

1 Include all relevant data

2 **PMIX_ALLOC_NETWORK** "pmix.alloc.net" (array)

3 Array of `pmix_info_t` describing requested network resources. This must include at
4 least: `PMIX_ALLOC_NETWORK_ID`, `PMIX_ALLOC_NETWORK_TYPE`, and
5 `PMIX_ALLOC_NETWORK_ENDPTS`, plus whatever other descriptors are desired.

6 **PMIX_ALLOC_NETWORK_ID** "pmix.alloc.netid" (char*)

7 The key to be used when accessing this requested network allocation. The allocation will be
8 returned/stored as a `pmix_data_array_t` of `pmix_info_t` indexed by this key and
9 containing at least one entry with the same key and the allocated resource description. The
10 type of the included value depends upon the network support. For example, a TCP allocation
11 might consist of a comma-delimited string of socket ranges such as
12 "32000-32100,33005,38123-38146". Additional entries will consist of any provided
13 resource request directives, along with their assigned values. Examples include:
14 `PMIX_ALLOC_NETWORK_TYPE` - the type of resources provided;
15 `PMIX_ALLOC_NETWORK_PLANE` - if applicable, what plane the resources were assigned
16 from; `PMIX_ALLOC_NETWORK_QOS` - the assigned QoS; `PMIX_ALLOC_BANDWIDTH` -
17 the allocated bandwidth; `PMIX_ALLOC_NETWORK_SEC_KEY` - a security key for the
18 requested network allocation. NOTE: the assigned values may differ from those requested,
19 especially if `PMIX_INFO_REQD` was not set in the request.

20 **PMIX_ALLOC_NETWORK_SEC_KEY** "pmix.alloc.nsec" (`pmix_byte_object_t`)

21 Network security key

22 **PMIX_ALLOC_NETWORK_TYPE** "pmix.alloc.nettype" (char*)

23 Type of desired transport (e.g., "tcp", "udp")

24 **PMIX_ALLOC_NETWORK_PLANE** "pmix.alloc.netplane" (char*)

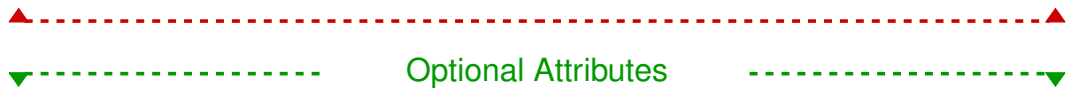
25 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

26 **PMIX_ALLOC_NETWORK_ENDPTS** "pmix.alloc.endpts" (`size_t`)

27 Number of endpoints to allocate per process

28 **PMIX_ALLOC_NETWORK_ENDPTS_NODE** "pmix.alloc.endpts.nd" (`size_t`)

29 Number of endpoints to allocate per node



Optional Attributes

30 PMIx libraries that support this operation may support the following:

31 **PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (`float`)

32 Mbits/sec.

33 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)

34 Quality of service level.

35 **PMIX_ALLOC_TIME** "pmix.alloc.time" (`uint32_t`)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

Time in seconds.

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.

1
PMIx v2.0

Format

C

```

2 pmix_status_t
3 PMIx_server_setup_local_support(const pmix_namespace_t nspace,
4                               pmix_info_t info[], size_t ninfo,
5                               pmix_op_cbfunc_t cbfunc,
6                               void *cbdata);

```

C

- 7 **IN nspace**
Namespace (string)
- 8
- 9 **IN info**
Array of info structures (array of handles)
- 10
- 11 **IN ninfo**
Number of elements in the *info* array (**size_t**)
- 12
- 13 **IN cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)
- 14
- 15 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 16

17 Returns one of the following:

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

Description

25 Provide a function by which the local PMIx server can perform any application-specific operations
26 prior to spawning local clients of a given application. For example, a network library might need to
27 setup the local driver for “instant on” addressing. The data provided in the *info* array is the data
28 returned to the host RM by the callback function executed as a result of a call to

29 [PMIx_server_setup_application](#).

Advice to PMIx server hosts

31 Host environments are required to execute this operation prior to starting any local application
32 processes from the specified namespace.

1 11.1.12 PMIx_server_IOF_deliver

2 Summary

3 Provide a function by which the host environment can pass forwarded IO to the PMIx server library
4 for distribution to its clients.

5 Format

PMIx v3.0

C

```
6 pmix_status_t  
7 PMIx_server_IOF_deliver(const pmix_proc_t *source,  
8 pmix_iof_channel_t channel,  
9 const pmix_byte_object_t *bo,  
10 const pmix_info_t info[], size_t ninfo,  
11 pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

12 **IN source**
13 Pointer to [pmix_proc_t](#) identifying source of the IO (handle)
14 **IN channel**
15 IO channel of the data ([pmix_iof_channel_t](#))
16 **IN bo**
17 Pointer to [pmix_byte_object_t](#) containing the payload to be delivered (handle)
18 **IN info**
19 Array of [pmix_info_t](#) metadata describing the data (array of handles)
20 **IN ninfo**
21 Number of elements in the *info* array ([size_t](#))
22 **IN cbfunc**
23 Callback function [pmix_op_cbfunc_t](#) (function reference)
24 **IN cbdata**
25 Data to be passed to the callback function (memory reference)

26 Returns one of the following:

- 27 • [PMIX_SUCCESS](#) , indicating that the request is being processed by the host environment - result
28 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
29 function prior to returning from the API.
- 30 • [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and
31 returned *success* - the *cbfunc* will not be called
- 32 • a PMIx error constant indicating either an error in the input or that the request was immediately
33 processed and failed - the *cbfunc* will not be called

Description

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

```
pmix_status_t
PMIx_server_collect_inventory(const pmix_info_t directives[],
                             size_t ndirs,
                             pmix_info_cbfunc_t cbfunc,
                             void *cbdata);
```

IN directives

Array of `pmix_info_t` directing the request (array of handles)

IN ndirs

Number of elements in the *directives* array (`size_t`)

IN cbfunc

Callback function to return collected data (`pmix_info_cbfunc_t` function reference)

IN cbdata

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

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant. In the event the function returns an error, the *cbfunc* will not be called.

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.

1 11.1.14 PMIx_server_deliver_inventory

2 Summary

3 Pass collected inventory to the PMIx server library for storage

4 Format

PMIx v3.0

C

```
5 pmix_status_t
6 PMIx_server_deliver_inventory(const pmix_info_t info[],
7                               size_t ninfo,
8                               const pmix_info_t directives[],
9                               size_t ndirs,
10                              pmix_op_cbfunc_t cbfunc,
11                              void *cbdata);
```

C

12 **IN info**
13 Array of [pmix_info_t](#) containing the inventory (array of handles)

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

16 **IN directives**
17 Array of [pmix_info_t](#) directing the request (array of handles)

18 **IN ndirs**
19 Number of elements in the *directives* array (**size_t**)

20 **IN cbfunc**
21 Callback function [pmix_op_cbfunc_t](#) (function reference)

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

24 Returns one of the following:

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

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

```
1 typedef struct pmix_server_module_3_0_0_t
2     /* v1x interfaces */
3     pmix_server_client_connected_fn_t    client_connected;
4     pmix_server_client_finalized_fn_t    client_finalized;
5     pmix_server_abort_fn_t               abort;
6     pmix_server_fence_nb_fn_t            fence_nb;
7     pmix_server_dmodex_req_fn_t           direct_modex;
8     pmix_server_publish_fn_t              publish;
9     pmix_server_lookup_fn_t               lookup;
10    pmix_server_unpublish_fn_t             unpublish;
11    pmix_server_spawn_fn_t                 spawn;
12    pmix_server_connect_fn_t               connect;
13    pmix_server_disconnect_fn_t            disconnect;
14    pmix_server_register_events_fn_t       register_events;
15    pmix_server_deregister_events_fn_t     deregister_events;
16    pmix_server_listener_fn_t              listener;
17    /* v2x interfaces */
18    pmix_server_notify_event_fn_t          notify_event;
19    pmix_server_query_fn_t                  query;
20    pmix_server_tool_connection_fn_t        tool_connected;
21    pmix_server_log_fn_t                     log;
22    pmix_server_alloc_fn_t                   allocate;
23    pmix_server_job_control_fn_t             job_control;
24    pmix_server_monitor_fn_t                 monitor;
25    /* v3x interfaces */
26    pmix_server_get_cred_fn_t                get_credential;
27    pmix_server_validate_cred_fn_t           validate_credential;
28    pmix_server_iof_fn_t                     iof_pull;
29    pmix_server_stdin_fn_t                   push_stdin;
30    pmix_server_module_t;
```

C

32 11.2.2 pmix_server_client_connected_fn_t

33 Summary

34 Notify the host server that a client connected to this server.

1
PMIx v1.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_client_connected_fn_t) (
3     const pmix_proc_t *proc,
4     void* server_object,
5     pmix_op_cbfunc_t cbfunc,
6     void *cbdata)

```

C

- 7 **IN** `proc`
- 8 `pmix_proc_t` structure (handle)
- 9 **IN** `server_object`
- 10 object reference (memory reference)
- 11 **IN** `cbfunc`
- 12 Callback function `pmix_op_cbfunc_t` (function reference)
- 13 **IN** `cbdata`
- 14 Data to be passed to the callback function (memory reference)

15 Returns one of the following:

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

Description

23 Notify the host environment that a client has called **PMIx_Init**. Note that the client will be in a

24 blocked state until the host server executes the callback function, thus allowing the PMIx server

25 support library to release the client. The `server_object` parameter will be the value of the

26 `server_object` parameter passed to **PMIx_server_register_client** by the host server

27 when registering the connecting client. If provided, an implementation of

28 `pmix_server_client_connected_fn_t` is only required to call the callback function

29 designated. A host server can choose to not be notified when clients connect by setting

30 `pmix_server_client_connected_fn_t` to **NULL**.

31

32 It is possible that only a subset of the clients in a namespace call **PMIx_Init**. The server's

33 `pmix_server_client_connected_fn_t` implementation should not depend on being

34 called once per rank in a namespace or delay calling the callback function until all ranks have

35 connected. However, if a rank makes any PMIx calls, it must first call **PMIx_Init** and therefore

36 the server's `pmix_server_client_connected_fn_t` will be called before any other

37 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

```
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)
```

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)
```

1 **IN** **proc**
2 **pmix_proc_t** structure identifying the process requesting the abort (handle)
3 **IN** **server_object**
4 object reference (memory reference)
5 **IN** **status**
6 exit status (integer)
7 **IN** **msg**
8 exit status message (string)
9 **IN** **procs**
10 Array of **pmix_proc_t** structures identifying the processes to be terminated (array of
11 handles)
12 **IN** **nprocs**
13 Number of elements in the *procs* array (integer)
14 **IN** **cbfunc**
15 Callback function **pmix_op_cbfunc_t** (function reference)
16 **IN** **cbdata**
17 Data to be passed to the callback function (memory reference)

18 Returns one of the following:

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

26 **Description**

27 A local client called **PMIx_Abort** . Note that the client will be in a blocked state until the host
28 server executes the callback function, thus allowing the PMIx server library to release the client.
29 The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all
30 processes in the client's namespace are to be terminated.

31 **11.2.5 pmix_server_fence_nb_fn_t**

32 **Summary**

33 At least one client called either **PMIx_Fence** or **PMIx_Fence_nb** .

1
PMIx v1.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_fence_fn_t) (  
3     const pmix_proc_t procs[],  
4     size_t nprocs,  
5     const pmix_info_t info[],  
6     size_t ninfo,  
7     char *data, size_t ndata,  
8     pmix_modex_cbfnc_t cbfunc,  
9     void *cbdata)
```

C

10 **IN** **procs**
11 Array of [pmix_proc_t](#) structures identifying operation participants(array of handles)

12 **IN** **nprocs**
13 Number of elements in the *procs* array (integer)

14 **IN** **info**
15 Array of info structures (array of handles)

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

18 **IN** **data**
19 (string)

20 **IN** **ndata**
21 (integer)

22 **IN** **cbfunc**
23 Callback function [pmix_modex_cbfnc_t](#) (function reference)

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

26 Returns one of the following:

- 27
- 28 • [PMIX_SUCCESS](#) , indicating that the request is being processed by the host environment - result
29 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
prior to returning from the API.
 - 30 • [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and
31 returned *success* - the *cbfunc* will not be called
 - 32 • a PMIx error constant indicating either an error in the input or that the request was immediately
33 processed and failed - the *cbfunc* will not be called

Required Attributes

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_fence_nb_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);
```

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.

1 11.2.6 pmix_server_dmodex_req_fn_t

2 Summary

3 Used by the PMIx server to request its local host contact the PMIx server on the remote node that
4 hosts the specified proc to obtain and return a direct modex blob for that proc.

5 Format

PMIx v1.0

C

```
6 typedef pmix_status_t (*pmix_server_dmodex_req_fn_t) (  
7     const pmix_proc_t *proc,  
8     const pmix_info_t info[],  
9     size_t ninfo,  
10    pmix_modex_cbfunc_t cbfunc,  
11    void *cbdata)
```

C

12 IN proc

13 pmix_proc_t structure identifying the process whose data is being requested (handle)

14 IN info

15 Array of info structures (array of handles)

16 IN ninfo

17 Number of elements in the info array (integer)

18 IN cbfunc

19 Callback function pmix_modex_cbfunc_t (function reference)

20 IN cbdata

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

22 Returns one of the following:

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

Required Attributes

28 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

```
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)
```

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)

1 **IN cldata**

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

3 Returns one of the following:

- 4 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
5 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
6 prior to returning from the API.
- 7 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
8 returned *success* - the *cbfunc* will not be called
- 9 • a PMIx error constant indicating either an error in the input or that the request was immediately
10 processed and failed - the *cbfunc* will not be called

▼----- Required Attributes -----▼

11 PMIx libraries are required to pass any provided attributes to the host environment for processing.
12 In addition, the following attributes are required to be included in the passed *info* array:

13 **PMIX_USERID** "pmix.euid" (uint32_t)
14 Effective user id.

15 **PMIX_GRPID** "pmix.egid" (uint32_t)
16 Effective group id.

17
18 Host environments that implement this entry point are required to support the following attributes:

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

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

▲----- Optional Attributes -----▲

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

24 **PMIX_TIMEOUT** "pmix.timeout" (int)
25 Time in seconds before the specified operation should time out (0 indicating infinite) in
26 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
27 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_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_cbfnc_t cbfunc,
    void *cbdata)
```

1 **IN** **proc**
 2 **pmix_proc_t** structure of the process seeking the data (handle)
 3 **IN** **keys**
 4 (array of strings)
 5 **IN** **info**
 6 Array of info structures (array of handles)
 7 **IN** **ninfo**
 8 Number of elements in the *info* array (integer)
 9 **IN** **cbfunc**
 10 Callback function **pmix_lookup_cbfunc_t** (function reference)
 11 **IN** **cbdata**
 12 Data to be passed to the callback function (memory reference)

13 Returns one of the following:

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

Required Attributes

21 PMIx libraries are required to pass any provided attributes to the host environment for processing.
 22 In addition, the following attributes are required to be included in the passed *info* array:

23 **PMIX_USERID** "pmix.euid" (**uint32_t**)
 24 Effective user id.

25 **PMIX_GRPID** "pmix.egid" (**uint32_t**)
 26 Effective group id.

27

28 Host environments that implement this entry point are required to support the following attributes:

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

31 **PMIX_WAIT** "pmix.wait" (**int**)
 32 Caller requests that the PMIx server wait until at least the specified number of values are
 33 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_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.

1
PMIx v1.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_unpublish_fn_t) (  
3     const pmix_proc_t *proc,  
4     char **keys,  
5     const pmix_info_t info[],  
6     size_t ninfo,  
7     pmix_op_cbfunc_t cbfunc,  
8     void *cbdata)
```

C

9 **IN** `proc`
10 `pmix_proc_t` structure identifying the process making the request (handle)
11 **IN** `keys`
12 (array of strings)
13 **IN** `info`
14 Array of info structures (array of handles)
15 **IN** `ninfo`
16 Number of elements in the *info* array (integer)
17 **IN** `cbfunc`
18 Callback function `pmix_op_cbfunc_t` (function reference)
19 **IN** `cbdata`
20 Data to be passed to the callback function (memory reference)

21 Returns one of the following:

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

Required Attributes

29 PMIx libraries are required to pass any provided attributes to the host environment for processing.
30 In addition, the following attributes are required to be included in the passed *info* array:

31 **PMIX_USERID** "pmix.euid" (uint32_t)
32 Effective user id.
33 **PMIX_GRPID** "pmix.egid" (uint32_t)
34 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_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.

1
PMIx v1.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_spawn_fn_t) (  
3     const pmix_proc_t *proc,  
4     const pmix_info_t job_info[],  
5     size_t ninfo,  
6     const pmix_app_t apps[],  
7     size_t napps,  
8     pmix_spawn_cbfunc_t cbfunc,  
9     void *cbdata)
```

C

10 **IN** `proc`
11 `pmix_proc_t` structure of the process making the request (handle)
12 **IN** `job_info`
13 Array of info structures (array of handles)
14 **IN** `ninfo`
15 Number of elements in the `jobinfo` array (integer)
16 **IN** `apps`
17 Array of `pmix_app_t` structures (array of handles)
18 **IN** `napps`
19 Number of elements in the `apps` array (integer)
20 **IN** `cbfunc`
21 Callback function `pmix_spawn_cbfunc_t` (function reference)
22 **IN** `cbdata`
23 Data to be passed to the callback function (memory reference)

24 Returns one of the following:

- 25 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
26 will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function
27 prior to returning from the API.
- 28 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
29 returned `success` - the `cbfunc` will not be called
- 30 • a PMIx error constant indicating either an error in the input or that the request was immediately
31 processed and failed - the `cbfunc` will not be called

Required Attributes

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.
33 In addition, the following attributes are required to be included in the passed `info` array:

34 **PMIX_USERID** "pmix.euid" (`uint32_t`)
35 Effective user id.

1 **PMIX_GRPID** "pmix.egid" (uint32_t)

2 Effective group id.

3
4 Host environments that provide this module entry point are required to pass the **PMIX_SPAWNED**
5 and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those
6 values can be returned to clients upon connection to the PMIx server. In addition, they are required
7 to support the following attributes when present in either the *job_info* or the *info* array of an
8 element of the *apps* array:

9 **PMIX_WDIR** "pmix.wdir" (char*)

10 Working directory for spawned processes.

11 **PMIX_SET_SESSION_CWD** "pmix.ssnwd" (bool)

12 Set the application's current working directory to the session working directory assigned by
13 the RM - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for
14 the rank to discover the session working directory assigned to the provided namespace

15 **PMIX_PREFIX** "pmix.prefix" (char*)

16 Prefix to use for starting spawned processes.

17 **PMIX_HOST** "pmix.host" (char*)

18 Comma-delimited list of hosts to use for spawned processes.

19 **PMIX_HOSTFILE** "pmix.hostfile" (char*)

20 Hostfile to use for spawned processes.



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

22 **PMIX_ADD_HOSTFILE** "pmix.addhostfile" (char*)

23 Hostfile listing hosts to add to existing allocation.

24 **PMIX_ADD_HOST** "pmix.addhost" (char*)

25 Comma-delimited list of hosts to add to the allocation.

26 **PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)

27 Preload binaries onto nodes.

28 **PMIX_PRELOAD_FILES** "pmix.preloadfiles" (char*)

29 Comma-delimited list of files to pre-position on nodes.

30 **PMIX_PERSONALITY** "pmix.pers" (char*)

31 Name of personality to use.

32 **PMIX_MAPPER** "pmix.mapper" (char*)

1 Mapping mechanism to use for placing spawned processes - when accessed using
2 `PMIx_Get` , use the `PMIX_RANK_WILDCARD` value for the rank to discover the mapping
3 mechanism used for the provided namespace.

4 `PMIX_DISPLAY_MAP` "pmix.dispmap" (bool)

5 Display process mapping upon spawn.

6 `PMIX_PPR` "pmix.ppr" (char*)

7 Number of processes to spawn on each identified resource.

8 `PMIX_MAPBY` "pmix.mapby" (char*)

9 Process mapping policy - when accessed using `PMIx_Get` , use the
10 `PMIX_RANK_WILDCARD` value for the rank to discover the mapping policy used for the
11 provided namespace

12 `PMIX_RANKBY` "pmix.rankby" (char*)

13 Process ranking policy - when accessed using `PMIx_Get` , use the
14 `PMIX_RANK_WILDCARD` value for the rank to discover the ranking algorithm used for the
15 provided namespace

16 `PMIX_BINDTO` "pmix.bindto" (char*)

17 Process binding policy - when accessed using `PMIx_Get` , use the
18 `PMIX_RANK_WILDCARD` value for the rank to discover the binding policy used for the
19 provided namespace

20 `PMIX_NON_PMI` "pmix.nonpmi" (bool)

21 Spawned processes will not call `PMIx_Init` .

22 `PMIX_STDIN_TGT` "pmix.stdin" (uint32_t)

23 Spawned process rank that is to receive `stdin`.

24 `PMIX_FWD_STDIN` "pmix.fwd.stdin" (bool)

25 Forward this process's `stdin` to the designated process.

26 `PMIX_FWD_STDOUT` "pmix.fwd.stdout" (bool)

27 Forward `stdout` from spawned processes to this process.

28 `PMIX_FWD_STDERR` "pmix.fwd.stderr" (bool)

29 Forward `stderr` from spawned processes to this process.

30 `PMIX_DEBUGGER_DAEMONS` "pmix.debugger" (bool)

31 Spawned application consists of debugger daemons.

32 `PMIX_TAG_OUTPUT` "pmix.tagout" (bool)

33 Tag application output with the identity of the source process.

34 `PMIX_TIMESTAMP_OUTPUT` "pmix.tsout" (bool)

35 Timestamp output from applications.

36 `PMIX_MERGE_STDERR_STDOUT` "pmix.mergeerrout" (bool)

1 Merge `stdout` and `stderr` streams from application processes.

2 **PMIX_OUTPUT_TO_FILE** "`pmix.outfile`" (`char*`)

3 Output application output to the specified file.

4 **PMIX_INDEX_ARGV** "`pmix.indxargv`" (`bool`)

5 Mark the `argv` with the rank of the process.

6 **PMIX_CPUS_PER_PROC** "`pmix.cpusperproc`" (`uint32_t`)

7 Number of cpus to assign to each rank - when accessed using `PMIx_Get`, use the
8 **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the
9 provided namespace

10 **PMIX_NO_PROCS_ON_HEAD** "`pmix.nolocal`" (`bool`)

11 Do not place processes on the head node.

12 **PMIX_NO_OVERSUBSCRIBE** "`pmix.noover`" (`bool`)

13 Do not oversubscribe the cpus.

14 **PMIX_REPORT_BINDINGS** "`pmix.repbinding`" (`bool`)

15 Report bindings of the individual processes.

16 **PMIX_CPU_LIST** "`pmix.cpulist`" (`char*`)

17 List of cpus to use for this job - when accessed using `PMIx_Get`, use the
18 **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided
19 namespace

20 **PMIX_JOB_RECOVERABLE** "`pmix.recover`" (`bool`)

21 Application supports recoverable operations.

22 **PMIX_JOB_CONTINUOUS** "`pmix.continuous`" (`bool`)

23 Application is continuous, all failed processes should be immediately restarted.

24 **PMIX_MAX_RESTARTS** "`pmix.maxrestarts`" (`uint32_t`)

25 Maximum number of times to restart a job - when accessed using `PMIx_Get`, use the
26 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
27 namespace

28 **PMIX_TIMEOUT** "`pmix.timeout`" (`int`)

29 Time in seconds before the specified operation should time out (`0` indicating infinite) in
30 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
31 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

```
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)
```

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.

1 11.2.12 pmix_server_disconnect_fn_t

2 Summary

3 Disconnect a previously connected set of processes.

4 Format

PMIx v1.0

```
5 typedef pmix_status_t (*pmix_server_disconnect_fn_t) (  
6     const pmix_proc_t procs[],  
7     size_t nprocs,  
8     const pmix_info_t info[],  
9     size_t ninfo,  
10    pmix_op_cbfunc_t cbfunc,  
11    void *cbdata)
```

12 IN procs

13 Array of [pmix_proc_t](#) structures identifying participants (array of handles)

14 IN nprocs

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

16 IN info

17 Array of info structures (array of handles)

18 IN ninfo

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

20 IN cbfunc

21 Callback function [pmix_op_cbfunc_t](#) (function reference)

22 IN cbdata

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

24 Returns one of the following:

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

Required Attributes

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

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 `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

C

```

2 typedef pmix_status_t (*pmix_server_register_events_fn_t) (
3     pmix_status_t *codes,
4     size_t ncodes,
5     const pmix_info_t info[],
6     size_t ninfo,
7     pmix_op_cbfunc_t cbfunc,
8     void *cbdata)

```

C

- 9 **IN codes**
Array of [pmix_status_t](#) values (array of handles)
- 11 **IN ncodes**
Number of elements in the *codes* array (integer)
- 13 **IN info**
Array of info structures (array of handles)
- 15 **IN ninfo**
Number of elements in the *info* array (integer)
- 17 **IN cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)
- 19 **IN cbdata**
Data to be passed to the callback function (memory reference)

21 Returns one of the following:

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

Required Attributes

29 PMIx libraries are required to pass any provided attributes to the host environment for processing.
30 In addition, the following attributes are required to be included in the passed *info* array:

- 31 **PMIX_USERID** "pmix.euid" (uint32_t)
32 Effective user id.
- 33 **PMIX_GRPID** "pmix.egid" (uint32_t)
34 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 between `PMIX_ERR_SYS_BASE` and `PMIX_ERR_SYS_OTHER`, inclusive) 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.

1
PMIx v1.0

Format

C

```
typedef pmix_status_t (*pmix_server_deregister_events_fn_t) (  
    pmix_status_t *codes,  
    size_t ncodes,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

C

- 7 **IN codes**
8 Array of `pmix_status_t` values (array of handles)
- 9 **IN ncodes**
10 Number of elements in the `codes` array (integer)
- 11 **IN cbfunc**
12 Callback function `pmix_op_cbfunc_t` (function reference)
- 13 **IN cbdata**
14 Data to be passed to the callback function (memory reference)

15 Returns one of the following:

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

Description

24 Deregister to receive notifications for the specified events to which the PMIx server has previously
25 registered.

Advice to PMIx library implementers

26 The PMIx server library must track all client registrations. This module function shall only be
27 called when:

- 28 • the library is deregistering environmental codes (i.e., a PMIx codes in the range between
29 **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER**, inclusive) or codes that lies outside
30 the defined PMIx range of constants; and
- 31 • no client (including the server library itself) remains registered for notifications on any included
32 code - i.e., a code should be included in this call only when no registered notifications against it
33 remain.

1 11.2.15 pmix_server_notify_event_fn_t

2 Summary

3 Notify the specified processes of an event.

4 Format

PMIx v2.0

C

```
5 typedef pmix_status_t (*pmix_server_notify_event_fn_t) (pmix_status_t code,  
6                                                         const pmix_proc_t *source,  
7                                                         pmix_data_range_t range,  
8                                                         pmix_info_t info[],  
9                                                         size_t ninfo,  
10                                                         pmix_op_cbfunc_t cbfunc,  
11                                                         void *cbdata);
```

C

12 **IN code**

13 The `pmix_status_t` event code being referenced structure (handle)

14 **IN source**

15 `pmix_proc_t` of process that generated the event (handle)

16 **IN range**

17 `pmix_data_range_t` range over which the event is to be distributed (handle)

18 **IN info**

19 Optional array of `pmix_info_t` structures containing additional information on the event
20 (array of handles)

21 **IN ninfo**

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

23 **IN cbfunc**

24 Callback function `pmix_op_cbfunc_t` (function reference)

25 **IN cbdata**

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

27 Returns one of the following:

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

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

```
typedef pmix_status_t (*pmix_server_listener_fn_t) (  
    int listening_sd,  
    pmix_connection_cbfunc_t cbfunc,  
    void *cbdata)
```

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

```
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)
```

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.phtable" (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.

```

1 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
2     Report minimum and maximum values.
3 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
4     String identifier of the allocation whose status is being requested.
5 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
6     Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
7

```



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.



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_cbfnc_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_cbfnc_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.

1
PMIx v2.0

Format

C

```
2 typedef void (*pmix_server_log_fn_t) (  
3     const pmix_proc_t *client,  
4     const pmix_info_t data[], size_t ndata,  
5     const pmix_info_t directives[], size_t ndirs,  
6     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

7 **IN** `client`
8 `pmix_proc_t` structure (handle)
9 **IN** `data`
10 Array of info structures (array of handles)
11 **IN** `ndata`
12 Number of elements in the `data` array (integer)
13 **IN** `directives`
14 Array of info structures (array of handles)
15 **IN** `ndirs`
16 Number of elements in the `directives` array (integer)
17 **IN** `cbfunc`
18 Callback function `pmix_op_cbfunc_t` (function reference)
19 **IN** `cbdata`
20 Data to be passed to the callback function (memory reference)

Required Attributes

21 PMIx libraries are required to pass any provided attributes to the host environment for processing.
22 In addition, the following attributes are required to be included in the passed *info* array:

23 **PMIX_USERID** "pmix.euid" (uint32_t)
24 Effective user id.

25 **PMIX_GRPID** "pmix.egid" (uint32_t)
26 Effective group id.

27
28 Host environments that provide this module entry point are required to support the following
29 attributes:

30 **PMIX_LOG_STDERR** "pmix.log.stderr" (char*)
31 Log string to `stderr`.

32 **PMIX_LOG_STDOUT** "pmix.log.stdout" (char*)
33 Log string to `stdout`.

34 **PMIX_LOG_SYSLOG** "pmix.log.syslog" (char*)

1 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,
2 otherwise to local syslog




Optional Attributes

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

4 **PMIX_LOG_MSG** "pmix.log.msg" (pmix_byte_object_t)

5 Message blob to be sent somewhere.

6 **PMIX_LOG_EMAIL** "pmix.log.email" (pmix_data_array_t)

7 Log via email based on **pmix_info_t** containing directives.

8 **PMIX_LOG_EMAIL_ADDR** "pmix.log.emaddr" (char*)


9 Comma-delimited list of email addresses that are to receive the message.

10 **PMIX_LOG_EMAIL_SUBJECT** "pmix.log.emsub" (char*)

11 Subject line for email.

12 **PMIX_LOG_EMAIL_MSG** "pmix.log.emmsg" (char*)

13 Message to be included in email.



Description

14 Log data on behalf of a client. This function is not intended for output of computational results, but
15 rather for reporting status and error messages. The host must not execute the callback function prior
16 to returning from the API.
17

18 11.2.20 pmix_server_alloc_fn_t

19 Summary

20 Request allocation operations on behalf of a client.

1
PMIx v2.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_alloc_fn_t) (
3     const pmix_proc_t *client,
4     pmix_alloc_directive_t directive,
5     const pmix_info_t data[], size_t ndata,
6     pmix_info_cbfunc_t cbfunc, void *cbdata)

```

C

- 7 **IN client**
- 8 `pmix_proc_t` structure of process making request (handle)
- 9 **IN directive**
- 10 Specific action being requested (`pmix_alloc_directive_t`)
- 11 **IN data**
- 12 Array of info structures (array of handles)
- 13 **IN ndata**
- 14 Number of elements in the *data* array (integer)
- 15 **IN cbfunc**
- 16 Callback function `pmix_info_cbfunc_t` (function reference)
- 17 **IN cbdata**
- 18 Data to be passed to the callback function (memory reference)

19 Returns one of the following:

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

Required Attributes

27 PMIx libraries are required to pass any provided attributes to the host environment for processing.
28 In addition, the following attributes are required to be included in the passed *info* array:

- 29 **PMIX_USERID** "pmix.euid" (`uint32_t`)
- 30 Effective user id.
- 31 **PMIX_GRPID** "pmix.egid" (`uint32_t`)
- 32 Effective group id.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

Host environments that provide this module entry point are required to support the following attributes:

- PMIX_ALLOC_ID** "pmix.alloc.id" (char*)
Provide a 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

1 from; `PMIX_ALLOC_NETWORK_QOS` - the assigned QoS; `PMIX_ALLOC_BANDWIDTH` -
2 the allocated bandwidth; `PMIX_ALLOC_NETWORK_SEC_KEY` - a security key for the
3 requested network allocation. NOTE: the assigned values may differ from those requested,
4 especially if `PMIX_INFO_REQD` was not set in the request.

5 `PMIX_ALLOC_BANDWIDTH` "pmix.alloc.bw" (float)
6 Mbits/sec.

7 `PMIX_ALLOC_NETWORK_QOS` "pmix.alloc.netqos" (char*)
8 Quality of service level.



9 **Description**

10 Request new allocation or modifications to an existing allocation on behalf of a client. Several
11 broad categories are envisioned, including the ability to:

- 12 • Request allocation of additional resources, including memory, bandwidth, and compute for an
13 existing allocation. Any additional allocated resources will be considered as part of the current
14 allocation, and thus will be released at the same time.
- 15 • Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not
16 affiliated with) the allocation of the requestor - thus the termination of one allocation will not
17 impact the other.
- 18 • Extend the reservation on currently allocated resources, subject to scheduling availability and
19 priorities.
- 20 • Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to
21 the scheduler with a promise to return them upon subsequent request.

22 The callback function provides a *status* to indicate whether or not the request was granted, and to
23 provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of
24 `pmix_info_t` structures.

25 **11.2.21 pmix_server_job_control_fn_t**

26 **Summary**

27 Execute a job control action on behalf of a client.

1
PMIx v2.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_job_control_fn_t) (  
3     const pmix_proc_t *requestor,  
4     const pmix_proc_t targets[], size_t ntargets,  
5     const pmix_info_t directives[], size_t ndirs,  
6     pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

7 **IN requestor**
8 [pmix_proc_t](#) structure of requesting process (handle)
9 **IN targets**
10 Array of proc structures (array of handles)
11 **IN ntargets**
12 Number of elements in the *targets* array (integer)
13 **IN directives**
14 Array of info structures (array of handles)
15 **IN ndirs**
16 Number of elements in the *info* array (integer)
17 **IN cbfunc**
18 Callback function [pmix_op_cbfunc_t](#) (function reference)
19 **IN cbdata**
20 Data to be passed to the callback function (memory reference)

21 Returns one of the following:

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

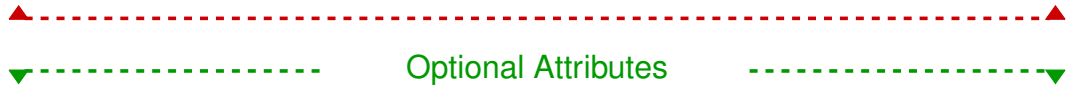
Required Attributes

29 PMIx libraries are required to pass any attributes provided by the client to the host environment for
30 processing. In addition, the following attributes are required to be included in the passed *info* array:

31 **PMIX_USERID** "pmix.euid" (uint32_t)
32 Effective user id.
33 **PMIX_GRPID** "pmix.egid" (uint32_t)
34 Effective group id.

1
2 Host environments that provide this module entry point are required to support the following
3 attributes:

4 **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
5 Provide a string identifier for this request.
6 **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
7 Pause the specified processes.
8 **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
9 Resume ("un-pause") the specified processes.
10 **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)
11 Forcibly terminate the specified processes and cleanup.
12 **PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)
13 Send given signal to specified processes.
14 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)
15 Politely terminate the specified processes.



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

17 **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
18 Cancel the specified request (**NULL** implies cancel all requests from this requestor).
19 **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)
20 Restart the specified processes using the given checkpoint ID.
21 **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)
22 Checkpoint the specified processes and assign the given ID to it.
23 **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)
24 Use event notification to trigger a process checkpoint.
25 **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)
26 Use the given signal to trigger a process checkpoint.
27 **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)
28 Time in seconds to wait for a checkpoint to complete.
29 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**
30 "pmix.jctrl.ckmethod" (pmix_data_array_t)
31 Array of **pmix_info_t** declaring each method and value supported by this application.
32 **PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)

1 Regular expression identifying nodes that are to be provisioned.

2 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvning" (char*)

3 Name of the image that is to be provisioned.

4 **PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

5 Indicate that the job can be pre-empted.



6 Description

7 Execute a job control action on behalf of a client. The *targets* array identifies the processes to

8 which the requested job control action is to be applied. A **NULL** value can be used to indicate all

9 processes in the caller's namespace. The use of **PMIX_RANK_WILDARD** can also be used to

10 indicate that all processes in the given namespace are to be included.

11 The directives are provided as **pmix_info_t** structures in the *directives* array. The callback

12 function provides a *status* to indicate whether or not the request was granted, and to provide some

13 information as to the reason for any denial in the **pmix_info_cbfnc_t** array of

14 **pmix_info_t** structures.

15 11.2.22 pmix_server_monitor_fn_t

16 Summary

17 Request that a client be monitored for activity.

18 Format

PMIx v2.0

```

19 typedef pmix_status_t (*pmix_server_monitor_fn_t) (
20     const pmix_proc_t *requestor,
21     const pmix_info_t *monitor, pmix_status_t error,
22     const pmix_info_t directives[], size_t ndirs,
23     pmix_info_cbfnc_t cbfunc, void *cbdata);

```

24 **IN requestor**

25 **pmix_proc_t** structure of requesting process (handle)

26 **IN monitor**

27 **pmix_info_t** identifying the type of monitor being requested (handle)

28 **IN error**

29 Status code to use in generating event if alarm triggers (integer)

30 **IN directives**

31 Array of info structures (array of handles)

32 **IN ndirs**

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

1 **IN** `cbfunc`
2 Callback function `pmix_op_cbfunc_t` (function reference)

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

5 Returns one of the following:

- 6 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
7 will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function
8 prior to returning from the API.
- 9 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
10 returned `success` - the `cbfunc` will not be called
- 11 • a PMIx error constant indicating either an error in the input or that the request was immediately
12 processed and failed - the `cbfunc` will not be called

13 This entry point is only called for monitoring requests that are not directly supported by the PMIx
14 server library itself.

▼----- Required Attributes -----▼

15 If supported by the PMIx server library, then the library must not pass any supported attributes to
16 the host environment. Any attributes provided by the client that are not directly supported by the
17 server library must be passed to the host environment if it provides this module entry. In addition,
18 the following attributes are required to be included in the passed `info` array:

19 **PMIX_USERID** "pmix.euid" (uint32_t)
20 Effective user id.

21 **PMIX_GRPID** "pmix.egid" (uint32_t)
22 Effective group id.

23
24 Host environments are not required to support any specific monitoring attributes.



▼----- Optional Attributes -----▼

25 The following attributes may be implemented by a host environment.

26 **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
27 Provide a string identifier for this request.

28 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
29 Identifier to be canceled (**NULL** means cancel all monitoring for this process).

30 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
31 The application desires to control the response to a monitoring event.

32 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)

1 Register to have the PMIx server monitor the requestor for heartbeats.

2 **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)

3 Time in seconds before declaring heartbeat missed.

4 **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)

5 Number of heartbeats that can be missed before generating the event.

6 **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)

7 Register to monitor file for signs of life.

8 **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)

9 Monitor size of given file is growing to determine if the application is running.

10 **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)

11 Monitor time since last access of given file to determine if the application is running.

12 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)

13 Monitor time since last modified of given file to determine if the application is running.

14 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)

15 Time in seconds between checking the file.

16 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)

17 Number of file checks that can be missed before generating the event.



18 **Description**

19 Request that a client be monitored for activity.



Advice to PMIx server hosts

20 If this module entry is provided and called by the PMIx server library, then the host environment
21 must either provide the requested services or return **PMIX_ERR_NOT_SUPPORTED** to the
22 provided *cbfunc*.



23 **11.2.23 pmix_server_get_cred_fn_t**

24 **Summary**

25 Request a credential from the host environment

1
PMIx v3.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_get_cred_fn_t) (
3     const pmix_proc_t *proc,
4     const pmix_info_t directives[],
5     size_t ndirs,
6     pmix_credential_cbfunc_t cbfunc,
7     void *cbdata);

```

C

- 8 **IN** `proc`
- 9 `pmix_proc_t` structure of requesting process (handle)
- 10 **IN** `directives`
- 11 Array of info structures (array of handles)
- 12 **IN** `ndirs`
- 13 Number of elements in the *info* array (integer)
- 14 **IN** `cbfunc`
- 15 Callback function to return the credential (`pmix_credential_cbfunc_t` function
- 16 reference)
- 17 **IN** `cbdata`
- 18 Data to be passed to the callback function (memory reference)

19 Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. In the event
20 the function returns an error, the *cbfunc* will not be called.

Required Attributes

21 If the PMIx library does not itself provide the requested credential, then it is required to pass any
22 attributes provided by the client to the host environment for processing. In addition, it must include
23 the following attributes in the passed *info* array:

- 24 **PMIX_USERID** "pmix.euid" (`uint32_t`)
- 25 Effective user id.
- 26 **PMIX_GRPID** "pmix.egid" (`uint32_t`)
- 27 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

1
PMIx v3.0

Format

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

- 9 **IN** **proc**
10 **pmix_proc_t** structure of requesting process (handle)
- 11 **IN** **cred**
12 Pointer to **pmix_byte_object_t** containing the credential (handle)
- 13 **IN** **directives**
14 Array of info structures (array of handles)
- 15 **IN** **ndirs**
16 Number of elements in the *info* array (integer)
- 17 **IN** **cbfunc**
18 Callback function to return the result (**pmix_validation_cbfunc_t** function
19 reference)
- 20 **IN** **cbdata**
21 Data to be passed to the callback function (memory reference)

22 Returns one of the following:

- 23 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
24 will be returned in the provided *cbfunc*
- 25 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
26 returned *success* - the *cbfunc* will not be called
- 27 • a PMIx error constant indicating either an error in the input or that the request was immediately
28 processed and failed - the *cbfunc* will not be called

Required Attributes

29 If the PMIx library does not itself validate the credential, then it is required to pass any attributes
30 provided by the client to the host environment for processing. In addition, it must include the
31 following attributes in the passed *info* array:

- 32 **PMIX_USERID** "pmix.euid" (uint32_t)
33 Effective user id.
- 34 **PMIX_GRPID** "pmix.egid" (uint32_t)
35 Effective group id.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

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.

1
PMIx v3.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_iof_fn_t) (
3     const pmix_proc_t procs[], size_t nprocs,
4     const pmix_info_t directives[], size_t ndirs,
5     pmix_iof_channel_t channels,
6     pmix_op_cbfunc_t cbfunc, void *cbdata);

```

C

- 7 **IN procs**
Array `pmix_proc_t` identifiers whose IO is being requested (handle)
- 8
- 9 **IN nprocs**
Number of elements in *procs* (`size_t`)
- 10
- 11 **IN directives**
Array of `pmix_info_t` structures further defining the request (array of handles)
- 12
- 13 **IN ndirs**
Number of elements in the *info* array (integer)
- 14
- 15 **IN channels**
Bitmask identifying the channels to be forwarded (`pmix_iof_channel_t`)
- 16
- 17 **IN cbfunc**
Callback function `pmix_op_cbfunc_t` (function reference)
- 18
- 19 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 20

21 Returns one of the following:

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

Required Attributes

29 The following attributes are required to be included in the passed *info* array:

- 30 **PMIX_USERID** "pmix.euid" (`uint32_t`)
Effective user id.
- 31
- 32 **PMIX_GRPID** "pmix.egid" (`uint32_t`)
Effective group id.
- 33

1
2
3
4
5
6
7
8
9
10
11

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 -----▼

12
13
14
15
16
17
18
19
20
21

- 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 -----▼

26
27
28

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.



1 11.2.26 pmix_server_stdin_fn_t

2 Summary

3 Pass standard input data to the host environment for transmission to specified recipients.

4 Format

PMIx v3.0

C

```
5 typedef pmix_status_t (*pmix_server_stdin_fn_t) (  
6     const pmix_proc_t *source,  
7     const pmix_proc_t targets[],  
8     size_t ntargets,  
9     const pmix_info_t directives[],  
10    size_t ndirs,  
11    const pmix_byte_object_t *bo,  
12    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

13 **IN source**

14 [pmix_proc_t](#) structure of source process (handle)

15 **IN targets**

16 Array of [pmix_proc_t](#) target identifiers (handle)

17 **IN ntargets**

18 Number of elements in the *targets* array (integer)

19 **IN directives**

20 Array of info structures (array of handles)

21 **IN ndirs**

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

23 **IN bo**

24 Pointer to [pmix_byte_object_t](#) containing the payload (handle)

25 **IN cbfunc**

26 Callback function [pmix_op_cbfunc_t](#) (function reference)

27 **IN cbdata**

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

29 Returns one of the following:

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

Required Attributes

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*.

CHAPTER 12

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.

12.1 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 PR RTE command line:

```
$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp
```

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.

- 1 • Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are
2 involved in an operation are translated by their PMIx server into their *native* identifier prior to the
3 operation being passed to the host environment. For example, an application can define a process
4 group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the
5 group id of *foo*. If the application subsequently calls the **PMIx_Fence** API with a process
6 identifier of {foo, PMIX_RANK_WILDCARD}, the PMIx server will replace that identifier
7 with an array consisting of {210456, 0} and {210456, 1} - the host-assigned identifiers of the
8 participating processes - prior to passing the request up to the host environment

9 The two concepts do, however, overlap in one specific area. Process *groups* are included in the
10 process *set* information returned by calls to **PMIx_Query_info_nb**. Thus, a *process group* can
11 effectively be considered an extended version of a *process set* that adds dynamic definition and
12 operational context to the *process set* concept.

Advice to PMIx library implementers

13 PMIx implementations are required to include all active *group* identifiers in the returned list of
14 process *set* names provided in response to the appropriate **PMIx_Query_info_nb** call.

12.2 Process Groups

16 PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for
17 purposes such as passing events or participating in PMIx fence operations. As with processes that
18 assemble via **PMIx_Connect**, each member of the group is provided with both the job-level
19 information of any other namespace represented in the group, and the contact information for all
20 group members. However, *groups* differ from **PMIx_Connect** assemblages in the following key
21 areas:

- 22 • Relation to the host environment
 - 23 – Calls to **PMIx_Connect** are relayed to the host environment. This means that the host RM
24 should treat the failure of any process in the specified assemblage as a reportable event and
25 take appropriate action. However, the environment is not required to define a new identifier for
26 the connected assemblage or any of its member processes, nor does it define a new rank for
27 each process within that assemblage. In addition, the PMIx server does not provide any
28 tracking support for the assemblage. Thus, the caller is responsible for addressing members of
29 the connected assemblage using their RM-provided identifiers.

- 1 – Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed,
2 the server creates a tracker that associates the specified processes with the user-provided group
3 identifier, and assigns a new *group rank* based on their relative position in the array of
4 processes provided in the call to `PMIx_Group_construct`. Members of the group can
5 subsequently utilize the group identifier in PMIx function calls to address the group’s
6 members, using either `PMIX_RANK_WILDCARD` to refer to all of them or the group-level
7 rank of specific members. The PMIx server will translate the specified processes into their
8 RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment
9 has no visibility into the group’s existence or membership.

Advice to users

10 User-provided group identifiers must be distinct from anything provided by the RM so as to
11 avoid collisions between group identifiers and RM-assigned namespaces. This can usually be
12 accomplished through the use of an application-specific prefix – e.g., “myapp-foo”

• Construction procedure

- 13 – `PMIx_Connect` calls require that every process call the API before completing – i.e., it is
14 modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a
15 given application thread can only be involved in one connect/accept operation at a time, and is
16 blocked in that operation until all specified processes participate. In addition, there is no
17 provision for replacing processes in the assemblage due to failure to participate, nor a
18 mechanism by which a process might decline participation.
19 – PMIx Groups are designed to be more flexible in their construction procedure by relaxing
20 these constraints. While a standard blocking form of constructing groups is provided, the event
21 notification system is utilized to provide a designated *group leader* with the ability to replace
22 participants that fail to participate within a given timeout period. This provides a mechanism
23 by which the application can, if desired, replace members on-the-fly or allow the group to
24 proceed with partial membership. In such cases, the final group membership is returned to all
25 participants upon completion of the operation.
26 – PMIx Groups are designed to be more flexible in their construction procedure by relaxing
27 these constraints. While a standard blocking form of constructing groups is provided, the event
28 notification system is utilized to provide a designated *group leader* with the ability to replace
29 participants that fail to participate within a given timeout period. This provides a mechanism
30 by which the application can, if desired, replace members on-the-fly or allow the group to
31 proceed with partial membership. In such cases, the final group membership is returned to all
32 participants upon completion of the operation.

27 Additionally, PMIx supports dynamic definition of group membership based on an invite/join
28 model. A process can asynchronously initiate construction of a group of any processes via the
29 `PMIx_Group_invite` function call. Invitations are delivered via a PMIx event (using the
30 `PMIX_GROUP_INVITED` event) to the invited processes which can then either accept or
31 decline the invitation using the `PMIx_Group_join` API. The initiating process tracks
32 responses by registering for the events generated by the call to `PMIx_Group_join`,
33 timeouts, or process terminations, optionally replacing processes that decline the invitation,
34 fail to respond in time, or terminate without responding. Upon completion of the operation,
35 the final list of participants is communicated to each member of the new group.

• Destruct procedure

- 1 – Processes that assemble via **PMIx_Connect** must all depart the assemblage together – i.e.,
2 no member can depart the assemblage while leaving the remaining members in it. Even the
3 non-blocking form of **PMIx_Disconnect** retains this requirement in that members remain
4 a part of the assemblage until all members have called **PMIx_Disconnect_nb**
- 5 – Members of a PMIx Group may depart the group at any time via the **PMIx_Group_leave**
6 API. Other members are notified of the departure via the **PMIX_GROUP_LEFT** event to
7 distinguish such events from those reporting process termination. This leaves the remaining
8 members free to continue group operations. The **PMIx_Group_destruct** operation offers
9 a collective method akin to **PMIx_Disconnect** for deconstructing the entire group.

10 Note that applications supporting dynamic group behaviors such as asynchronous departure
11 take responsibility for ensuring global consistency in the group definition prior to executing
12 group collective operations - i.e., it is the application's responsibility to either ensure that
13 knowledge of the current group membership is globally consistent across the participants, or to
14 register for appropriate events to deal with the lack of consistency during the operation.

15 In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected*
16 when constructed via **PMIx_Connect** . The relevant APIs are explained below.

Advice to users

17 The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs
18 must register for the corresponding events. Failure to do so will likely lead to operational failures.
19 Users are recommended to utilize the **PMIX_TIMEOUT** directive (or retain an internal timer) on
20 calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have
21 not registered for required events will never respond.

22 12.2.1 **PMIx_Group_construct**

23 **Summary**

24 Construct a PMIx process group

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_construct(const char grp[],
4                     const pmix_proc_t procs[], size_t nprocs,
5                     const pmix_info_t directives[], size_t ndirs,
6                     pmix_info_t **results, size_t *nresults)

```

C

- 7 **IN grp**
8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
9 group identifier (string)
- 10 **IN procs**
11 Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes
12 (array of handles)
- 13 **IN nprocs**
14 Number of elements in the *procs* array (**size_t**)
- 15 **IN directives**
16 Array of **pmix_info_t** structures (array of handles)
- 17 **IN ndirs**
18 Number of elements in the *directives* array (**size_t**)
- 19 **INOUT results**
20 Pointer to a location where the array of **pmix_info_t** describing the results of the
21 operation is to be returned (pointer to handle)
- 22 **INOUT nresults**
23 Pointer to a **size_t** location where the number of elements in *results* is to be returned
24 (memory reference)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
28 operation
- 29 • a PMIx error constant indicating either an error in the input or that the request failed to be
30 completed

Required Attributes

31 The following attributes are *required* to be supported by all PMIx libraries that support this
32 operation:

- 33 **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)
34 This process is the leader of the group
- 35 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

1 Participation is optional - do not return an error if any of the specified processes terminate
2 without having joined. The default is false

3 **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)

4 Group operation only involves local processes. PMIx implementations are *required* to
5 automatically scan an array of group members for local vs remote processes - if only local
6 processes are detected, the implementation need not execute a global collective for the
7 operation unless a context ID has been requested from the host environment. This can result
8 in significant time savings. This attribute can be used to optimize the operation by indicating
9 whether or not only local processes are represented, thus allowing the implementation to
10 bypass the scan. The default is false

11 Host environments that support this operation are *required* to provide the following attributes:

12 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

13 Notify remaining members when another member terminates without first leaving the group.
14 The default is false

15 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

16 Notify remaining members when another member terminates without first leaving the group.
17 The default is false

▲-----▲
▼-----▼ **Optional Attributes** -----▼

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

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

20 Time in seconds before the specified operation should time out (0 indicating infinite) in
21 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
22 the target process from ever exposing its data.

▲-----▲
▼-----▼ **Advice to PMIx library implementers** -----▼

23 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
24 environment due to race condition considerations between completion of the operation versus
25 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
26 directly in the PMIx server library must take care to resolve the race condition and should avoid
27 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
28 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

1 some members fail to participate. The *results* array will contain the final group membership in the
2 latter case. Note that this use-case can cause the operation to hang if the `PMIX_TIMEOUT`
3 attribute is not specified and one or more group members fail to call `PMIX_Group_construct`
4 while continuing to execute. Also, note that no leader or member failed events will be generated
5 during the operation.

6 Processes in a group under construction are not allowed to leave the group until group construction
7 is complete. Upon completion of the construct procedure, each group member will have access to
8 the job-level information of all namespaces represented in the group plus any information posted
9 via `PMIX_Put` (subject to the usual scoping directives) for every group member.

▼ ————— Advice to PMIx library implementers ————— ▼

10 At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related
11 information from each participating namespace plus any information posted by group members via
12 `PMIX_Put` (subject to scoping directives) is available to each member via calls to `PMIX_Get` .



▼ ————— Advice to PMIx server hosts ————— ▼

13 The collective nature of this API generally results in use of a fence-like operation by the backend
14 host environment. Host environments that utilize the array of process participants as a *signature* for
15 such operations may experience potential conflicts should both a `PMIX_Group_construct`
16 and a `PMIX_Fence` operation involving the same participants be simultaneously executed. As
17 PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve
18 any potential conflicts.



19 12.2.2 `PMIX_Group_construct_nb`

20 Summary

21 Non-blocking form of `PMIX_Group_construct`

Format

C

```

2 pmix_status_t
3 PMIx_Group_construct_nb(const char grp[],
4                        const pmix_proc_t procs[], size_t nprocs,
5                        const pmix_info_t directives[], size_t ndirs,
6                        pmix_info_cbfunc_t cbfunc, void *cbdata)

```

C

- 7 **IN grp**
- 8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
- 9 group identifier (string)
- 10 **IN procs**
- 11 Array of **pmix_proc_t** structures containing the PMIx identifiers of the member processes
- 12 (array of handles)
- 13 **IN nprocs**
- 14 Number of elements in the *procs* array (**size_t**)
- 15 **IN directives**
- 16 Array of **pmix_info_t** structures (array of handles)
- 17 **IN ndirs**
- 18 Number of elements in the *directives* array (**size_t**)
- 19 **IN cbfunc**
- 20 Callback function **pmix_info_cbfunc_t** (function reference)
- 21 **IN cbdata**
- 22 Data to be passed to the callback function (memory reference)

23 Returns one of the following:

- 24 • **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
- 25 callback function will be executed upon completion of the operation. Note that the library *must*
- 26 *not* invoke the callback function prior to returning from the API.
- 27 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
- 28 returned *success* - the *cbfunc* will *not* be called
- 29 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
- 30 will *not* be called
- 31 • a non-zero PMIx error constant indicating a reason for the request to have been rejected - the
- 32 *cbfunc* will *not* be called

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

34 constants:

- 35 • **PMIX_SUCCESS** The operation succeeded and all specified members participated.

- 1 ● **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members
2 participated - the final group membership is included in the callback function
- 3 ● **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
4 does not.
- 5 ● a non-zero PMIx error constant indicating a reason for the request's failure

▼----- Required Attributes -----▼

6 PMIx libraries that choose not to support this operation *must* return
7 **PMIX_ERR_NOT_SUPPORTED** when the function is called.

8 The following attributes are *required* to be supported by all PMIx libraries that support this
9 operation:

10 **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)

11 This process is the leader of the group

12 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

13 Participation is optional - do not return an error if any of the specified processes terminate
14 without having joined. The default is false

15 **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)

16 Group operation only involves local processes. PMIx implementations are *required* to
17 automatically scan an array of group members for local vs remote processes - if only local
18 processes are detected, the implementation need not execute a global collective for the
19 operation unless a context ID has been requested from the host environment. This can result
20 in significant time savings. This attribute can be used to optimize the operation by indicating
21 whether or not only local processes are represented, thus allowing the implementation to
22 bypass the scan. The default is false

23 Host environments that support this operation are *required* to provide the following attributes:

24 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

25 Notify remaining members when another member terminates without first leaving the group.
26 The default is false

27 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

28 Notify remaining members when another member terminates without first leaving the group.
29 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**.

12.2.3 PMIx_Group_destruct

Summary

Destruct a PMIx process group

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_destruct(const char grp[],
4                     const pmix_info_t directives[], size_t ndirs)

```

C

- 5 **IN grp**
- 6 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
- 7 identifier of the group to be destructed (string)
- 8 **IN directives**
- 9 Array of **pmix_info_t** structures (array of handles)
- 10 **IN ndirs**
- 11 Number of elements in the *directives* array (**size_t**)

12 Returns one of the following:

- 13 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 14 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
- 15 operation
- 16 • a PMIx error constant indicating either an error in the input or that the request failed to be
- 17 completed

Required Attributes

18 For implementations and host environments that support the operation, there are no identified
19 required attributes for this API.

Optional Attributes

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

- 21 **PMIX_TIMEOUT** "pmix.timeout" (**int**)
- 22 Time in seconds before the specified operation should time out (0 indicating infinite) in
- 23 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
- 24 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.

12.2.4 `PMIx_Group_destruct_nb`

Summary

Non-blocking form of `PMIx_Group_destruct`

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_destruct_nb(const char grp[],
4                       const pmix_info_t directives[], size_t ndirs,
5                       pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN grp**
- 7 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
- 8 identifier of the group to be destructed (string)
- 9 **IN directives**
- 10 Array of **pmix_info_t** structures (array of handles)
- 11 **IN ndirs**
- 12 Number of elements in the *directives* array (**size_t**)
- 13 **IN cbfunc**
- 14 Callback function **pmix_op_cbfunc_t** (function reference)
- 15 **IN cbdata**
- 16 Data to be passed to the callback function (memory reference)

17 Returns one of the following:

- 18 • **PMIX_SUCCESS**, indicating that the request is being processed - result will be returned in the
- 19 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
- 20 from the API.
- 21 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
- 22 returned *success* - the *cbfunc* will *not* be called
- 23 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
- 24 will *not* be called
- 25 • a PMIx error constant indicating either an error in the input or that the request was immediately
- 26 processed and failed - the *cbfunc* will *not* be called

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

28 constants:

- 29 • **PMIX_SUCCESS** The operation was successfully completed
- 30 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
- 31 does not.
- 32 • 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`.

12.2.5 PMIx_Group_invite

Summary

Asynchronously construct a PMIx process group

Format

C

```

2 pmix_status_t
3 PMIx_Group_invite(const char grp[],
4                   const pmix_proc_t procs[], size_t nprocs,
5                   const pmix_info_t directives[], size_t ndirs,
6                   pmix_info_t **results, size_t *nresult)

```

C

- 7 **IN grp**
8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
9 group identifier (string)
- 10 **IN procs**
11 Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be
12 invited (array of handles)
- 13 **IN nprocs**
14 Number of elements in the *procs* array (**size_t**)
- 15 **IN directives**
16 Array of **pmix_info_t** structures (array of handles)
- 17 **IN ndirs**
18 Number of elements in the *directives* array (**size_t**)
- 19 **INOUT results**
20 Pointer to a location where the array of **pmix_info_t** describing the results of the
21 operation is to be returned (pointer to handle)
- 22 **INOUT nresults**
23 Pointer to a **size_t** location where the number of elements in *results* is to be returned
24 (memory reference)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
28 operation
- 29 • a PMIx error constant indicating either an error in the input or that the request failed to be
30 completed

Required Attributes

31 The following attributes are *required* to be supported by all PMIx libraries that support this
32 operation:

- 33 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (**bool**)
34 Participation is optional - do not return an error if any of the specified processes terminate
35 without having joined. The default is false

1 Host environments that support this operation are *required* to provide the following attributes:

2 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

3 Notify remaining members when another member terminates without first leaving the group.
4 The default is false

5 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

6 Notify remaining members when another member terminates without first leaving the group.
7 The default is false



▼----- Optional Attributes -----▼

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

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

10 Time in seconds before the specified operation should time out (0 indicating infinite) in
11 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
12 the target process from ever exposing its data.



▼----- Advice to PMIx library implementers -----▼

13 We recommend that implementation of the **PMIX_TIMEOUT** attribute be left to the host
14 environment due to race condition considerations between completion of the operation versus
15 internal timeout in the PMIx server library. Implementers that choose to support **PMIX_TIMEOUT**
16 directly in the PMIx server library must take care to resolve the race condition and should avoid
17 passing **PMIX_TIMEOUT** to the host environment so that multiple competing timeouts are not
18 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.

1 Failure of the inviting process at any time will cause a `PMIX_GROUP_LEADER_FAILED` event to
2 be delivered to all participants so they can optionally declare a new leader. A new leader is
3 identified by providing the `PMIX_GROUP_LEADER` attribute in the results array in the return of
4 the event handler. Only one process is allowed to return that attribute, declaring itself as the new
5 leader. Results of the leader selection will be communicated to all participants via a
6 `PMIX_GROUP_LEADER_SELECTED` event identifying the new leader. If no leader was selected,
7 then the status code provided in the event handler will provide an error value so the participants can
8 take appropriate action.

9 12.2.6 `PMIx_Group_invite_nb`

10 Summary

11 Non-blocking form of `PMIx_Group_invite`

12 Format

PMIx v4.0

C

```
13 pmix_status_t  
14 PMIx_Group_invite_nb(const char grp[],  
15                     const pmix_proc_t procs[], size_t nprocs,  
16                     const pmix_info_t directives[], size_t ndirs,  
17                     pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

- 18 **IN** `grp`
19 NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the
20 group identifier (string)
- 21 **IN** `procs`
22 Array of `pmix_proc_t` structures containing the PMIx identifiers of the processes to be
23 invited (array of handles)
- 24 **IN** `nprocs`
25 Number of elements in the `procs` array (`size_t`)
- 26 **IN** `directives`
27 Array of `pmix_info_t` structures (array of handles)
- 28 **IN** `ndirs`
29 Number of elements in the `directives` array (`size_t`)
- 30 **IN** `cbfunc`
31 Callback function `pmix_info_cbfunc_t` (function reference)
- 32 **IN** `cbdata`
33 Data to be passed to the callback function (memory reference)

34 Returns one of the following:

- 1 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
2 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
3 from the API.
- 4 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
5 returned *success* - the *cbfunc* will *not* be called
- 6 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
7 will *not* be called
- 8 • a PMIx error constant indicating either an error in the input or that the request was immediately
9 processed and failed - the *cbfunc* will *not* be called

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

- 12 • **PMIX_SUCCESS** The operation succeeded and all specified members participated.
- 13 • **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members
14 participated - the final group membership is included in the callback function
- 15 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
16 does not.
- 17 • a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

18 The following attributes are *required* to be supported by all PMIx libraries that support this
19 operation:

20 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

21 Participation is optional - do not return an error if any of the specified processes terminate
22 without having joined. The default is false

23 Host environments that support this operation are *required* to provide the following attributes:

24 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

25 Notify remaining members when another member terminates without first leaving the group.
26 The default is false

27 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

28 Notify remaining members when another member terminates without first leaving the group.
29 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**.

12.2.7 PMIx_Group_join

Summary

Accept an invitation to join a PMIx process group

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_join(const char grp[],
4                 const pmix_proc_t *leader,
5                 pmix_group_opt_t opt,
6                 const pmix_info_t directives[], size_t ndirs,
7                 pmix_info_t **results, size_t *nresult)

```

C

- 8 **IN grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)
- 10 **IN leader**
Process that generated the invitation (handle)
- 12 **IN opt**
Accept or decline flag (**pmix_group_opt_t**)
- 14 **IN directives**
Array of **pmix_info_t** structures (array of handles)
- 16 **IN ndirs**
Number of elements in the *directives* array (**size_t**)
- 18 **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)
- 20 **INOUT nresults**
Pointer to a **size_t** location where the number of elements in *results* is to be returned (memory reference)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation
- 28 • a PMIx error constant indicating either an error in the input or that the request failed to be completed

Required Attributes

31 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.

1 Calling this function causes the inviting process (aka the *group leader*) to be notified that the
2 process has either accepted or declined the request. The blocking form of the API will return once
3 the group has been completely constructed or the group’s construction has failed (as described
4 below) – likewise, the callback function of the non-blocking form will be executed upon the same
5 conditions.

6 Failure of the leader during the call to `PMIx_Group_join` will cause a
7 `PMIX_GROUP_LEADER_FAILED` event to be delivered to all invited participants so they can
8 optionally declare a new leader. A new leader is identified by providing the
9 `PMIX_GROUP_LEADER` attribute in the results array in the return of the event handler. Only one
10 process is allowed to return that attribute, declaring itself as the new leader. Results of the leader
11 selection will be communicated to all participants via a `PMIX_GROUP_LEADER_SELECTED`
12 event identifying the new leader. If no leader was selected, then the status code provided in the
13 event handler will provide an error value so the participants can take appropriate action.

14 Any participant that returns `PMIX_GROUP_CONSTRUCT_ABORT` from the leader failed event
15 handler will cause all participants to receive an event notifying them of that status. Similarly, the
16 leader may elect to abort the procedure by either returning `PMIX_GROUP_CONSTRUCT_ABORT`
17 from the handler assigned to the `PMIX_GROUP_INVITE_ACCEPTED` or
18 `PMIX_GROUP_INVITE_DECLINED` codes, or by generating an event for the abort code. Abort
19 events will be sent to all invited participants.

20 12.2.8 `PMIx_Group_join_nb`

21 Summary

22 Non-blocking form of `PMIx_Group_join`

23 Format

PMIx v4.0

```
24 pmix_status_t  
25 PMIx_Group_join_nb(const char grp[],  
26                   const pmix_proc_t *leader,  
27                   pmix_group_opt_t opt,  
28                   const pmix_info_t directives[], size_t ndirs,  
29                   pmix_info_cbfunc_t cbfunc, void *cbdata)
```

30 IN `grp`

31 `NULL`-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the
32 group identifier (string)

33 IN `leader`

34 Process that generated the invitation (handle)

1 **IN** **opt**
 2 Accept or decline flag (`pmix_group_opt_t`)
 3 **IN** **directives**
 4 Array of `pmix_info_t` structures (array of handles)
 5 **IN** **ndirs**
 6 Number of elements in the *directives* array (`size_t`)
 7 **IN** **cbfunc**
 8 Callback function `pmix_info_cbfunc_t` (function reference)
 9 **IN** **cbdata**
 10 Data to be passed to the callback function (memory reference)

11 Returns one of the following:

- 12 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
 13 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
 14 from the API.
- 15 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
 16 returned *success* - the *cbfunc* will *not* be called
- 17 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
 18 will *not* be called
- 19 • a PMIx error constant indicating either an error in the input or that the request was immediately
 20 processed and failed - the *cbfunc* will *not* be called

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

- 23 • **PMIX_SUCCESS** The operation succeeded and group membership is in the callback function
 24 parameters
- 25 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
 26 does not.
- 27 • a non-zero PMIx error constant indicating a reason for the request's failure

▼----- Required Attributes -----▼

28 There are no identified required attributes for implementers.

▲-----

▼----- Optional Attributes -----▼

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

30 **PMIX_TIMEOUT** "`pmix.timeout`" (`int`)
 31 Time in seconds before the specified operation should time out (0 indicating infinite) in
 32 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
 33 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`.

12.2.9 PMIx_Group_leave

Summary

Leave a PMIx process group

Format

PMIx v4.0

```
pmix_status_t
PMIx_Group_leave(const char grp[],
                 const pmix_info_t directives[], size_t ndirs)
```

IN grp

NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

IN directives

Array of `pmix_info_t` structures (array of handles)

IN ndirs

Number of elements in the *directives* array (`size_t`)

Returns 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. All PMIx-based collectives such as `PMIx_Fence` in action across the group will automatically be adjusted if the collective was called with the `PMIX_GROUP_FT_COLLECTIVE` attribute (default is false) – otherwise, the standard error return behavior for that collective will be executed.

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.

12.2.10 `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:

- 1 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
2 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
3 from the API.
- 4 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
5 returned *success* - the *cbfunc* will *not* be called
- 6 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
7 will *not* be called
- 8 • a PMIx error constant indicating either an error in the input or that the request was immediately
9 processed and failed - the *cbfunc* will *not* be called

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

- 12 • **PMIX_SUCCESS** The operation succeeded - i.e., the **PMIX_GROUP_LEFT** event was
13 generated
- 14 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx library supports this operation, the host RM
15 does not.
- 16 • a non-zero PMIx error constant indicating a reason for the request's failure

17  **Required Attributes**
There are no identified required attributes for implementers.

18 **Description**

19 Non-blocking version of the **PMIx_Group_leave** operation. The callback function will be
20 called once the event has been locally generated and is not indicative of remote receipt.

APPENDIX A

Acknowledgements

1 This document represents the work of many people who have contributed to the PMIx community.
2 Without the hard work and dedication of these people this document would not have been possible.
3 The sections below list some of the active participants and organizations in the various PMIx
4 standard iterations.

5 **A.1 Version 3.0**

6 The following list includes some of the active participants in the PMIx v3 standardization process.

- 7 • Ralph H. Castain, Andrew Friedley, Brandon Yates
- 8 • Joshua Hursey
- 9 • Aurelien Bouteiller and George Bosilca
- 10 • Dirk Schubert
- 11 • Kevin Harms

12 The following institutions supported this effort through time and travel support for the people listed
13 above.

- 14 • Intel Corporation
- 15 • IBM, Inc.
- 16 • University of Tennessee, Knoxville
- 17 • The Exascale Computing Project, an initiative of the US Department of Energy
- 18 • National Science Foundation
- 19 • Argonne National Laboratory
- 20 • Allinea (ARM)

1 **A.2 Version 2.0**

2 The following list includes some of the active participants in the PMIx v2 standardization process.

3 • Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm
4 and Terry Wilmarth

5 • Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi

6 • Aurelien Bouteiller and George Bosilca

7 • Artem Polyakov, Igor Ivanov and Boris Karasev

8 • Gilles Gouaillardet

9 • Michael A Raymond and Jim Stoffel

10 • Dirk Schubert

11 • Moe Jette

12 • Takahiro Kawashima and Shinji Sumimoto

13 • Howard Pritchard

14 • David Beer

15 • Brice Goglin

16 • Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt

17 • Adam Moody and Martin Schulz

18 • Ryan Grant and Stephen Olivier

19 • Michael Karo

20 The following institutions supported this effort through time and travel support for the people listed
21 above.

22 • Intel Corporation

23 • IBM, Inc.

24 • University of Tennessee, Knoxville

25 • The Exascale Computing Project, an initiative of the US Department of Energy

26 • National Science Foundation

27 • Mellanox, Inc.

28 • Research Organization for Information Science and Technology

29 • HPE Co.

- 1 • Allinea (ARM)
- 2 • SchedMD, Inc.
- 3 • Fujitsu Limited
- 4 • Los Alamos National Laboratory
- 5 • Adaptive Solutions, Inc.
- 6 • INRIA
- 7 • Oak Ridge National Laboratory
- 8 • Lawrence Livermore National Laboratory
- 9 • Sandia National Laboratory
- 10 • Altair

11 **A.3 Version 1.0**

12 The following list includes some of the active participants in the PMIx v1 standardization process.

- 13 • Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- 14 • Joshua Hursey and David Solt
- 15 • Aurelien Bouteiller and George Bosilca
- 16 • Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- 17 • Gilles Gouaillardet
- 18 • Gary Brown
- 19 • Moe Jette

20 The following institutions supported this effort through time and travel support for the people listed
21 above.

- 22 • Intel Corporation
- 23 • IBM, Inc.
- 24 • University of Tennessee, Knoxville
- 25 • Mellanox, Inc.
- 26 • Research Organization for Information Science and Technology
- 27 • Adaptive Solutions, Inc.
- 28 • 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

- application, [7](#), [9](#), [67](#), [68](#), [117](#), [125](#), [178](#), [233](#), [235](#)
 - Defintion, [12](#)
- host environment
 - Defintion, [13](#)
- job, [7](#), [9](#), [67–69](#), [117](#), [125](#), [127](#), [178](#), [228](#), [229](#), [232](#), [233](#), [235](#), [245](#), [247](#)
 - Defintion, [12](#)
- namespace
 - Defintion, [12](#)
- PMIx_Abort, [5](#), [27](#), [147](#), [257](#), [258](#)
 - Defintion, [146](#)
- PMIX_ADD_ENVAR
 - Defintion, [80](#)
- PMIX_ADD_HOST, [149](#), [153](#), [271](#)
 - Defintion, [75](#)
- PMIX_ADD_HOSTFILE, [149](#), [153](#), [271](#)
 - Defintion, [75](#)
- PMIX_ALLOC_BANDWIDTH, [81](#), [181](#), [184](#), [246](#), [291](#)
 - Defintion, [81](#)
- PMIX_ALLOC_CPU_LIST, [181](#), [184](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_DIRECTIVE, [61](#)
- PMIx_Alloc_directive_string, [7](#)
 - Defintion, [103](#)
- pmix_alloc_directive_t, [42](#), [61](#), [103](#), [289](#)
 - Defintion, [42](#)
- PMIX_ALLOC_EXTEND, [42](#)
- PMIX_ALLOC_EXTERNAL, [42](#)
- PMIX_ALLOC_ID, [180](#), [183](#), [290](#)
 - Defintion, [80](#)
- PMIX_ALLOC_MEM_SIZE, [181](#), [184](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK, [181](#), [184](#), [246](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK_ENDPTS, [81](#), [181](#), [182](#), [184](#), [185](#), [246](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK_ENDPTS_NODE, [182](#), [185](#), [246](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK_ID, [81](#), [181](#), [184](#), [246](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK_PLANE, [81](#), [181](#), [182](#), [184](#), [185](#), [246](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK_QOS, [81](#), [181](#), [182](#), [184](#), [246](#), [291](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NETWORK_SEC_KEY, [81](#), [181](#), [182](#), [184](#), [185](#), [246](#), [291](#)
 - Defintion, [82](#)
- PMIX_ALLOC_NETWORK_TYPE, [81](#), [181](#), [182](#), [184](#), [185](#), [246](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NEW, [42](#)
- PMIX_ALLOC_NODE_LIST, [181](#), [184](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NUM_CPU_LIST, [181](#), [184](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NUM_CPUS, [181](#), [184](#), [290](#)
 - Defintion, [81](#)
- PMIX_ALLOC_NUM_NODES, [181](#), [183](#), [290](#)
 - Defintion, [80](#)

PMIX_ALLOC_REAQUIRE, 42
 PMIX_ALLOC_RELEASE, 42
 PMIX_ALLOC_TIME, 181, 184, 246, 290
 Defintion, 81
 PMIX_ALLOCATED_NODELIST, 230
 Defintion, 67
 PMIx_Allocation_request, 8, 179, 185
 Defintion, 180
 PMIx_Allocation_request_nb, 6, 80, 171, 185
 Defintion, 182
 PMIX_ANL_MAP
 Defintion, 72
 PMIX_APP, 60
 PMIX_APP_CONSTRUCT
 Defintion, 48
 PMIX_APP_CREATE
 Defintion, 48
 PMIX_APP_DESTRUCT
 Defintion, 48
 PMIX_APP_FREE
 Defintion, 48
 PMIX_APP_INFO, 120, 123, 127, 174
 Defintion, 68
 PMIX_APP_INFO_ARRAY, 69, 236
 Defintion, 68
 PMIX_APP_INFO_CREATE, 8, 9
 Defintion, 49
 PMIX_APP_MAP_REGEX
 Defintion, 73
 PMIX_APP_MAP_TYPE
 Defintion, 73
 PMIX_APP_RANK, 229
 Defintion, 66
 PMIX_APP_SIZE, 127, 229, 235
 Defintion, 69
 pmix_app_t, 8, 9, 47–49, 148, 152, 270
 Defintion, 47
 PMIX_APPEND_ENVAR
 Defintion, 80
 PMIX_APPLDR, 229, 235
 Defintion, 67
 PMIX_APPNUM, 68, 120, 123, 127, 174, 229, 236
 Defintion, 66
 PMIX_ARCH
 Defintion, 66
 PMIX_ATTR_UNDEF
 Defintion, 61
 PMIX_AVAIL_PHYS_MEMORY, 230
 Defintion, 70
 PMIX_BINDTO, 150, 154, 230, 272
 Defintion, 75
 PMIX_BOOL, 60
 PMIX_BUFFER, 60
 PMIX_BYTE, 60
 PMIX_BYTE_OBJECT, 60
 PMIX_BYTE_OBJECT_CREATE
 Defintion, 55
 PMIX_BYTE_OBJECT_DESTRUCT
 Defintion, 54
 PMIX_BYTE_OBJECT_FREE
 Defintion, 55
 PMIX_BYTE_OBJECT_LOAD
 Defintion, 55
 pmix_byte_object_t, 54, 55, 60, 98, 168, 223, 250, 299, 303
 Defintion, 54
 pmix_check_key
 Defintion, 22
 pmix_check_nspace
 Defintion, 23
 pmix_check_procid
 Defintion, 26
 PMIX_CLEANUP_EMPTY, 187, 190
 Defintion, 83
 PMIX_CLEANUP_IGNORE, 187, 190
 Defintion, 83
 PMIX_CLEANUP_LEAVE_TOPDIR, 187, 190
 Defintion, 83
 PMIX_CLEANUP_RECURSIVE, 187, 190
 Defintion, 83
 PMIX_CLIENT_ATTRIBUTES, 10, 175
 Defintion, 85
 PMIX_CLIENT_AVG_MEMORY

Defintion, [70](#)
 PMIX_CLIENT_FUNCTIONS
 Defintion, [85](#)
 PMIX_CLUSTER_ID
 Defintion, [66](#)
 PMIX_COLLECT_DATA, [131](#), [133](#), [260](#)
 Defintion, [71](#)
 PMIX_COLLECTIVE_ALGO, [7](#), [131](#), [134](#),
 [157](#), [160](#), [260](#), [275](#)
 Defintion, [71](#)
 PMIX_COLLECTIVE_ALGO_REQD, [131](#),
 [134](#), [157](#), [160](#), [260](#), [275](#)
 Defintion, [72](#)
 PMIX_COMMAND, [61](#)
 PMIx_Commit, [5](#), [96](#), [118](#), [130](#), [156](#), [244](#),
 [263](#)
 Defintion, [130](#)
 PMIX_COMPRESSED_STRING, [61](#)
 PMIx_Connect, [6](#), [7](#), [20](#), [151](#), [156](#), [158](#), [160](#),
 [162](#), [306–308](#)
 Defintion, [156](#)
 PMIX_CONNECT_MAX_RETRIES, [110](#)
 Defintion, [63](#)
 PMIx_Connect_nb, [6](#), [159](#)
 Defintion, [159](#)
 PMIX_CONNECT_REQUESTED, [20](#)
 PMIX_CONNECT_RETRY_DELAY, [109](#)
 Defintion, [63](#)
 PMIX_CONNECT_SYSTEM_FIRST, [109](#),
 [111](#), [113](#)
 Defintion, [63](#)
 PMIX_CONNECT_TO_SYSTEM, [109](#), [111](#),
 [113](#)
 Defintion, [62](#)
 pmix_connection_cbfunc_t, [282](#)
 Defintion, [96](#)
 PMIX_COSPAWN_APP
 Defintion, [76](#)
 PMIX_CPU_LIST, [151](#), [155](#), [273](#)
 Defintion, [76](#)
 PMIX_CPUS_PER_PROC, [150](#), [155](#), [273](#)
 Defintion, [76](#)
 PMIX_CPUSET
 Defintion, [65](#)
 PMIX_CRED_TYPE, [298](#)
 Defintion, [84](#)
 PMIX_CREDENTIAL
 Defintion, [65](#)
 pmix_credential_cbfunc_t, [221](#), [297](#)
 Defintion, [97](#)
 PMIX_CRYPTO_KEY
 Defintion, [84](#)
 PMIX_DAEMON_MEMORY
 Defintion, [70](#)
 PMIX_DATA_ARRAY, [61](#)
 PMIX_DATA_ARRAY_CONSTRUCT
 Defintion, [30](#), [58](#)
 PMIX_DATA_ARRAY_CREATE
 Defintion, [31](#), [59](#)
 PMIX_DATA_ARRAY_DESTRUCT
 Defintion, [31](#), [59](#)
 PMIX_DATA_ARRAY_FREE
 Defintion, [31](#)
 PMIX_DATA_ARRAY_RELEASE
 Defintion, [59](#)
 pmix_data_array_t, [8](#), [9](#), [30–32](#), [58](#), [59](#), [61](#),
 [77](#), [81](#), [176](#), [179](#), [181](#), [184](#),
 [234–237](#), [246](#), [284](#), [290](#)
 Defintion, [30](#), [58](#)
 PMIX_DATA_BUFFER_CONSTRUCT,
 [215](#), [216](#)
 Defintion, [56](#), [212](#)
 PMIX_DATA_BUFFER_CREATE, [215](#),
 [216](#)
 Defintion, [57](#), [211](#)
 PMIX_DATA_BUFFER_DESTRUCT
 Defintion, [57](#), [212](#)
 PMIX_DATA_BUFFER_LOAD
 Defintion, [57](#), [213](#)
 PMIX_DATA_BUFFER_RELEASE
 Defintion, [57](#), [211](#)
 pmix_data_buffer_t, [56–58](#), [211–215](#), [219](#)
 Defintion, [56](#)
 PMIX_DATA_BUFFER_UNLOAD
 Defintion, [58](#), [213](#)
 PMIx_Data_copy, [7](#)

Defintion, [217](#)
 PMIx_Data_copy_payload, [7](#)
 Defintion, [218](#)
 PMIx_Data_pack, [7](#), [215](#)
 Defintion, [214](#)
 PMIx_Data_print, [7](#)
 Defintion, [218](#)
 PMIX_DATA_RANGE, [61](#)
 PMIx_Data_range_string, [7](#)
 Defintion, [102](#)
 pmix_data_range_t, [29](#), [61](#), [102](#), [209](#), [281](#)
 Defintion, [29](#)
 PMIX_DATA_SCOPE, [119](#), [123](#)
 Defintion, [72](#)
 PMIX_DATA_TYPE, [61](#)
 PMIx_Data_type_string, [7](#)
 Defintion, [102](#)
 pmix_data_type_t, [30](#), [31](#), [34](#), [36](#), [38](#), [46](#), [53](#),
 [58–61](#), [102](#), [214](#), [216–218](#)
 Defintion, [60](#)
 PMIx_Data_unpack, [7](#)
 Defintion, [215](#)
 PMIX_DEBUG_APP_DIRECTIVES
 Defintion, [80](#)
 PMIX_DEBUG_JOB
 Defintion, [79](#)
 PMIX_DEBUG_JOB_DIRECTIVES
 Defintion, [80](#)
 PMIX_DEBUG_STOP_IN_INIT
 Defintion, [79](#)
 PMIX_DEBUG_STOP_ON_EXEC
 Defintion, [79](#)
 PMIX_DEBUG_WAIT_FOR_NOTIFY
 Defintion, [79](#)
 PMIX_DEBUG_WAITING_FOR_NOTIFY
 Defintion, [79](#)
 PMIX_DEBUGGER_DAEMONS, [150](#), [154](#),
 [272](#)
 Defintion, [76](#)
 PMIx_Deregister_event_handler, [6](#)
 Defintion, [207](#)
 PMIx_Disconnect, [6](#), [7](#), [20](#), [158](#), [162](#), [164](#),
 [308](#)
 Defintion, [160](#)
 PMIx_Disconnect_nb, [6](#), [164](#), [308](#)
 Defintion, [162](#)
 PMIX_DISPLAY_MAP, [149](#), [154](#), [272](#)
 Defintion, [75](#)
 pmix_dmodex_response_fn_t, [244](#)
 Defintion, [95](#)
 PMIX_DOUBLE, [60](#)
 PMIX_DSTPATH
 Defintion, [63](#)
 PMIX_EMBED_BARRIER, [108](#)
 Defintion, [72](#)
 PMIX_ENUM_VALUE, [10](#), [51](#)
 Defintion, [85](#)
 PMIX_ENVAR, [61](#)
 PMIX_ENVAR_CONSTRUCT
 Defintion, [43](#)
 PMIX_ENVAR_CREATE
 Defintion, [43](#)
 PMIX_ENVAR_DESTRUCT
 Defintion, [43](#)
 PMIX_ENVAR_FREE
 Defintion, [44](#)
 PMIX_ENVAR_LOAD
 Defintion, [44](#)
 pmix_envar_t, [43](#), [44](#), [61](#)
 Defintion, [42](#)
 PMIX_ERR_BAD_PARAM, [19](#)
 PMIX_ERR_COMM_FAILURE, [19](#)
 PMIX_ERR_DATA_VALUE_NOT_FOUND,
 [19](#)
 PMIX_ERR_DEBUGGER_RELEASE, [18](#)
 PMIX_ERR_DUPLICATE_KEY, [20](#)
 PMIX_ERR_EVENT_REGISTRATION, [20](#)
 PMIX_ERR_HANDSHAKE_FAILED, [18](#)
 PMIX_ERR_IN_ERRNO, [19](#)
 PMIX_ERR_INIT, [19](#)
 PMIX_ERR_INVALID_ARG, [19](#)
 PMIX_ERR_INVALID_ARGS, [19](#)
 PMIX_ERR_INVALID_CRED, [18](#)
 PMIX_ERR_INVALID_KEY, [19](#)
 PMIX_ERR_INVALID_KEY_LENGTH, [19](#)
 PMIX_ERR_INVALID_KEYVALP, [19](#)

PMIX_ERR_INVALID_LENGTH, 19
 PMIX_ERR_INVALID_NAMESPACE, 19
 PMIX_ERR_INVALID_NUM_ARGS, 19
 PMIX_ERR_INVALID_NUM_PARSED, 19
 PMIX_ERR_INVALID_OPERATION, 20
 PMIX_ERR_INVALID_SIZE, 19
 PMIX_ERR_INVALID_TERMINATION, 20
 PMIX_ERR_INVALID_VAL, 19
 PMIX_ERR_INVALID_VAL_LENGTH, 19
 PMIX_ERR_JOB_TERMINATED, 20
 PMIX_ERR_LOST_CONNECTION_TO_CLIENT, 19
 PMIX_ERR_LOST_CONNECTION_TO_SERVER, 19
 PMIX_ERR_LOST_PEER_CONNECTION, 19
 PMIX_ERR_NO_PERMISSIONS, 19
 PMIX_ERR_NODE_DOWN, 20
 PMIX_ERR_NODE_OFFLINE, 20
 PMIX_ERR_NOMEM, 19
 PMIX_ERR_NOT_FOUND, 19
 PMIX_ERR_NOT_IMPLEMENTED, 19
 PMIX_ERR_NOT_SUPPORTED, 19
 PMIX_ERR_OUT_OF_RESOURCE, 19
 PMIX_ERR_PACK_FAILURE, 19
 PMIX_ERR_PACK_MISMATCH, 19
 PMIX_ERR_PROC_ABORTED, 18
 PMIX_ERR_PROC_ABORTING, 18
 PMIX_ERR_PROC_CHECKPOINT, 18
 PMIX_ERR_PROC_ENTRY_NOT_FOUND, 18
 PMIX_ERR_PROC_MIGRATE, 18
 PMIX_ERR_PROC_REQUESTED_ABORT, 18
 PMIX_ERR_PROC_RESTART, 18
 PMIX_ERR_READY_FOR_HANDSHAKE, 18
 PMIX_ERR_RESOURCE_BUSY, 19
 PMIX_ERR_SERVER_FAILED_REQUEST, 18
 PMIX_ERR_SERVER_NOT_AVAIL, 19
 PMIX_ERR_SILENT, 18
 PMIX_ERR_SYS_OTHER, 20
 PMIX_ERR_TIMEOUT, 19
 PMIX_ERR_TYPE_MISMATCH, 18
 PMIX_ERR_UNKNOWN_DATA_TYPE, 18
 PMIX_ERR_UNPACK_FAILURE, 19
 PMIX_ERR_UNPACK_INADEQUATE_SPACE, 18
 PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, 19
 PMIX_ERR_UNREACH, 19
 PMIX_ERR_UPDATE_ENDPOINTS, 20
 PMIX_ERR_WOULD_BLOCK, 18
 PMIX_ERROR, 18
 PMIx_Error_string, 6
 Defintion, 101
 PMIX_EVENT_ACTION_COMPLETE, 20
 PMIX_EVENT_ACTION_DEFERRED, 20
 PMIX_EVENT_ACTION_TIMEOUT, 206
 Defintion, 74
 PMIX_EVENT_AFFECTED_PROC, 206, 210
 Defintion, 73
 PMIX_EVENT_AFFECTED_PROCS, 206, 210
 Defintion, 73
 PMIX_EVENT_BASE, 107, 110, 116
 Defintion, 62
 PMIX_EVENT_CUSTOM_RANGE, 206, 210
 Defintion, 73
 PMIX_EVENT_DO_NOT_CACHE
 Defintion, 74
 PMIX_EVENT_HDLR_AFTER, 205
 Defintion, 73
 PMIX_EVENT_HDLR_APPEND, 206
 Defintion, 73
 PMIX_EVENT_HDLR_BEFORE, 205
 Defintion, 73
 PMIX_EVENT_HDLR_FIRST, 205
 Defintion, 73
 PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, 205

Defintion, [73](#)
 PMIX_EVENT_HDLR_LAST, [205](#)
 Defintion, [73](#)
 PMIX_EVENT_HDLR_LAST_IN_CATEGORY,
 [205](#)
 Defintion, [73](#)
 PMIX_EVENT_HDLR_NAME, [205](#)
 Defintion, [73](#)
 PMIX_EVENT_HDLR_PREPEND, [205](#)
 Defintion, [73](#)
 PMIX_EVENT_NO_ACTION_TAKEN, [20](#)
 PMIX_EVENT_NO_TERMINATION
 Defintion, [74](#)
 PMIX_EVENT_NON_DEFAULT, [210](#)
 Defintion, [73](#)
 pmix_event_notification_cbfunc_fn_t, [92, 93](#)
 Defintion, [92](#)
 PMIX_EVENT_PARTIAL_ACTION_TAKEN,
 [20](#)
 PMIX_EVENT_PROXY
 Defintion, [74](#)
 PMIX_EVENT_RETURN_OBJECT, [206](#)
 Defintion, [74](#)
 PMIX_EVENT_SILENT_TERMINATION,
 [206](#)
 Defintion, [74](#)
 PMIX_EVENT_TERMINATE_JOB, [206](#)
 Defintion, [74](#)
 PMIX_EVENT_TERMINATE_NODE, [206](#)
 Defintion, [74](#)
 PMIX_EVENT_TERMINATE_PROC, [206](#)
 Defintion, [74](#)
 PMIX_EVENT_TERMINATE_SESSION,
 [206](#)
 Defintion, [74](#)
 PMIX_EVENT_TEXT_MESSAGE
 Defintion, [74](#)
 PMIX_EVENT_WANT_TERMINATION
 Defintion, [74](#)
 pmix_evhdlr_reg_cbfunc_t, [91, 205](#)
 Defintion, [91](#)
 PMIX_EXISTS, [18](#)
 PMIX_EXIT_CODE
 Defintion, [67](#)
 PMIX_EXTERNAL_ERR_BASE, [21](#)
 PMIx_Fence, [4, 5, 11, 116, 132, 134, 156,](#)
 [158, 162, 244, 258, 261, 306, 312,](#)
 [317, 331](#)
 Defintion, [130](#)
 PMIx_Fence_nb, [5, 10, 89, 134, 258, 261](#)
 Defintion, [132](#)
 PMIx_Finalize, [5, 20, 27, 72, 107, 108, 156,](#)
 [256, 257](#)
 Defintion, [108](#)
 PMIX_FLOAT, [60](#)
 PMIX_FWD_ALL_CHANNELS, [42](#)
 PMIX_FWD_NO_CHANNELS, [42](#)
 PMIX_FWD_STDDIAG, [9](#)
 Defintion, [76](#)
 PMIX_FWD_STDDIAG_CHANNEL, [42](#)
 PMIX_FWD_STDERR, [150, 154, 272, 286](#)
 Defintion, [76](#)
 PMIX_FWD_STDERR_CHANNEL, [42](#)
 PMIX_FWD_STDIN, [150, 154, 272, 286](#)
 Defintion, [75](#)
 PMIX_FWD_STDIN_CHANNEL, [42](#)
 PMIX_FWD_STDOUT, [150, 154, 272, 286](#)
 Defintion, [76](#)
 PMIX_FWD_STDOUT_CHANNEL, [42](#)
 PMIX_GDS_ACTION_COMPLETE, [20](#)
 PMIX_GDS_MODULE, [107, 110, 116](#)
 Defintion, [65](#)
 PMIx_generate_ppn, [6](#)
 Defintion, [226](#)
 PMIx_generate_regex, [6, 232](#)
 Defintion, [225](#)
 PMIx_Get, [3, 5, 7, 32, 61–65, 69, 71–80, 82,](#)
 [83, 107, 119, 121, 123–127, 129,](#)
 [149–151, 153–155, 177–179, 227,](#)
 [230, 253, 271–273, 305, 312](#)
 Defintion, [118](#)
 PMIx_Get_credential, [8, 84, 298](#)
 Defintion, [221](#)
 PMIx_Get_nb, [5, 90](#)
 Defintion, [121](#)
 PMIx_Get_version, [6, 14](#)

Defintion, **105**
 PMIX_GLOBAL, **29**
 PMIX_GLOBAL_RANK, **231**
 Defintion, **66**
 PMIX_GROUP_ACCEPT, **54**
 PMIX_GROUP_ASSIGN_CONTEXT_ID,
 310, 314, 321, 324
 Defintion, **86**
 PMIx_Group_construct, **307, 311, 312, 315**
 Defintion, **308**
 PMIx_Group_construct_nb, **315**
 Defintion, **312**
 PMIX_GROUP_CONTEXT_ID
 Defintion, **86**
 PMIX_GROUP_DECLINE, **54**
 PMIx_Group_destruct, **308, 317, 319**
 Defintion, **315**
 PMIx_Group_destruct_nb, **319**
 Defintion, **317**
 PMIX_GROUP_ENDPT_DATA
 Defintion, **86**
 PMIX_GROUP_FT_COLLECTIVE, **331**
 Defintion, **86**
 PMIX_GROUP_ID
 Defintion, **86**
 PMIx_Group_invite, **307, 322, 323, 325**
 Defintion, **319**
 PMIX_GROUP_INVITE_DECLINE
 Defintion, **86**
 PMIx_Group_invite_nb
 Defintion, **323**
 PMIx_Group_join, **54, 307, 322, 325, 327,**
 328, 330
 Defintion, **325**
 PMIx_Group_join_nb, **325, 330**
 Defintion, **328**
 PMIX_GROUP_LEADER, **309, 311, 314,**
 323, 328
 Defintion, **86**
 PMIx_Group_leave, **308, 331, 332**
 Defintion, **330**
 PMIx_Group_leave_nb
 Defintion, **331**
 PMIX_GROUP_LOCAL_ONLY, **310, 314**
 Defintion, **86**
 PMIX_GROUP_MEMBERSHIP, **311**
 Defintion, **86**
 PMIX_GROUP_NOTIFY_TERMINATION,
 310, 311, 314, 317, 321, 324
 Defintion, **86**
 pmix_group_opt_t, **54, 326, 329**
 Defintion, **54**
 PMIX_GROUP_OPTIONAL, **309, 311, 314,**
 320, 324
 Defintion, **86**
 PMIX_GRPID, **99, 135, 137, 139, 141, 142,**
 144, 175, 180, 183, 186, 189, 192,
 194, 196, 199, 222, 224, 264–269,
 271, 278, 284, 286, 287, 289, 292,
 295, 297, 299, 301, 304
 Defintion, **63**
 pmix_hdlr_reg_cbfunc_t, **165, 167**
 Defintion, **100**
 PMIx_Heartbeat, **6**
 Defintion, **195**
 PMIX_HOST, **149, 153, 271**
 Defintion, **75**
 PMIX_HOST_ATTRIBUTES, **10, 175, 179**
 Defintion, **85**
 PMIX_HOST_FUNCTIONS
 Defintion, **85**
 PMIX_HOSTFILE, **149, 153, 271**
 Defintion, **75**
 PMIX_HOSTNAME, **68, 69, 120, 123, 129,**
 174, 231
 Defintion, **67**
 PMIX_HWLOC_HOLE_KIND
 Defintion, **71**
 PMIX_HWLOC_SHARE_TOPO
 Defintion, **71**
 PMIX_HWLOC_SHMEM_ADDR
 Defintion, **71**
 PMIX_HWLOC_SHMEM_FILE
 Defintion, **71**
 PMIX_HWLOC_SHMEM_SIZE
 Defintion, **71**

PMIX_HWLOC_XML_V1, 230
 Defintion, **71**
 PMIX_HWLOC_XML_V2, 230
 Defintion, **71**
 PMIX_IMMEDIATE, 119, 123
 Defintion, **71**
 PMIX_INDEX_ARGV, 150, 155, 273
 Defintion, **76**
 PMIX_INFO, 60
 PMIX_INFO_ARRAY_END, 40
 pmix_info_cbfunc_t, 87, 90, 173, 183, 186,
 188, 189, 191, 194, 195, 251, 283,
 289, 291, 294, 313, 323, 329
 Defintion, **90**
 PMIX_INFO_CONSTRUCT
 Defintion, **37**
 PMIX_INFO_CREATE, 40, 41
 Defintion, **37**
 PMIX_INFO_DESTRUCT
 Defintion, **37**
 PMIX_INFO_DIRECTIVES, 61
 PMIx_Info_directives_string, 7
 Defintion, **102**
 pmix_info_directives_t, 39, 40, 61, 102
 Defintion, **39**
 PMIX_INFO_FREE
 Defintion, **37**
 PMIX_INFO_IS_END, 8, 9
 Defintion, **41**
 PMIX_INFO_IS_OPTIONAL
 Defintion, **41**
 PMIX_INFO_IS_REQUIRED, 39, 40
 Defintion, **41**
 PMIX_INFO_LOAD
 Defintion, **38**
 PMIX_INFO_OPTIONAL
 Defintion, **40**
 PMIX_INFO_REQD, 40
 PMIX_INFO_REQUIRED, 39
 Defintion, **40**
 pmix_info_t, 4, 7–9, 11, 22, 29, 36–42,
 49–51, 53, 61, 68, 69, 79, 81, 82,
 91–93, 99, 100, 106–109, 114,
 116, 136, 140, 165, 167, 168, 177,
 179–181, 183, 184, 188, 191, 195,
 198, 201, 209, 221, 223, 232,
 234–237, 246, 250–252, 281, 285,
 286, 288, 290, 291, 293, 294, 301,
 309, 311, 313, 316, 318, 320, 323,
 326, 329–331
 Defintion, **36**
 PMIX_INFO_TRUE
 Defintion, **39**
 PMIX_INFO_XFER, 232
 Defintion, **38**
 PMIx_Init, 7, 75, 79, 105, 107, 108, 150,
 154, 255, 272
 Defintion, **105**
 PMIx_Initialized, 5
 Defintion, **104**
 PMIX_INT, 60
 PMIX_INT16, 60
 PMIX_INT32, 60
 PMIX_INT64, 60
 PMIX_INT8, 60
 PMIX_INTERNAL, 29
 PMIX_IOF_BUFFERING_SIZE, 166, 169,
 302
 Defintion, **84**
 PMIX_IOF_BUFFERING_TIME, 166, 169,
 302
 Defintion, **84**
 PMIX_IOF_CACHE_SIZE, 166, 169, 302
 Defintion, **84**
 pmix_iof_cbfunc_t, 165
 Defintion, **99**
 PMIX_IOF_CHANNEL, 61
 PMIx_IOF_channel_string, 8
 Defintion, **103**
 pmix_iof_channel_t, 42, 61, 100, 103, 165,
 250, 301
 Defintion, **42**
 PMIX_IOF_COMPLETE
 Defintion, **84**
 PMIx_IOF_deregister, 8
 Defintion, **167**

PMIX_IOF_DROP_NEWEST, 166, 169, 302
 Defintion, **84**
 PMIX_IOF_DROP_OLDEST, 166, 169, 302
 Defintion, **84**
 PMIx_IOF_pull, 8, 167
 Defintion, **165**
 PMIx_IOF_push, 8
 Defintion, **168**
 PMIX_IOF_TAG_OUTPUT, 166
 Defintion, **84**
 PMIX_IOF_TIMESTAMP_OUTPUT, 166
 Defintion, **84**
 PMIX_IOF_XML_OUTPUT, 166
 Defintion, **84**
 PMIX_JCTRL_CHECKPOINT, 19
 PMIX_JCTRL_CHECKPOINT_COMPLETE,
 19
 PMIX_JCTRL_PREEMPT_ALERT, 19
 PMIX_JOB_CONTINUOUS, 151, 155, 273
 Defintion, **76**
 PMIx_Job_control, 8, 191
 Defintion, **186**
 PMIx_Job_control_nb, 6, 82, 171, 185, 231
 Defintion, **188**
 PMIX_JOB_CTRL_CANCEL, 187, 190,
 293
 Defintion, **82**
 PMIX_JOB_CTRL_CHECKPOINT, 187,
 190, 293
 Defintion, **82**
 PMIX_JOB_CTRL_CHECKPOINT_EVENT,
 187, 190, 293
 Defintion, **82**
 PMIX_JOB_CTRL_CHECKPOINT_METHOD,
 188, 190, 293
 Defintion, **82**
 PMIX_JOB_CTRL_CHECKPOINT_SIGNAL,
 187, 190, 293
 Defintion, **82**
 PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT,
 188, 190, 293
 Defintion, **82**
 PMIX_JOB_CTRL_ID, 186, 189, 293
 Defintion, **82**
 PMIX_JOB_CTRL_KILL, 187, 189, 293
 Defintion, **82**
 PMIX_JOB_CTRL_PAUSE, 186, 189, 293
 Defintion, **82**
 PMIX_JOB_CTRL_PREEMPTIBLE, 188,
 191, 294
 Defintion, **82**
 PMIX_JOB_CTRL_PROVISION, 188, 191,
 293
 Defintion, **82**
 PMIX_JOB_CTRL_PROVISION_IMAGE,
 188, 191, 294
 Defintion, **82**
 PMIX_JOB_CTRL_RESTART, 187, 190,
 293
 Defintion, **82**
 PMIX_JOB_CTRL_RESUME, 187, 189,
 293
 Defintion, **82**
 PMIX_JOB_CTRL_SIGNAL, 187, 190, 293
 Defintion, **82**
 PMIX_JOB_CTRL_TERMINATE, 187,
 190, 293
 Defintion, **82**
 PMIX_JOB_INFO, 119, 123, 127, 174
 Defintion, **68**
 PMIX_JOB_INFO_ARRAY, 8, 69, 235
 Defintion, **68**
 PMIX_JOB_NUM_APPS, 127, 230, 235
 Defintion, **69**
 PMIX_JOB_RECOVERABLE, 151, 155,
 273
 Defintion, **76**
 PMIX_JOB_SIZE, 7, 9, 121, 124, 127, 228,
 235
 Defintion, **69**
 PMIX_JOB_TERM_STATUS
 Defintion, **72**
 PMIX_JOBID, 68, 120, 123, 127, 174, 228,
 235
 Defintion, **66**
 pmix_key_t, 21, 22, 53, 117, 119

Defintion, [21](#)
 PMIX_KVAL, [60](#)
 PMIX_LAUNCH_DIRECTIVE, [20](#)
 PMIX_LAUNCHER
 Defintion, [63](#)
 PMIX_LAUNCHER_READY, [20](#)
 PMIX_LOCAL, [29](#)
 PMIX_LOCAL_CPUSSETS, [228](#), [238](#)
 Defintion, [67](#)
 PMIX_LOCAL_PEERS, [228](#), [238](#)
 Defintion, [67](#)
 PMIX_LOCAL_PROCS, [230](#)
 Defintion, [67](#)
 PMIX_LOCAL_RANK, [174](#), [175](#), [229](#)
 Defintion, [66](#)
 PMIX_LOCAL_SIZE, [228](#)
 Defintion, [69](#)
 PMIX_LOCAL_TOPO
 Defintion, [70](#)
 PMIX_LOCALITY
 Defintion, [67](#)
 PMIX_LOCALITY_STRING
 Defintion, [70](#)
 PMIX_LOCALLDR, [230](#)
 Defintion, [67](#)
 PMIx_Log, [8](#), [198](#)
 Defintion, [196](#)
 PMIX_LOG_EMAIL, [198](#), [201](#), [288](#)
 Defintion, [79](#)
 PMIX_LOG_EMAIL_ADDR, [198](#), [201](#), [288](#)
 Defintion, [79](#)
 PMIX_LOG_EMAIL_MSG, [198](#), [201](#), [288](#)
 Defintion, [79](#)
 PMIX_LOG_EMAIL_SENDER_ADDR
 Defintion, [79](#)
 PMIX_LOG_EMAIL_SERVER
 Defintion, [79](#)
 PMIX_LOG_EMAIL_SRVR_PORT
 Defintion, [79](#)
 PMIX_LOG_EMAIL_SUBJECT, [198](#), [201](#),
 [288](#)
 Defintion, [79](#)
 PMIX_LOG_GENERATE_TIMESTAMP,
 [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_GLOBAL_DATASTORE, [198](#),
 [201](#)
 Defintion, [79](#)
 PMIX_LOG_GLOBAL_SYSLOG, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_JOB_RECORD, [198](#), [201](#)
 Defintion, [79](#)
 PMIX_LOG_LOCAL_SYSLOG, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_MSG, [288](#)
 Defintion, [79](#)
 PMIx_Log_nb, [6](#), [78](#), [201](#)
 Defintion, [198](#)
 PMIX_LOG_ONCE, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_SOURCE, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_STDERR, [197](#), [200](#), [287](#)
 Defintion, [78](#)
 PMIX_LOG_STDOUT, [197](#), [200](#), [287](#)
 Defintion, [78](#)
 PMIX_LOG_SYSLOG, [197](#), [200](#), [287](#)
 Defintion, [78](#)
 PMIX_LOG_SYSLOG_PRI, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_TAG_OUTPUT, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_TIMESTAMP, [197](#), [200](#)
 Defintion, [78](#)
 PMIX_LOG_TIMESTAMP_OUTPUT, [197](#),
 [200](#)
 Defintion, [78](#)
 PMIX_LOG_XML_OUTPUT, [197](#), [200](#)
 Defintion, [78](#)
 PMIx_Lookup, [6](#), [44](#), [135](#), [140](#), [142](#)
 Defintion, [138](#)
 pmix_lookup_cbfunc_t, [89](#), [266](#)
 Defintion, [89](#)
 PMIx_Lookup_nb, [89](#), [90](#)
 Defintion, [140](#)
 PMIX_MAP_BLOB

Defintion, [73](#)
 PMIX_MAPBY, [149](#), [154](#), [230](#), [272](#)
 Defintion, [75](#)
 PMIX_MAPPER, [74](#), [149](#), [153](#), [271](#)
 Defintion, [75](#)
 PMIX_MAX_KEYLEN, [17](#)
 PMIX_MAX_NSLEN, [17](#)
 PMIX_MAX_PROCS, [9](#), [51](#), [69](#), [70](#), [129](#),
 [228](#)
 Defintion, [70](#)
 PMIX_MAX_RESTARTS, [151](#), [155](#), [273](#)
 Defintion, [77](#)
 PMIX_MAX_VALUE, [10](#), [51](#)
 Defintion, [85](#)
 PMIX_MERGE_STDERR_STDOUT, [150](#),
 [154](#), [272](#)
 Defintion, [76](#)
 PMIX_MIN_VALUE, [10](#), [51](#)
 Defintion, [85](#)
 PMIX_MODEL_AFFINITY_POLICY
 Defintion, [64](#)
 PMIX_MODEL_CPU_TYPE
 Defintion, [64](#)
 PMIX_MODEL_DECLARED, [20](#)
 PMIX_MODEL_LIBRARY_NAME
 Defintion, [64](#)
 PMIX_MODEL_LIBRARY_VERSION
 Defintion, [64](#)
 PMIX_MODEL_NUM_CPUS
 Defintion, [64](#)
 PMIX_MODEL_NUM_THREADS
 Defintion, [64](#)
 PMIX_MODEL_PHASE_NAME
 Defintion, [64](#)
 PMIX_MODEL_PHASE_TYPE
 Defintion, [64](#)
 PMIX_MODEL_RESOURCES, [20](#)
 pmix_modex_cbfunc_t, [87](#), [259](#), [262](#)
 Defintion, [87](#)
 PMIX_MONITOR_APP_CONTROL, [192](#),
 [194](#), [295](#)
 Defintion, [83](#)
 PMIX_MONITOR_CANCEL, [192](#), [194](#), [295](#)
 Defintion, [83](#)
 PMIX_MONITOR_FILE, [192](#), [193](#), [195](#),
 [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_FILE_ACCESS, [193](#),
 [195](#), [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_FILE_ALERT, [20](#)
 PMIX_MONITOR_FILE_CHECK_TIME,
 [193](#), [195](#), [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_FILE_DROPS, [193](#),
 [195](#), [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_FILE_MODIFY, [193](#),
 [195](#), [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_FILE_SIZE, [193](#), [195](#),
 [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_HEARTBEAT, [192](#),
 [195](#), [295](#)
 Defintion, [83](#)
 PMIX_MONITOR_HEARTBEAT_ALERT,
 [20](#)
 PMIX_MONITOR_HEARTBEAT_DROPS,
 [192](#), [195](#), [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_HEARTBEAT_TIME,
 [192](#), [195](#), [296](#)
 Defintion, [83](#)
 PMIX_MONITOR_ID, [192](#), [194](#), [295](#)
 Defintion, [83](#)
 PMIX_NET_TOPO
 Defintion, [70](#)
 PMIX_NO_OVERSUBSCRIBE, [151](#), [155](#),
 [273](#)
 Defintion, [76](#)
 PMIX_NO_PROCS_ON_HEAD, [151](#), [155](#),
 [273](#)
 Defintion, [76](#)
 PMIX_NODE_INFO, [120](#), [123](#), [129](#), [174](#)
 Defintion, [68](#)

PMIX_NODE_INFO_ARRAY, 69, 235, 237
 Defintion, **69**
 PMIX_NODE_LIST
 Defintion, **67**
 PMIX_NODE_MAP, 9, 67, 228, 229,
 234–236, 247
 Defintion, **72**
 PMIX_NODE_RANK, 229
 Defintion, **66**
 PMIX_NODE_SIZE, 129, 230
 Defintion, **70**
 PMIX_NODEID, 68, 69, 120, 123, 129, 174,
 229
 Defintion, **67**
 PMIX_NON_PMI, 150, 154, 272
 Defintion, **75**
 pmix_notification_fn_t, 93, 205
 Defintion, **93**
 PMIX_NOTIFY_ALLOC_COMPLETE, 19
 PMIX_NOTIFY_COMPLETION
 Defintion, **72**
 PMIx_Notify_event, 7
 Defintion, **208**
 PMIX_NPROC_OFFSET, 230
 Defintion, **66**
 PMIX_NSDIR, 66
 Defintion, **66**
 PMIX_NSPACE, 68, 120, 123, 127,
 174–176, 235
 Defintion, **66**
 pmix_nspace_t, 22, 23, 25, 88
 Defintion, **22**
 PMIX_NUM_NODES, 117, 121, 124, 125,
 127, 234, 235
 Defintion, **70**
 PMIX_NUM_SLOTS
 Defintion, **70**
 pmix_op_cbfunc_t, 89, 92, 95, 137, 144,
 159, 163, 168, 199, 208, 209, 227,
 240–242, 249, 250, 252, 255, 256,
 258, 263, 268, 274, 276, 278, 280,
 281, 287, 292, 295, 301, 303, 318,
 331
 Defintion, **89**
 PMIX_OPENMP_PARALLEL_ENTERED,
 20
 PMIX_OPENMP_PARALLEL_EXITED,
 20
 PMIX_OPERATION_IN_PROGRESS, 20
 PMIX_OPERATION_SUCCEEDED, 20
 PMIX_OPTIONAL, 119, 122
 Defintion, **72**
 PMIX_OUTPUT_TO_FILE, 150, 155, 273
 Defintion, **76**
 PMIX_PARENT_ID, 148, 152, 153, 271
 Defintion, **67**
 PMIX_PDATA, 60
 PMIX_PDATA_CONSTRUCT
 Defintion, **45**
 PMIX_PDATA_CREATE
 Defintion, **45**
 PMIX_PDATA_DESTRUCT
 Defintion, **45**
 PMIX_PDATA_FREE
 Defintion, **45**
 PMIX_PDATA_LOAD
 Defintion, **46**
 pmix_pdata_t, 44–47, 89, 90, 140
 Defintion, **44**
 PMIX_PDATA_XFER
 Defintion, **47**
 PMIX_PERSIST, 61
 PMIX_PERSIST_APP, 30
 PMIX_PERSIST_FIRST_READ, 30
 PMIX_PERSIST_INDEF, 30
 PMIX_PERSIST_INVALID, 30
 PMIX_PERSIST_PROC, 30
 PMIX_PERSIST_SESSION, 30
 PMIX_PERSISTENCE, 136, 138, 264
 Defintion, **72**
 PMIx_Persistence_string, 7
 Defintion, **102**
 pmix_persistence_t, 30, 61, 102
 Defintion, **30**
 PMIX_PERSONALITY, 149, 153, 271
 Defintion, **74**

PMIX_PID, 60
 PMIX_POINTER, 61
 PMIX_PPR, 149, 154, 272
 Defintion, 75
 PMIX_PREFIX, 149, 153, 271
 Defintion, 75
 PMIX_PRELOAD_BIN, 149, 153, 271
 Defintion, 75
 PMIX_PRELOAD_FILES, 149, 153, 271
 Defintion, 75
 PMIX_PREPEND_ENVAR
 Defintion, 80
 PMIX_PROC, 60
 PMIX_PROC_BLOB
 Defintion, 73
 PMIX_PROC_CONSTRUCT, 24
 Defintion, 54
 PMIX_PROC_CREATE
 Defintion, 25
 PMIX_PROC_DATA, 236
 Defintion, 72
 PMIX_PROC_DESTRUCT
 Defintion, 24
 PMIX_PROC_FREE, 172
 Defintion, 25
 PMIX_PROC_HAS_CONNECTED, 20
 PMIX_PROC_INFO, 61
 PMIX_PROC_INFO_CONSTRUCT
 Defintion, 28
 PMIX_PROC_INFO_CREATE
 Defintion, 28
 PMIX_PROC_INFO_DESTRUCT
 Defintion, 28
 PMIX_PROC_INFO_FREE
 Defintion, 28
 pmix_proc_info_t, 27, 28, 61, 77, 176, 284
 Defintion, 27
 PMIX_PROC_LOAD
 Defintion, 25
 PMIX_PROC_MAP, 9, 228, 234, 235, 247
 Defintion, 72
 PMIX_PROC_PID
 Defintion, 67
 PMIX_PROC_RANK, 61
 PMIX_PROC_STATE, 61
 PMIX_PROC_STATE_ABORTED, 27
 PMIX_PROC_STATE_ABORTED_BY_SIG,
 27
 PMIX_PROC_STATE_CALLED_ABORT,
 27
 PMIX_PROC_STATE_CANNOT_RESTART,
 27
 PMIX_PROC_STATE_COMM_FAILED,
 27
 PMIX_PROC_STATE_CONNECTED, 26
 PMIX_PROC_STATE_ERROR, 26
 PMIX_PROC_STATE_FAILED_TO_LAUNCH,
 27
 PMIX_PROC_STATE_FAILED_TO_START,
 27
 PMIX_PROC_STATE_KILLED_BY_CMD,
 27
 PMIX_PROC_STATE_LAUNCH_UNDERWAY,
 26
 PMIX_PROC_STATE_MIGRATING, 27
 PMIX_PROC_STATE_PREPPED, 26
 PMIX_PROC_STATE_RESTART, 26
 PMIX_PROC_STATE_RUNNING, 26
 PMIX_PROC_STATE_STATUS
 Defintion, 72
 PMIx_Proc_state_string, 7
 Defintion, 101
 pmix_proc_state_t, 26, 61, 101
 Defintion, 26
 PMIX_PROC_STATE_TERM_NON_ZERO,
 27
 PMIX_PROC_STATE_TERM_WO_SYNC,
 27
 PMIX_PROC_STATE_TERMINATE, 26
 PMIX_PROC_STATE_TERMINATED, 26
 PMIX_PROC_STATE_UNDEF, 26
 PMIX_PROC_STATE_UNTERMINATED,
 26
 pmix_proc_t, 23–26, 46, 53, 60, 67, 73, 93,
 97, 100, 107, 109, 111, 113, 121,
 131–133, 146, 206, 209, 210, 214,

215, 231, 241–244, 250, 255, 256,
 258, 259, 262, 263, 266, 268, 270,
 274, 276, 281, 283, 287, 289, 292,
 294, 297, 299, 301, 303, 309, 313,
 320, 323, 327
 Definition, [24](#)
 PMIX_PROC_TERMINATED, [20](#)
 PMIX_PROC_URI, [177](#)
 Definition, [67](#)
 PMIX_PROCDIR
 Definition, [66](#)
 PMIx_Process_monitor, [8](#), [195](#)
 Definition, [191](#)
 PMIx_Process_monitor_nb, [6](#), [83](#), [171](#), [195](#)
 Definition, [193](#)
 PMIX_PROCID, [174–176](#), [231](#)
 Definition, [66](#)
 PMIX_PROGRAMMING_MODEL
 Definition, [64](#)
 PMIX_PSET_NAME, [305](#)
 Definition, [63](#)
 PMIx_Publish, [5](#), [29](#), [30](#), [72](#), [136–138](#), [264](#),
[265](#)
 Definition, [135](#)
 PMIx_Publish_nb, [5](#), [138](#)
 Definition, [137](#)
 PMIx_Put, [5](#), [29](#), [32](#), [96](#), [118](#), [121](#), [124](#), [130](#),
[132](#), [156](#), [178](#), [244](#), [263](#), [312](#), [322](#)
 Definition, [117](#)
 PMIX_QUERY, [61](#)
 PMIX_QUERY_ALLOC_STATUS, [176](#),
[285](#)
 Definition, [78](#)
 PMIX_QUERY_ATTRIBUTE_SUPPORT,
[175](#), [178](#)
 Definition, [78](#)
 PMIX_QUERY_AUTHORIZATIONS
 Definition, [77](#)
 PMIX_QUERY_CONSTRUCT
 Definition, [49](#)
 PMIX_QUERY_CREATE
 Definition, [50](#)
 PMIX_QUERY_DEBUG_SUPPORT, [176](#),
[284](#)
 Definition, [77](#)
 PMIX_QUERY_DESTRUCT
 Definition, [50](#)
 PMIX_QUERY_FREE
 Definition, [50](#)
 PMIx_Query_info_nb, [6](#), [7](#), [49](#), [69](#), [77](#), [129](#),
[156](#), [171](#), [177–179](#), [248](#), [305](#), [306](#)
 Definition, [173](#)
 PMIX_QUERY_JOB_STATUS, [176](#), [284](#)
 Definition, [77](#)
 PMIX_QUERY_LOCAL_ONLY, [284](#)
 Definition, [77](#)
 PMIX_QUERY_LOCAL_PROC_TABLE,
[176](#), [284](#)
 Definition, [77](#)
 PMIX_QUERY_MEMORY_USAGE, [176](#),
[284](#)
 Definition, [77](#)
 PMIX_QUERY_NAMESPACES, [176](#), [284](#)
 Definition, [77](#)
 PMIX_QUERY_NUM_PSETS
 Definition, [78](#)
 PMIX_QUERY_PARTIAL_SUCCESS, [19](#)
 PMIX_QUERY_PROC_TABLE, [176](#), [284](#)
 Definition, [77](#)
 PMIX_QUERY_PSET_NAMES
 Definition, [78](#)
 PMIX_QUERY_QUALIFIERS_CREATE,
[8](#), [9](#)
 Definition, [50](#)
 PMIX_QUERY_QUEUE_LIST, [176](#), [284](#)
 Definition, [77](#)
 PMIX_QUERY_QUEUE_STATUS, [176](#),
[284](#)
 Definition, [77](#)
 PMIX_QUERY_REFRESH_CACHE, [174](#),
[177](#), [178](#)
 Definition, [77](#)
 PMIX_QUERY_REPORT_AVG, [176](#), [284](#)
 Definition, [77](#)
 PMIX_QUERY_REPORT_MINMAX, [176](#),
[285](#)

Definition, [77](#)
 PMIX_QUERY_SPAWN_SUPPORT, [176](#),
 [284](#)
 Definition, [77](#)
 pmix_query_t, [8](#), [9](#), [49](#), [50](#), [61](#), [175](#), [176](#),
 [178](#), [283](#), [285](#)
 Definition, [49](#)
 PMIX_RANGE, [136](#), [138](#), [139](#), [141](#), [143](#),
 [145](#), [206](#), [264](#), [266](#), [269](#), [282](#)
 Definition, [72](#)
 PMIX_RANGE_CUSTOM, [29](#)
 PMIX_RANGE_GLOBAL, [29](#)
 PMIX_RANGE_INVALID, [29](#)
 PMIX_RANGE_LOCAL, [29](#)
 PMIX_RANGE_NAMESPACE, [29](#)
 PMIX_RANGE_PROC_LOCAL, [29](#)
 PMIX_RANGE_RM, [29](#)
 PMIX_RANGE_SESSION, [29](#)
 PMIX_RANGE_UNDEF, [29](#)
 PMIX_RANK, [174–176](#), [229](#)
 Definition, [66](#)
 PMIX_RANK_INVALID, [24](#)
 PMIX_RANK_LOCAL_NODE, [23](#)
 PMIX_RANK_LOCAL_PEERS, [24](#)
 pmix_rank_t, [23–25](#), [61](#)
 Definition, [23](#)
 PMIX_RANK_UNDEF, [23](#)
 PMIX_RANK_VALID, [24](#)
 PMIX_RANK_WILDCARD, [23](#)
 PMIX_RANKBY, [150](#), [154](#), [230](#), [272](#)
 Definition, [75](#)
 PMIX_RECONNECT_SERVER
 Definition, [63](#)
 PMIX_REGATTR, [61](#)
 PMIX_REGATTR_CONSTRUCT
 Definition, [52](#)
 PMIX_REGATTR_CREATE
 Definition, [52](#)
 PMIX_REGATTR_DESTRUCT
 Definition, [52](#)
 PMIX_REGATTR_FREE
 Definition, [53](#)
 PMIX_REGATTR_LOAD
 Definition, [53](#)
 pmix_regattr_t, [10](#), [51–53](#), [61](#), [85](#), [179](#), [247](#),
 [248](#)
 Definition, [51](#)
 PMIX_REGATTR_XFER
 Definition, [53](#)
 PMIx_Register_attributes, [10](#)
 Definition, [247](#)
 PMIX_REGISTER_CLEANUP, [187](#), [190](#)
 Definition, [82](#)
 PMIX_REGISTER_CLEANUP_DIR, [187](#),
 [190](#)
 Definition, [82](#)
 PMIx_Register_event_handler, [6](#), [92](#), [171](#)
 Definition, [204](#)
 PMIX_REGISTER_NODATA, [227](#)
 Definition, [72](#)
 pmix_release_cbfunc_t, [87](#)
 Definition, [87](#)
 PMIX_REMOTE, [29](#)
 PMIX_REPORT_BINDINGS, [151](#), [155](#), [273](#)
 Definition, [76](#)
 PMIX_REQUESTOR_IS_CLIENT, [148](#),
 [152](#)
 Definition, [63](#)
 PMIX_REQUESTOR_IS_TOOL, [148](#), [152](#)
 Definition, [63](#)
 PMIx_Resolve_nodes, [6](#)
 Definition, [172](#)
 PMIx_Resolve_peers, [6](#)
 Definition, [171](#)
 PMIX_RM_NAME
 Definition, [80](#)
 PMIX_RM_VERSION
 Definition, [80](#)
 PMIX_SCOPE, [61](#)
 PMIx_Scope_string, [7](#)
 Definition, [102](#)
 pmix_scope_t, [29](#), [61](#), [102](#), [118](#)
 Definition, [29](#)
 PMIX_SCOPE_UNDEF, [29](#)
 PMIX_SEND_HEARTBEAT
 Definition, [83](#)

pmix_server_abort_fn_t
 Defintion, [257](#)
 pmix_server_alloc_fn_t
 Defintion, [288](#)
 PMIX_SERVER_ATTRIBUTES, [10](#), [175](#)
 Defintion, [85](#)
 pmix_server_client_connected_fn_t, [89](#), [220](#),
 [242](#), [255](#)
 Defintion, [254](#)
 pmix_server_client_finalized_fn_t, [257](#)
 Defintion, [256](#)
 PMIx_server_collect_inventory, [8](#)
 Defintion, [251](#)
 pmix_server_connect_fn_t, [275](#), [277](#)
 Defintion, [274](#)
 PMIx_server_deliver_inventory, [8](#)
 Defintion, [252](#)
 PMIx_server_deregister_client, [6](#)
 Defintion, [242](#)
 pmix_server_deregister_events_fn_t
 Defintion, [279](#)
 PMIx_server_deregister_nspace, [6](#), [242](#)
 Defintion, [239](#)
 pmix_server_disconnect_fn_t, [277](#)
 Defintion, [276](#)
 pmix_server_dmodex_req_fn_t, [8](#), [9](#), [87](#)
 Defintion, [262](#)
 PMIx_server_dmodex_request, [6](#), [95](#), [96](#),
 [244](#)
 Defintion, [243](#)
 PMIX_SERVER_ENABLE_MONITORING
 Defintion, [62](#)
 pmix_server_fencenb_fn_t, [9](#), [87](#), [261](#)
 Defintion, [258](#)
 PMIx_server_finalize, [6](#)
 Defintion, [116](#)
 PMIX_SERVER_FUNCTIONS
 Defintion, [85](#)
 PMIX_SERVER_GATEWAY
 Defintion, [62](#)
 pmix_server_get_cred_fn_t, [300](#)
 Defintion, [296](#)
 PMIX_SERVER_HOSTNAME
 Defintion, [63](#)
 PMIx_server_init, [6](#), [105](#), [248](#), [253](#)
 Defintion, [114](#)
 PMIx_server_IOF_deliver, [8](#), [164](#)
 Defintion, [250](#)
 pmix_server_iof_fn_t
 Defintion, [300](#)
 pmix_server_job_control_fn_t
 Defintion, [291](#)
 pmix_server_listener_fn_t
 Defintion, [282](#)
 pmix_server_log_fn_t
 Defintion, [286](#)
 pmix_server_lookup_fn_t
 Defintion, [265](#)
 pmix_server_module_t, [114](#), [116](#), [248](#), [253](#)
 Defintion, [253](#)
 pmix_server_monitor_fn_t
 Defintion, [294](#)
 pmix_server_notify_event_fn_t, [94](#)
 Defintion, [281](#)
 PMIX_SERVER_NAMESPACE, [113](#), [114](#), [229](#)
 Defintion, [62](#)
 PMIX_SERVER_PIDINFO, [109](#), [111](#), [113](#)
 Defintion, [62](#)
 pmix_server_publish_fn_t
 Defintion, [263](#)
 pmix_server_query_fn_t
 Defintion, [283](#)
 PMIX_SERVER_RANK, [114](#), [230](#)
 Defintion, [62](#)
 PMIx_server_register_client, [6](#), [220](#), [242](#),
 [255](#), [257](#)
 Defintion, [241](#)
 pmix_server_register_events_fn_t
 Defintion, [277](#)
 PMIx_server_register_nspace, [6](#), [7](#), [14](#), [68](#),
 [89](#), [232](#), [235](#)
 Defintion, [227](#)
 PMIX_SERVER_REMOTE_CONNECTIONS,
 [115](#)
 Defintion, [62](#)
 PMIx_server_setup_application, [6](#), [9](#), [94](#), [95](#),

249, 253
 Definition, [245](#)
 PMIx_server_setup_fork, [6](#)
 Definition, [243](#)
 PMIx_server_setup_local_support, [6](#)
 Definition, [248](#)
 pmix_server_spawn_fn_t, [88](#)
 Definition, [269](#)
 pmix_server_stdin_fn_t
 Definition, [303](#)
 PMIX_SERVER_SYSTEM_SUPPORT, [114](#)
 Definition, [62](#)
 PMIX_SERVER_TMPDIR, [114](#)
 Definition, [62](#)
 pmix_server_tool_connection_fn_t, [220](#)
 Definition, [285](#)
 PMIX_SERVER_TOOL_SUPPORT, [114](#),
[220](#)
 Definition, [62](#)
 pmix_server_unpublish_fn_t
 Definition, [267](#)
 PMIX_SERVER_URI, [109](#), [111](#), [113](#), [177](#)
 Definition, [63](#)
 pmix_server_validate_cred_fn_t
 Definition, [298](#)
 PMIX_SESSION_ID, [68](#), [119](#), [123](#), [126](#),
[174](#), [229](#), [234](#), [235](#)
 Definition, [67](#)
 PMIX_SESSION_INFO, [119](#), [123](#), [125](#), [174](#)
 Definition, [67](#)
 PMIX_SESSION_INFO_ARRAY, [8](#), [69](#),
[228](#), [234](#)
 Definition, [68](#)
 PMIX_SET_ENVAR
 Definition, [80](#)
 PMIX_SET_SESSION_CWD, [149](#), [153](#), [271](#)
 Definition, [76](#)
 PMIX_SETUP_APP_ALL, [245](#)
 Definition, [85](#)
 PMIX_SETUP_APP_ENVARS, [245](#)
 Definition, [85](#)
 PMIX_SETUP_APP_NONENVARS, [245](#)
 Definition, [85](#)
 pmix_setup_application_cbfunc_t, [245](#)
 Definition, [94](#)
 PMIX_SINGLE_LISTENER, [106](#)
 Definition, [64](#)
 PMIX_SIZE, [60](#)
 PMIX_SOCKET_MODE, [106](#), [110](#), [115](#)
 Definition, [64](#)
 PMIx_Spawn, [6](#), [47](#), [66](#), [74](#), [79](#), [80](#), [147](#), [148](#),
[152](#), [153](#), [155](#), [231](#), [243](#), [269](#), [274](#)
 Definition, [147](#)
 pmix_spawn_cbfunc_t, [88](#), [152](#), [270](#)
 Definition, [88](#)
 PMIx_Spawn_nb, [6](#), [47](#), [88](#)
 Definition, [152](#)
 PMIX_SPAWN_TOOL
 Definition, [77](#)
 PMIX_SPAWNED, [148](#), [152](#), [153](#), [271](#)
 Definition, [65](#)
 PMIX_STATUS, [60](#)
 pmix_status_t, [18](#), [21](#), [35](#), [36](#), [60](#), [91–93](#),
[95–99](#), [101](#), [204](#), [209](#), [278](#), [280](#),
[281](#)
 Definition, [18](#)
 PMIX_STDIN_TGT, [150](#), [154](#), [272](#)
 Definition, [75](#)
 PMIx_Store_internal, [6](#)
 Definition, [124](#)
 PMIX_STRING, [60](#)
 PMIX_SUCCESS, [18](#)
 pmix_system_event
 Definition, [21](#)
 PMIX_SYSTEM_TMPDIR, [114](#)
 Definition, [62](#)
 PMIX_TAG_OUTPUT, [150](#), [154](#), [272](#)
 Definition, [76](#)
 PMIX_TCP_DISABLE_IPV4, [107](#), [110](#), [115](#)
 Definition, [65](#)
 PMIX_TCP_DISABLE_IPV6, [107](#), [110](#), [115](#)
 Definition, [65](#)
 PMIX_TCP_IF_EXCLUDE, [106](#), [110](#), [115](#)
 Definition, [65](#)
 PMIX_TCP_IF_INCLUDE, [106](#), [110](#), [115](#)
 Definition, [65](#)

PMIX_TCP_IPV4_PORT, 107, 110, 115
 Defintion, **65**

PMIX_TCP_IPV6_PORT, 107, 110, 115
 Defintion, **65**

PMIX_TCP_REPORT_URI, 106, 110, 115
 Defintion, **65**

PMIX_TCP_URI, 109, 111
 Defintion, **65**

PMIX_TDIR_RMCLEAN
 Defintion, **66**

PMIX_THREADING_MODEL
 Defintion, **64**

PMIX_TIME, 60

PMIX_TIME_REMAINING, 171, 176, 285
 Defintion, **78**

PMIX_TIMEOUT, 4, 11, 120, 121, 123, 124,
 131, 132, 134, 136, 138–143, 145,
 157, 158, 160, 161, 163, 164, 222,
 224, 260, 263, 264, 267, 269, 273,
 275, 277, 298, 300, 308, 310, 312,
 315–317, 319, 321, 325, 327, 329,
 330
 Defintion, **71**

PMIX_TIMESTAMP_OUTPUT, 150, 154,
 272
 Defintion, **76**

PMIX_TIMEVAL, 60

PMIX_TMPDIR, 66
 Defintion, **66**

PMIX_TOOL_ATTRIBUTES, 10, 175
 Defintion, **85**

PMIx_tool_connect_to_server, 8
 Defintion, **112**

pmix_tool_connection_cbfunc_t, 285
 Defintion, **96**

PMIX_TOOL_DO_NOT_CONNECT, 109,
 111
 Defintion, **63**

PMIx_tool_finalize, 6
 Defintion, **112**

PMIX_TOOL_FUNCTIONS
 Defintion, **85**

PMIx_tool_init, 6, 62, 105, 112
 Defintion, **108**

PMIX_TOOL_NAMESPACE, 109
 Defintion, **62**

PMIX_TOOL_RANK, 109
 Defintion, **62**

PMIX_TOPOLOGY
 Defintion, **70**

PMIX_TOPOLOGY_FILE
 Defintion, **70**

PMIX_TOPOLOGY_SIGNATURE
 Defintion, **70**

PMIX_TOPOLOGY_XML
 Defintion, **70**

PMIX_UINT, 60

PMIX_UINT16, 60

PMIX_UINT32, 60

PMIX_UINT64, 60

PMIX_UINT8, 60

PMIX_UNDEF, 60

PMIX_UNIV_SIZE, 7–9, 121, 124, 125,
 228, 234
 Defintion, **69**

PMIx_Unpublish, 6, 143, 145
 Defintion, **142**

PMIx_Unpublish_nb, 6
 Defintion, **143**

PMIX_UNSET_ENVAR
 Defintion, **80**

PMIX_USERID, 99, 135, 137, 139, 141,
 142, 144, 175, 180, 183, 186, 189,
 192, 194, 196, 199, 222, 224,
 264–270, 278, 284, 286, 287, 289,
 292, 295, 297, 299, 301, 304
 Defintion, **63**

PMIX_USOCK_DISABLE, 106, 115
 Defintion, **64**

PMIx_Validate_credential, 8
 Defintion, **223**

pmix_validation_cbfunc_t, 223, 299
 Defintion, **98**

PMIX_VALUE, 60

pmix_value_cbfunc_t, 90
 Defintion, **90**

PMIX_VALUE_CONSTRUCT
 Defintion, [33](#)
 PMIX_VALUE_CREATE
 Defintion, [33](#)
 PMIX_VALUE_DESTRUCT
 Defintion, [33](#)
 PMIX_VALUE_FREE
 Defintion, [34](#)
 PMIX_VALUE_GET_NUMBER
 Defintion, [36](#)
 PMIX_VALUE_LOAD
 Defintion, [34](#)
 pmix_value_t, [32–36](#), [60](#), [90](#), [117](#), [118](#)
 Defintion, [32](#)
 PMIX_VALUE_UNLOAD
 Defintion, [35](#)
 PMIX_VALUE_XFER
 Defintion, [35](#)
 PMIX_VERSION_INFO
 Defintion, [63](#)
 PMIX_WAIT, [139–141](#), [266](#)
 Defintion, [71](#)
 PMIX_WDIR, [148](#), [153](#), [271](#)
 Defintion, [75](#)
 rank, [127](#), [236](#)
 Defintion, [12](#)
 resource manager
 Defintion, [13](#)
 session, [7](#), [9](#), [67](#), [68](#), [117](#), [125](#), [178](#), [232](#)
 Defintion, [12](#)
 slot
 Defintion, [12](#)
 slots
 Defintion, [12](#)
 workflow
 Defintion, [12](#)