

# Process Management Interface for Exascale (PMIx) Standard

Version 4.0 (Draft)

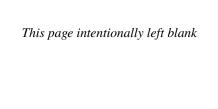
Created on June 23, 2020

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

Comments: Please provide comments on the PMIx Standard by filing issues on the document repository <a href="https://github.com/pmix/pmix-standard/issues">https://github.com/pmix/pmix-standard/issues</a> or by sending them to the PMIx Community mailing list at <a href="https://groups.google.com/forum/#!forum/pmix">https://groups.google.com/forum/#!forum/pmix</a>. 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.



# **Contents**

| 1. | Intro        | duction   | 1  |
|----|--------------|---|----|
|    | 1.1.         | Charter   | 2  |
|    | 1.2.         |   | 2  |
|    | 1.2.         | 1.2.1. Who should use the standard?                     | 2  |
|    |              | 1.2.2. What is defined in the standard?                 | 3  |
|    |              | 1.2.3. What is <i>not</i> defined in the standard?      | 3  |
|    |              | 1.2.4. General Guidance for PMIx Users and Implementors | 4  |
|    | 1.3.         | PMIx Architecture Overview                              | 4  |
|    | 1.3.         | 1.3.1. The PMIx Reference Implementation (PRI)          | 6  |
|    |              | 1.3.2. The PMIx Reference RunTime Environment (PRRTE)   | 7  |
|    | 1.4.         | Organization of this document                           | 7  |
|    | 1.5.         | Version 1.0: June 12, 2015                              | 8  |
|    | 1.6.         | Version 2.0: Sept. 2018                                 | 9  |
|    |              | Version 2.1: Dec. 2018                                  | 9  |
|    | 1.7.<br>1.8. |   |    |
|    |              |   | 10 |
|    | 1.9.         |   | 11 |
|    |              |   | 11 |
|    |              |   | 12 |
|    | 1.12.        | Version 4.0: June 2019                                  | 12 |
| 2. | PMIx         | Terms and Conventions                                   | 14 |
|    | 2.1.         | Notational Conventions                                  | 16 |
|    | 2.2.         | Semantics   | 17 |
|    | 2.3.         |   | 18 |
|    | 2.4.         | Procedure Conventions                                   | 18 |
|    | 2.5.         | Standard vs Reference Implementation                    | 19 |
|    |              |   |    |
| 3. | Data         | Structures and Types 2                                  | 20 |
|    | 3.1.         | Constants   | 21 |
|    |              | 3.1.1. PMIx Error Constants                             | 22 |

|      | 3.1.2.  | Macros for use with PMIx constants           | 27 |
|------|---------|--|----|
| 3.2. | Data Ty | pes  | 27 |
|      | 3.2.1.  | Key Structure                                | 27 |
|      | 3.2.2.  | Namespace Structure                          | 28 |
|      | 3.2.3.  | Rank Structure                               | 29 |
|      | 3.2.4.  | Process Structure                            | 30 |
|      | 3.2.5.  | Process structure support macros             | 30 |
|      | 3.2.6.  | Process State Structure                      | 32 |
|      | 3.2.7.  | Process Information Structure                | 33 |
|      | 3.2.8.  | Process Information Structure support macros | 33 |
|      | 3.2.9.  | Scope of Put Data                            | 35 |
|      | 3.2.10. | Job State Structure                          | 35 |
|      | 3.2.11. | Range of Published Data                      | 36 |
|      | 3.2.12. | Data Persistence Structure                   | 36 |
|      | 3.2.13. | Data Array Structure                         | 37 |
|      | 3.2.14. | Data array structure support macros          | 37 |
|      | 3.2.15. | Value Structure                              | 38 |
|      | 3.2.16. | Value structure support macros               | 39 |
|      | 3.2.17. | Info Structure                               | 43 |
|      | 3.2.18. | Info structure support macros                | 43 |
|      | 3.2.19. | Info Type Directives                         | 46 |
|      | 3.2.20. | Info Directive support macros                | 47 |
|      | 3.2.21. | Job Allocation Directives                    | 49 |
|      | 3.2.22. | IO Forwarding Channels                       | 49 |
|      | 3.2.23. | Environmental Variable Structure             | 49 |
|      | 3.2.24. | Environmental variable support macros        | 50 |
|      | 3.2.25. | Lookup Returned Data Structure               | 51 |
|      | 3.2.26. | Lookup data structure support macros         | 51 |
|      | 3.2.27. | Application Structure                        | 54 |
|      | 3.2.28. | App structure support macros                 | 55 |
|      | 3.2.29. | Query Structure                              | 56 |
|      | 3.2.30. | Query structure support macros               | 56 |
|      | 3 2 31  | Attribute registration structure             | 58 |

|      | 3.2.32. | Attribute registration structure support macros        | 59        |
|------|---------|--|-----------|
|      | 3.2.33. | PMIx Group Directives                                  | 61        |
|      | 3.2.34. | Byte Object Type                                       | 61        |
|      | 3.2.35. | Byte object support macros                             | 61        |
|      | 3.2.36. | Data Array Structure                                   | 63        |
|      | 3.2.37. | Data array support macros                              | 63        |
|      | 3.2.38. | Argument Array Macros                                  | 64        |
|      | 3.2.39. | Set Environment Variable                               | 68        |
| 3.3. | General | ized Data Types Used for Packing/Unpacking             | 69        |
| 3.4. | Reserve | d attributes   | <b>70</b> |
|      | 3.4.1.  | Initialization attributes                              | 71        |
|      | 3.4.2.  | Tool-related attributes                                | 71        |
|      | 3.4.3.  | Identification attributes                              | 72        |
|      | 3.4.4.  | Programming model attributes                           | 73        |
|      | 3.4.5.  | UNIX socket rendezvous socket attributes               | 73        |
|      | 3.4.6.  | TCP connection attributes                              | 74        |
|      | 3.4.7.  | Global Data Storage (GDS) attributes                   | 74        |
|      | 3.4.8.  | General process-level attributes                       | 74        |
|      | 3.4.9.  | Scratch directory attributes                           | 75        |
|      | 3.4.10. | Relative Rank Descriptive Attributes                   | 75        |
|      | 3.4.11. | Information retrieval attributes                       | 77        |
|      | 3.4.12. | Information storage attributes                         | 77        |
|      | 3.4.13. | Size information attributes                            | <b>78</b> |
|      | 3.4.14. | Memory information attributes                          | <b>79</b> |
|      | 3.4.15. | Topology information attributes                        | <b>79</b> |
|      | 3.4.16. | Request-related attributes                             | 80        |
|      | 3.4.17. | Server-to-PMIx library attributes                      | 82        |
|      | 3.4.18. | Server-to-Client attributes                            | 82        |
|      | 3.4.19. | Event handler registration and notification attributes | 82        |
|      | 3.4.20. | Fault tolerance attributes                             | 83        |
|      | 3.4.21. | Spawn attributes                                       | 84        |
|      | 3.4.22. | Query attributes                                       | 86        |
|      | 3.4.23. | Log attributes   | 87        |

|      | 3.4.24.  | Debugger attributes                               |
|------|----------|---|
|      | 3.4.25.  | Resource manager attributes                       |
|      | 3.4.26.  | Environment variable attributes                   |
|      | 3.4.27.  | Job Allocation attributes                         |
|      | 3.4.28.  | Job control attributes                            |
|      | 3.4.29.  | Monitoring attributes                             |
|      | 3.4.30.  | Security attributes                               |
|      | 3.4.31.  | IO Forwarding attributes                          |
|      | 3.4.32.  | Application setup attributes                      |
|      | 3.4.33.  | Attribute support level attributes                |
|      | 3.4.34.  | Descriptive attributes                            |
|      | 3.4.35.  | Process group attributes                          |
| 3.5. | Callbacl | c Functions                                       |
|      | 3.5.1.   | Release Callback Function                         |
|      | 3.5.2.   | Modex Callback Function                           |
|      | 3.5.3.   | Spawn Callback Function                           |
|      | 3.5.4.   | Op Callback Function                              |
|      | 3.5.5.   | Lookup Callback Function                          |
|      | 3.5.6.   | Value Callback Function                           |
|      | 3.5.7.   | Info Callback Function                            |
|      | 3.5.8.   | Event Handler Registration Callback Function      |
|      | 3.5.9.   | Notification Handler Completion Callback Function |
|      | 3.5.10.  | Notification Function                             |
|      | 3.5.11.  | Server Setup Application Callback Function        |
|      | 3.5.12.  | Server Direct Modex Response Callback Function    |
|      | 3.5.13.  | PMIx Client Connection Callback Function          |
|      | 3.5.14.  | PMIx Tool Connection Callback Function            |
|      | 3.5.15.  | Credential callback function                      |
|      | 3.5.16.  | Credential validation callback function           |
|      | 3.5.17.  | IOF delivery function                             |
|      | 3.5.18.  | IOF and Event registration function               |
| 3.6  | Constan  | t String Functions 111                            |

| 4. | Initia | alization | and Finalization                | 115 |
|----|--------|-----------|---------------------------------|-----|
|    | 4.1.   | Query .   |                                 | 115 |
|    |        | 4.1.1.    | PMIx_Initialized                | 115 |
|    |        | 4.1.2.    | PMIx_Get_version                | 116 |
|    | 4.2.   | Client In | nitialization and Finalization  | 116 |
|    |        | 4.2.1.    | PMIx_Init                       | 116 |
|    |        | 4.2.2.    | PMIx_Finalize                   | 119 |
|    | 4.3.   | Tool Ini  | tialization and Finalization    | 119 |
|    |        | 4.3.1.    | PMIx_tool_init                  | 119 |
|    |        | 4.3.2.    | PMIx_tool_finalize              | 123 |
|    |        | 4.3.3.    | PMIx_tool_connect_to_server     | 123 |
|    | 4.4.   | Server I  | nitialization and Finalization  | 125 |
|    |        | 4.4.1.    | PMIx_server_init                | 125 |
|    |        | 4.4.2.    | PMIx_server_finalize            | 127 |
| 5  | Key/   | Value M   | anagement                       | 128 |
| Ο. | 5.1.   |           | and Accessing Key/Value Pairs   | 128 |
|    | 3.1.   | 5.1.1.    | PMIx_Put                        | 128 |
|    |        | 5.1.2.    | PMIx_Get                        | 129 |
|    |        | 5.1.3.    | PMIx Get nb                     | 132 |
|    |        | 5.1.4.    | PMIx_Store_internal             | 135 |
|    |        | 5.1.5.    | Accessing information: examples | 136 |
|    | 5.2.   | Exchang   | ging Key/Value Pairs            | 140 |
|    |        | 5.2.1.    | PMIx_Commit                     | 141 |
|    |        | 5.2.2.    | PMIx_Fence                      | 141 |
|    |        | 5.2.3.    | PMIx_Fence_nb                   | 143 |
|    | 5.3.   | Publish   | and Lookup Data                 | 146 |
|    |        | 5.3.1.    | PMIx_Publish                    | 146 |
|    |        | 5.3.2.    | PMIx_Publish_nb                 | 148 |
|    |        | 5.3.3.    | PMIx_Lookup                     | 149 |
|    |        | 5.3.4.    | PMIx_Lookup_nb                  | 152 |
|    |        | 5.3.5.    | PMIx_Unpublish                  | 153 |
|    |        | 536       | PMTy Unnublish nh               | 155 |

| 6. | Proc | ess Ma  | nagement                          | 157 |
|----|------|---------|-----------------------------------|-----|
|    | 6.1. | Abort   |                                   | 157 |
|    |      | 6.1.1.  | PMIx_Abort                        | 157 |
|    | 6.2. | Process | Creation                          | 158 |
|    |      | 6.2.1.  | PMIx_Spawn                        | 158 |
|    |      | 6.2.2.  | PMIx_Spawn_nb                     | 163 |
|    | 6.3. | Connec  | eting and Disconnecting Processes | 167 |
|    |      | 6.3.1.  | PMIx_Connect                      | 168 |
|    |      | 6.3.2.  | PMIx_Connect_nb                   | 170 |
|    |      | 6.3.3.  | PMIx_Disconnect                   | 172 |
|    |      | 6.3.4.  | PMIx_Disconnect_nb                | 174 |
|    | 6.4. | IO Forv | warding                           | 176 |
|    |      | 6.4.1.  | PMIx_IOF_pull                     | 177 |
|    |      | 6.4.2.  | PMIx_IOF_deregister               | 179 |
|    |      | 6.4.3.  | PMIx_IOF_push                     | 180 |
| 7. | Job  | Manage  | ement and Reporting               | 183 |
|    | 7.1. | Query   |                                   | 183 |
|    |      | 7.1.1.  | PMIx_Resolve_peers                | 183 |
|    |      | 7.1.2.  | PMIx_Resolve_nodes                | 184 |
|    |      | 7.1.3.  | PMIx_Query_info                   | 185 |
|    |      | 7.1.4.  | PMIx_Query_info_nb                | 189 |
|    | 7.2. | Allocat | ion Requests                      | 195 |
|    |      | 7.2.1.  | PMIx_Allocation_request           | 196 |
|    |      | 7.2.2.  | PMIx_Allocation_request_nb        | 199 |
|    | 7.3. | Job Co  | ntrol                             | 202 |
|    |      | 7.3.1.  | PMIx_Job_control                  | 202 |
|    |      | 7.3.2.  | PMIx_Job_control_nb               | 205 |
|    | 7.4. | Process | and Job Monitoring                | 208 |
|    |      | 7.4.1.  | PMIx_Process_monitor              | 208 |
|    |      | 7.4.2.  | PMIx_Process_monitor_nb           | 210 |
|    |      | 7.4.3.  | PMIx_Heartbeat                    | 212 |
|    | 7.5. | Loggin  | g                                 | 213 |
|    |      | 7.5.1.  | PMIx Log                          | 213 |

|    |       | 7.5.2.               | PMIx_Log_nb                   | 215 |  |  |  |  |  |
|----|-------|----------------------|-------------------------------|-----|--|--|--|--|--|
| 8. | Even  | Event Notification 2 |                               |     |  |  |  |  |  |
|    | 8.1.  | Notifica             | tion and Management           | 219 |  |  |  |  |  |
|    |       | 8.1.1.               | PMIx_Register_event_handler   | 221 |  |  |  |  |  |
|    |       | 8.1.2.               | PMIx_Deregister_event_handler | 224 |  |  |  |  |  |
|    |       | 8.1.3.               | PMIx_Notify_event             | 225 |  |  |  |  |  |
| 9. | Data  | Packing              | g and Unpacking               | 229 |  |  |  |  |  |
|    | 9.1.  | Data Bu              | ffer Type                     | 229 |  |  |  |  |  |
|    | 9.2.  | Support              | Macros                        | 230 |  |  |  |  |  |
|    |       | 9.2.1.               | PMIX_DATA_BUFFER_CREATE       | 230 |  |  |  |  |  |
|    |       | 9.2.2.               | PMIX_DATA_BUFFER_RELEASE      | 230 |  |  |  |  |  |
|    |       | 9.2.3.               | PMIX_DATA_BUFFER_CONSTRUCT    | 230 |  |  |  |  |  |
|    |       | 9.2.4.               | PMIX_DATA_BUFFER_DESTRUCT     | 231 |  |  |  |  |  |
|    |       | 9.2.5.               | PMIX_DATA_BUFFER_LOAD         | 231 |  |  |  |  |  |
|    |       | 9.2.6.               | PMIX_DATA_BUFFER_UNLOAD       | 232 |  |  |  |  |  |
|    | 9.3.  | General              | Routines                      | 232 |  |  |  |  |  |
|    |       | 9.3.1.               | PMIx_Data_pack                | 232 |  |  |  |  |  |
|    |       | 9.3.2.               | PMIx_Data_unpack              | 234 |  |  |  |  |  |
|    |       | 9.3.3.               | PMIx_Data_copy                | 236 |  |  |  |  |  |
|    |       | 9.3.4.               | PMIx_Data_print               | 236 |  |  |  |  |  |
|    |       | 9.3.5.               | PMIx_Data_copy_payload        | 237 |  |  |  |  |  |
| 10 | .Secu | ırity                |                               | 239 |  |  |  |  |  |
|    | 10.1. | Obtainir             | ng Credentials                | 240 |  |  |  |  |  |
|    |       | 10.1.1.              | PMIx_Get_credential           | 240 |  |  |  |  |  |
|    |       | 10.1.2.              | PMIx_Get_credential_nb        | 241 |  |  |  |  |  |
|    | 10.2. | Validati             | ng Credentials                | 243 |  |  |  |  |  |
|    |       | 10.2.1.              | PMIx_Validate_credential      | 243 |  |  |  |  |  |
|    |       | 10.2.2.              | PMIx_Validate_credential_nb   | 245 |  |  |  |  |  |
| 11 | .Serv | er-Spec              | ific Interfaces               | 248 |  |  |  |  |  |
|    | 11.1. | Server S             | Support Functions             | 248 |  |  |  |  |  |
|    |       | 11.1.1.              | PMIx generate regex           | 248 |  |  |  |  |  |

|       | 11.1.2.  | PMIx_generate_ppn                             | 249 |
|-------|----------|---|-----|
|       | 11.1.3.  | PMIx_server_register_nspace                   | 250 |
|       | 11.1.4.  | PMIx_server_deregister_nspace                 | 263 |
|       | 11.1.5.  | PMIx_server_register_client                   | 265 |
|       | 11.1.6.  | PMIx_server_deregister_client                 | 266 |
|       | 11.1.7.  | PMIx_server_setup_fork                        | 267 |
|       | 11.1.8.  | PMIx_server_dmodex_request                    | 267 |
|       | 11.1.9.  | PMIx_server_setup_application                 | 269 |
|       | 11.1.10. | PMIx_Register_attributes                      | 271 |
|       | 11.1.11. | PMIx_server_setup_local_support               | 273 |
|       | 11.1.12. | PMIx_server_IOF_deliver                       | 274 |
|       | 11.1.13. | <pre>PMIx_server_collect_inventory</pre>      | 275 |
|       | 11.1.14. | PMIx_server_deliver_inventory                 | 276 |
| 11.2. | Server F | function Pointers                             | 278 |
|       | 11.2.1.  | <pre>pmix_server_module_t Module</pre>        | 278 |
|       | 11.2.2.  | <pre>pmix_server_client_connected_fn_t</pre>  | 279 |
|       | 11.2.3.  | <pre>pmix_server_client_finalized_fn_t</pre>  | 281 |
|       | 11.2.4.  | <pre>pmix_server_abort_fn_t</pre>             | 282 |
|       | 11.2.5.  | <pre>pmix_server_fencenb_fn_t</pre>           | 283 |
|       | 11.2.6.  | <pre>pmix_server_dmodex_req_fn_t</pre>        | 287 |
|       | 11.2.7.  | <pre>pmix_server_publish_fn_t</pre>           | 288 |
|       | 11.2.8.  | <pre>pmix_server_lookup_fn_t</pre>            | 290 |
|       | 11.2.9.  | <pre>pmix_server_unpublish_fn_t</pre>         | 292 |
|       | 11.2.10. | <pre>pmix_server_spawn_fn_t</pre>             | 294 |
|       | 11.2.11. | <pre>pmix_server_connect_fn_t</pre>           | 299 |
|       | 11.2.12. | <pre>pmix_server_disconnect_fn_t</pre>        | 301 |
|       | 11.2.13. | <pre>pmix_server_register_events_fn_t</pre>   | 303 |
|       | 11.2.14. | <pre>pmix_server_deregister_events_fn_t</pre> | 305 |
|       | 11.2.15. | <pre>pmix_server_notify_event_fn_t</pre>      | 307 |
|       | 11.2.16. | <pre>pmix_server_listener_fn_t</pre>          | 308 |
|       | 11.2.17. | <pre>pmix_server_query_fn_t</pre>             | 309 |
|       | 11.2.18. | <pre>pmix_server_tool_connection_fn_t</pre>   | 312 |
|       | 11.2.19. | pmix server log fn t                          | 313 |

|    |        | 11.2.20. | <pre>pmix_server_alloc_fn_t</pre>         | 315 |
|----|--------|----------|---|-----|
|    |        | 11.2.21. | <pre>pmix_server_job_control_fn_t</pre>   | 317 |
|    |        | 11.2.22. | <pre>pmix_server_monitor_fn_t</pre>       | 320 |
|    |        | 11.2.23. | <pre>pmix_server_get_cred_fn_t</pre>      | 323 |
|    |        | 11.2.24. | <pre>pmix_server_validate_cred_fn_t</pre> | 324 |
|    |        | 11.2.25. | <pre>pmix_server_iof_fn_t</pre>           | 326 |
|    |        | 11.2.26. | <pre>pmix_server_stdin_fn_t</pre>         | 329 |
|    |        | 11.2.27. | <pre>pmix_server_grp_fn_t</pre>           | 330 |
|    |        | 11.2.28. | <pre>pmix_server_fabric_fn_t</pre>        | 332 |
| 12 | .Fabri | ic Supp  | ort Definitions                           | 335 |
|    |        |          | upport Constants                          | 335 |
|    |        |          | upport Datatypes                          | 335 |
|    |        | 12.2.1.  | Fabric Coordinate Structure               | 336 |
|    |        | 12.2.2.  | Fabric Coordinate Support Macros          | 337 |
|    |        | 12.2.3.  | Fabric Coordinate Views                   | 338 |
|    |        | 12.2.4.  | Fabric Link State                         | 338 |
|    |        | 12.2.5.  | Fabric Operation Constants                | 339 |
|    |        | 12.2.6.  | Fabric registration structure             | 339 |
|    | 12.3.  | Fabric S | upport Attributes                         | 342 |
|    | 12.4.  | Fabric S | upport Functions                          | 345 |
|    |        | 12.4.1.  | PMIx_Fabric_register                      | 345 |
|    |        | 12.4.2.  | PMIx_Fabric_update                        | 347 |
|    |        | 12.4.3.  | PMIx_Fabric_deregister                    | 347 |
|    |        | 12.4.4.  | PMIx_Fabric_get_vertex_info               | 347 |
|    |        | 12.4.5.  | PMIx_Fabric_get_device_index              | 350 |
| 13 | .Proc  | ess Sets | s and Groups                              | 351 |
|    | 13.1.  | Process  | Sets                                      | 351 |
|    | 13.2.  | Process  | Groups                                    | 352 |
|    |        | 13.2.1.  | Group Operation Constants                 | 354 |
|    |        | 13.2.2.  | PMIx_Group_construct                      | 355 |
|    |        | 13.2.3.  | PMIx_Group_construct_nb                   | 358 |
|    |        | 13 2 4   | PMTx Group destruct                       | 361 |

|    |      | 13.2.5.  | PMIx_Group_destruct_nb            | 363 |
|----|------|----------|-----------------------------------|-----|
|    |      | 13.2.6.  | PMIx_Group_invite                 | 365 |
|    |      | 13.2.7.  | PMIx_Group_invite_nb              | 369 |
|    |      | 13.2.8.  | PMIx_Group_join                   | 371 |
|    |      | 13.2.9.  | PMIx_Group_join_nb                | 374 |
|    |      | 13.2.10. | PMIx_Group_leave                  | 376 |
|    |      | 13.2.11. | PMIx_Group_leave_nb               | 377 |
| Α. | Pyth | on Bind  | ings                              | 379 |
|    | A.1. |          | Considerations                    | 379 |
|    |      | A.1.1.   | Error Codes vs Python Exceptions  | 379 |
|    |      | A.1.2.   | Representation of Structured Data | 379 |
|    | A.2. | Datatype | e Definitions                     | 380 |
|    |      | A.2.1.   | Example                           | 385 |
|    | A.3. | Callback | k Function Definitions            | 385 |
|    |      | A.3.1.   | IOF Delivery Function             | 385 |
|    |      | A.3.2.   | Event Handler                     | 386 |
|    |      | A.3.3.   | Server Module Functions           | 387 |
|    | A.4. | PMIxCl   | ient                              | 400 |
|    |      | A.4.1.   | Client.init                       | 400 |
|    |      | A.4.2.   | Client.initialized                | 400 |
|    |      | A.4.3.   | Client.get_version                | 401 |
|    |      | A.4.4.   | Client.finalize                   | 401 |
|    |      | A.4.5.   | Client.abort                      | 401 |
|    |      | A.4.6.   | Client.store_internal             | 402 |
|    |      | A.4.7.   | Client.put                        | 402 |
|    |      | A.4.8.   | Client.commit                     | 403 |
|    |      | A.4.9.   | Client.fence                      | 403 |
|    |      | A.4.10.  | Client.get                        | 404 |
|    |      | A.4.11.  | Client.publish                    | 404 |
|    |      | A.4.12.  | Client.lookup                     | 405 |
|    |      | A.4.13.  | Client.unpublish                  | 405 |
|    |      | A.4.14.  | Client.spawn                      | 406 |
|    |      | A.4.15.  | Client.connect                    | 406 |

| A.4.16. | Client.disconnect               | 407 |
|---------|---------------------------------|-----|
| A.4.17. | Client.resolve_peers            | 407 |
| A.4.18. | Client.resolve_nodes            | 408 |
| A.4.19. | Client.query                    | 408 |
| A.4.20. | Client.log                      | 409 |
| A.4.21. | Client.allocate                 | 409 |
| A.4.22. | Client.job_ctrl                 | 410 |
| A.4.23. | Client.monitor                  | 410 |
| A.4.24. | Client.get_credential           | 411 |
| A.4.25. | Client.validate_credential      | 411 |
| A.4.26. | Client.group_construct          | 412 |
| A.4.27. | Client.group_invite             | 412 |
| A.4.28. | Client.group_join               | 413 |
| A.4.29. | Client.group_leave              | 414 |
| A.4.30. | Client.group_destruct           | 414 |
| A.4.31. | Client.register_event_handler   | 414 |
| A.4.32. | Client.deregister_event_handler | 415 |
| A.4.33. | Client.notify_event             | 415 |
| A.4.34. | Client.fabric_register          | 416 |
| A.4.35. | Client.fabric_update            | 416 |
| A.4.36. | Client.fabric_deregister        | 417 |
| A.4.37. | Client.fabric_get_vertex_info   | 417 |
| A.4.38. | Client.fabric_get_device_index  | 418 |
| A.4.39. | Client.error_string             | 418 |
| A.4.40. | Client.proc_state_string        | 418 |
| A.4.41. | Client.scope_string             | 419 |
| A.4.42. | Client.persistence_string       | 419 |
| A.4.43. | Client.data_range_string        | 420 |
| A.4.44. | Client.info_directives_string   | 420 |
| A.4.45. | Client.data_type_string         | 420 |
| A.4.46. | Client.alloc_directive_string   | 421 |
| A.4.47. | Client.iof_channel_string       | 421 |
| A.4.48. | Client.job_state_string         | 422 |

|    |      | A.4.49.      | Client.get_attribute_string | 422  |
|----|------|--------------|-----------------------------|------|
|    |      | A.4.50.      | Client.get_attribute_name   | 422  |
|    |      | A.4.51.      | Client.link_state_string    | 423  |
|    | A.5. | PMIxSe       | rver                        | 423  |
|    |      | A.5.1.       | Server.init                 | 423  |
|    |      | A.5.2.       | Server.finalize             | 424  |
|    |      | A.5.3.       | Server.generate_regex       | 424  |
|    |      | A.5.4.       | Server.generate_ppn         | 425  |
|    |      | A.5.5.       | Server.register_nspace      | 425  |
|    |      | A.5.6.       | Server.deregister_nspace    | 426  |
|    |      | A.5.7.       | Server.register_client      | 426  |
|    |      | A.5.8.       | Server.deregister_client    | 427  |
|    |      | A.5.9.       | Server.setup_fork           | 427  |
|    |      | A.5.10.      | Server.dmodex_request       | 428  |
|    |      | A.5.11.      | Server.setup_application    | 428  |
|    |      | A.5.12.      | Server.register_attributes  | 429  |
|    |      | A.5.13.      | Server.setup_local_support  | 429  |
|    |      | A.5.14.      | Server.iof_deliver          | 430  |
|    |      | A.5.15.      | Server.collect_inventory    | 430  |
|    |      | A.5.16.      | Server.deliver_inventory    | 431  |
|    | A.6. | PMIxTo       | ol                          | 431  |
|    |      | A.6.1.       | Tool.init                   | 431  |
|    |      | A.6.2.       | Tool.finalize               | 432  |
|    |      | A.6.3.       | Tool.connect_to_server      | 432  |
|    |      | A.6.4.       | Tool.iof_pull               | 433  |
|    |      | A.6.5.       | Tool.iof_deregister         | 433  |
|    |      | A.6.6.       | Tool.iof_push               | 434  |
|    | A.7. | Example      | e Usage                     | 434  |
|    |      | A.7.1.       | Python Client               | 435  |
|    |      | A.7.2.       | Python Server               | 437  |
| _  |      |              |                             |      |
| В. |      | •            | ements                      | 441  |
|    |      |              | 3.0                         | 441  |
|    | D 2  | <b>T</b> 7 • | 2.0                         | 4.40 |

| B.3.     | Version 1.0             | 443 |
|----------|-------------------------|-----|
| Bibliogr | Bibliography            |     |
| Index    |                         | 445 |
| Index of | ndex of APIs            |     |
| Index of | ndex of Support Macros  |     |
| Index of | ndex of Data Structures |     |
| Index of | ndex of Constants       |     |
| Index of | ndex of Attributes      |     |

#### **CHAPTER 1**

# Introduction

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

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

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

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

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

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

#### 1.1 Charter

5

6

7

8

11

12

13 14

15

16

17 18

19

29

The charter of the PMIx community is to:

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

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

#### 2 1.2 PMIx Standard Overview

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

#### 1.2.1 Who should use the standard?

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

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

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

#### 1.2.2 What is defined in the standard?

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

#### 1.2.3 What is *not* defined in the standard?

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

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

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

If an application (for example) truly relies on the <code>PMIX\_TIMEOUT</code> attribute in a call to <code>PMIx\_Fence</code>, it should set the required flag in the <code>pmix\_info\_t</code> for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, ignoring it if support is not available.

It is therefore critical that users and application implementers:

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

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

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

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

#### 1.2.4 General Guidance for PMIx Users and Implementors

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

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

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

#### 1.3 PMIx Architecture Overview

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

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

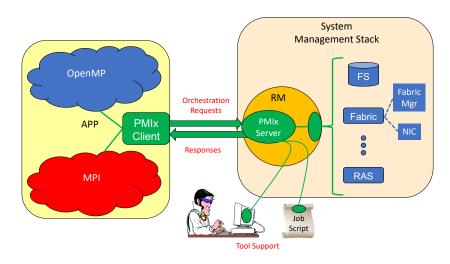


Figure 1.1.: PMIx-SMS Interactions

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

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

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

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

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

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

#### 1.3.1 The PMIx Reference Implementation (PRI)

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

- Validation of the standard.
  - No proposed change and/or extension to the PMIx standard is accepted without an accompanying prototype implementation in the PRI. This ensures that the proposal has undergone at least some minimal level of scrutiny and testing before being considered.
- Ease of adoption.
  - The PRI is designed to be particularly easy for resource managers (and the SMS in general) to adopt, thus facilitating a rapid uptake into that community for application portability. Both client and server PMIx libraries are included, along with examples of client usage and server-side integration. A list of supported environments and versions is maintained on the PMIx web site <a href="https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/">https://pmix.org/support/faq/what-apis-are-supported-on-my-rm/</a>

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

Instructions for downloading, and installing the PRI are available on the community's web site

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

#### 1.3.2 The PMIx Reference RunTime Environment (PRRTE)

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

## 1.4 Organization of this document

9

10

11

12

13

14

15

16

17

21

22

23

24

25

26

27

28

29

- 19 The remainder of this document is structured as follows:
- Introduction and Overview in Chapter 1 on page 1
  - Terms and Conventions in Chapter 2 on page 14
  - Data Structures and Types in Chapter 3 on page 20
  - PMIx Initialization and Finalization in Chapter 4 on page 115
  - Key/Value Management in Chapter 5 on page 128
  - Process Management in Chapter 6 on page 157
  - Job Management in Chapter 7 on page 183
  - Event Notification in Chapter 8 on page 219
  - Data Packing and Unpacking in Chapter 9 on page 229
  - Security in Chapter 10 on page 239
    - PMIx Server Specific Interfaces in Chapter 11 on page 248
    - Scheduler-Specific Interface in Chapter ?? on page ??
- Process Sets and Groups in Chapter 13 on page 351

- Network Coordinates in Chapter ?? on page ??
  - Python Bindings in Appendix A on page 379

1.5 Version 1.0: June 12, 2015 The PMIx version 1.0 ad hoc standard was defined in the PMIx Reference Implementation (PRI) 5 header files as part of the PRI v1.0.0 release prior to the creation of the formal PMIx 2.0 standard. 6 Below are a summary listing of the interfaces defined in the 1.0 headers. 7 Client APIs 8 PMIx Init, PMIx Initialized, PMIx Abort, PMIx Finalize - PMIx Put, PMIx Commit, 9 10 - PMIx Fence, PMIx Fence nb 11 - PMIx\_Get, PMIx\_Get\_nb 12 - PMIx Publish PMIx Publish nb 13 - PMIx\_Lookup, PMIx\_Lookup - PMIx\_Unpublish, PMIx\_Unpublish\_nb 14 15 - PMIx\_Spawn, PMIx\_Spawn\_nb - PMIx Connect, PMIx Connect nb 16 17 - PMIx Disconnect, PMIx Disconnect nb 18 - PMIx Resolve nodes, PMIx Resolve peers Server APIs 19 20 - PMIx server init, PMIx server finalize 21 - PMIx generate regex, PMIx generate ppn 22 - PMIx\_server\_register\_nspace.PMIx\_server\_deregister\_nspace - PMIx\_server\_register\_client, PMIx\_server\_deregister\_client 23 - PMIx \_server\_setup\_fork , PMIx\_server\_dmodex\_request 24 25 Common APIs 26 - PMIx\_Get\_version, PMIx\_Store\_internal, PMIx\_Error\_string 27 - PMIx Register errhandler, PMIx Deregister errhandler, PMIx Notify error

28

2

The **PMIx** Init API was subsequently modified in the PRI release v1.1.0.

#### 1 1.6 Version 2.0: Sept. 2018

The following APIs were introduced in v2.0 of the PMIx Standard: 2 3 Client APIs 4 - PMIx\_Query\_info\_nb, PMIx\_Log\_nb 5 - PMIx\_Allocation\_request\_nb, PMIx\_Job\_control\_nb, 6 PMIx Process monitor nb.PMIx Heartbeat Server APIs PMIx\_server\_setup\_application, PMIx\_server\_setup\_local\_support 9 Tool APIs - PMIx\_tool\_init, PMIx\_tool\_finalize 10 11 Common APIs 12 - PMIx Register event handler, PMIx Deregister event handler 13 - PMIx Notify event - PMIx\_Proc\_state\_string, PMIx\_Scope\_string 14 - PMIx\_Persistence\_string, PMIx\_Data\_range\_string 15 16 - PMIx\_Info\_directives\_string, PMIx\_Data\_type\_string 17 - PMIx\_Alloc\_directive\_string 18 - PMIx\_Data\_pack, PMIx\_Data\_unpack, PMIx\_Data\_copy 19 - PMIx Data print, PMIx Data copy payload 20 The **PMIx** Init API was modified in v2.0 of the standard from its ad hoc v1.0 signature to

# 24 1.7 Version 2.1; Dec. 2018

APIs were replaced.

21

22

23

25 26

27 28

29

30

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

include passing of a pmix info t array for flexibility and "future-proofing" of the API. In

addition, the PMIx Notify error, PMIx Register errhandler, and PMIx Deregister errhandler

- Clarify description of **PMIx\_Connect** and **PMIx\_Disconnect** APIs.
- Explain that values for the **PMIX\_COLLECTIVE\_ALGO** are environment-dependent
- Identify the namespace/rank values required for retrieving attribute-associated information using the PMIx\_Get API

- Provide definitions for session, job, application, and other terms used throughout the
   document
- Clarify definitions of PMIX\_UNIV\_SIZE versus PMIX\_JOB\_SIZE
  - Clarify server module function return values
    - Provide examples of the use of **PMIx Get** for retrieval of information
  - Clarify the use of PMIx Get versus PMIx Query info nb
  - Clarify return values for non-blocking APIs and emphasize that callback functions must not be invoked prior to return from the API
    - Provide detailed example for construction of the PMIx\_server\_register\_nspace input information array
    - Define information levels (e.g., **session** vs **job**) and associated attributes for both storing and retrieving values
    - Clarify roles of PMIx server library and host environment for collective operations
    - Clarify definition of **PMIX UNIV SIZE**

#### 15 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

4 5

6 7

8

9 10

11

12 13

14

16

20

21

22

23 24

25

#### 1.9 Version 3.0: Dec. 2018

- The following APIs were introduced in v3.0 of the PMIx Standard:
  Client APIs
- 4 PMIx\_Log, PMIx\_Job\_control
- 5 PMIx\_Allocation\_request, PMIx\_Process\_monitor
- 6 PMIx\_Get\_credential, PMIx\_Validate\_credential
- For the second of the
- 8 PMIx server IOF deliver
- 9 PMIx\_server\_collect\_inventory, PMIx\_server\_deliver\_inventory
- Tool APIs
- 11 PMIx\_IOF\_pull, PMIx\_IOF\_push, PMIx\_IOF\_deregister
- 12 PMIx\_tool\_connect\_to\_server
- Common APIs

14

20

21

22

23

24

25 26

30

- PMIx\_IOF\_channel\_string

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

#### 19 1.10 Version 3.1: Jan. 2019

- The v3.1 update includes clarifications and corrections from the v3.0 document:
- Direct modex upcall function ( pmix\_server\_dmodex\_req\_fn\_t ) cannot complete
  atomically as the API cannot return the requested information except via the provided callback
  function
  - Fix typo in name of **PMIX FWD STDDIAG** attribute
  - Correctly identify the information retrieval and storage attributes as "new" to v3 of the standard
  - Add missing pmix data array t definition and support macros
- Add a rule divider between implementer and host environment required attributes for clarity
- Add PMIX\_QUERY\_QUALIFIERS\_CREATE macro to simplify creation of pmix\_query\_t qualifiers
  - Add PMIX\_APP\_INFO\_CREATE macro to simplify creation of pmix\_app\_t directives

- Add new attributes to specify the level of information being requested where ambiguity may exist (see 3.4.11)
  - Add new attributes to assemble information by its level for storage where ambiguity may exist (see 3.4.12)
  - Add flag and PMIX\_INFO\_IS\_END macro for marking and detecting the end of a pmix\_info\_t array
  - Clarify that PMIX\_NUM\_SLOTS is duplicative of (a) PMIX\_UNIV\_SIZE when used at the session level and (b) PMIX\_MAX\_PROCS when used at the job and application levels, but leave it in for backward compatibility.
  - Clarify difference between PMIX\_JOB\_SIZE and PMIX\_MAX\_PROCS
  - Clarify that PMIx\_server\_setup\_application must be called per-job instead of per-application as the name implies. Unfortunately, this is a historical artifact. Note that both PMIX\_NODE\_MAP and PMIX\_PROC\_MAP must be included as input in the *info* array provided to that function. Further descriptive explanation of the "instant on" procedure will be provided in the next version of the PMIx Standard.
  - Clarify how the PMIx server expects data passed to the host by
     pmix\_server\_fencenb\_fn\_t should be aggregated across nodes, and provide a code
     snippet example

#### 19 1.11 Version 3.2: Oct. 2019

- The v3.2 update includes clarifications and corrections from the v3.1 document:
- Correct an error in the PMIx\_Allocation\_request function signature, and clarify the allocation ID attributes
- Rename the PMIX\_ALLOC\_ID attribute to PMIX\_ALLOC\_REQ\_ID to clarify that this is a string the user provides as a means to identify their request to query status
- Add a new PMIX\_ALLOC\_ID attribute that contains the identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to PMIx\_Spawn

#### 8 1.12 Version 4.0: June 2019

- The following changes were introduced in v4.0 of the PMIx Standard:
- Clarified that the PMIx\_Fence\_nb operation can immediately return

  PMIX\_OPERATION\_SUCCEEDED in lieu of passing the request to a PMIx server if only the calling process is involved in the operation

3

4

5

6

7

8

9

10 11

12

13

14

15

16

17

18

20

21

22

23

24 25

26

27

- Added the **PMIx\_Register\_attributes** API by which a host environment can register the attributes it supports for each server-to-host operation
- Added the ability to query supported attributes from the PMIx tool, client and server libraries, as
  well as the host environment via the new pmix\_regattr\_t structure. Both human-readable
  and machine-parsable output is supported. New attributes to support this operation include:
  - PMIX\_CLIENT\_ATTRIBUTES, PMIX\_SERVER\_ATTRIBUTES,
     PMIX\_TOOL\_ATTRIBUTES, and PMIX\_HOST\_ATTRIBUTES to identify which library supports the attribute; and
  - PMIX\_MAX\_VALUE , PMIX\_MIN\_VALUE , and PMIX\_ENUM\_VALUE to provide machine-parsable description of accepted values
- Add PMIX APP WILDCARD to reference all applications within a given job

- Fix signature of blocking APIs PMIx\_Allocation\_request, PMIx\_Job\_control,
   PMIx\_Process\_monitor, PMIx\_Get\_credential, and
   PMIx Validate credential to allow return of results
- Update description to provide an option for blocking behavior of the
   PMIx\_Register\_event\_handler, PMIx\_Deregister\_event\_handler,

   PMIx\_Notify\_event, PMIx\_IOF\_pull, PMIx\_IOF\_deregister, and
   PMIx\_IOF\_push APIs. The need for blocking forms of these functions was not initially anticipated but has emerged over time. For these functions, the return value is sufficient to provide the caller with information otherwise returned via callback. Thus, use of a NULL value as the callback function parameter was deemed a minimal disruption method for providing the desired capability

#### **CHAPTER 2**

# **PMIx Terms and Conventions**

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

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

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

If an application truly relies on the <code>PMIX\_TIMEOUT</code> attribute in a call to <code>PMIx\_Fence</code>, it should set the <code>required</code> flag in the <code>pmix\_info\_t</code> for that attribute. This informs the library and its SMS host that it must return an immediate error if this attribute is not supported. By not setting the flag, the library and SMS host are allowed to treat the attribute as optional, silently ignoring it if support is not available.

#### Advice to users -

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

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

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

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

- *scheduler* refers to the component of the SMS responsible for scheduling of resource allocations. This is also generally referred to as the *system workflow manager* for the purposes of this document, the *WLM* acronym will be used interchangeably to refer to the scheduler.
  - resource manager is used in a generic sense to represent the subsystem that will host the PMIx server library. This could be a vendor's RM, a programming library's RunTime Environment (RTE), or some other agent.
  - *host environment* is used interchangeably with *resource manager* to refer to the process hosting the PMIx server library.
  - *fabric* is used in a generic sense to refer to the networks within the system regardless of speed or protocol. Any use of the term *network* in the document should be considered interchangeable with *fabric*.
  - *fabric plane* refers to a collection of devices (Network Interface Cards (NICs)) and switches in a common logical or physical configuration. Fabric planes are often implemented in HPC clusters as separate overlay or physical networks controlled by a dedicated fabric manager.

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

| 20<br>21 | that applies only to programs for which the base language is C is shown as follows:   |  |  |
|----------|---|--|--|
|          | C   |  |  |
| 22       | C specific text   |  |  |
| 23       | int foo = 42;   |  |  |
| 24<br>25 | Some text is for information only, and is not part of the normative specification. These take several forms, described in their examples below: |  |  |
| 26       | Note: General text  |  |  |
|          | ▼Rationale  |  |  |
| 27       | 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   |
|-------------|---|
| 1<br>2<br>3 | 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   |
| 4           | Throughout this document, material that is primarily commentary to PMIx library implementers is   |
| 5<br>6      | 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   |
| 7           | Throughout this document, material that is primarily commentary aimed at host environments (e.g.,   |
| 8           | RMs and RTEs) providing support for the PMIx server library is set off in this section. Some  |
| 9           | readers may wish to skip these sections, while readers interested in integrating PMIx servers into  |
| 10          | their environment may want to read them carefully.  |
|             | lacklack  |

## 1 2.2 Semantics

13

14

15

- 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

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

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

#### 2.4 Procedure Conventions

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

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

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

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

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

## 2.5 Standard vs Reference Implementation

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

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

#### **CHAPTER 3**

# **Data Structures and Types**

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

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

#### Advice to PMIx library implementers —

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

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

#### Advice to users —

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

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

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

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

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

#### 3.1 Constants

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

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

Advice to PMIx library implementers -

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

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

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

Advice to PMIx library implementers -

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

### 3.1.1 PMIx Error Constants

3

4

5 6

7

8

9

10 11

12

13

16

17

18

19

20

21 22

23

24

25

26

27

28

29

30

31

32

33

34

The pmix\_status\_t structure is an int type for return status.

The tables shown in this section define the possible values for <code>pmix\_status\_t</code>. PMIx errors are required to always be negative, with 0 reserved for <code>PMIX\_SUCCESS</code>. Values in the list that were deprecated in later standards are denoted as such. Values added to the list in this version of the standard are shown in <code>magenta</code>.

### Advice to PMIx library implementers

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

### Advice to users -

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

### 3.1.1.1 General Error Constants

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

15 PMIX SUCCESS Success

PMIX ERROR General Error

PMIX ERR SILENT Silent error

PMIX\_ERR\_DEBUGGER\_RELEASE Error in debugger release

PMIX\_ERR\_PROC\_RESTART Fault tolerance: Error in process restart

**PMIX\_ERR\_PROC\_CHECKPOINT** Fault tolerance: Error in process checkpoint

**PMIX ERR PROC MIGRATE** Fault tolerance: Error in process migration

PMIX ERR PROC ABORTED Process was aborted

PMIX ERR PROC REQUESTED ABORT Process is already requested to abort

PMIX ERR PROC ABORTING Process is being aborted

PMIX ERR SERVER FAILED REQUEST Failed to connect to the server

**PMIX\_EXISTS** Requested operation would overwrite an existing value

**PMIX\_ERR\_INVALID\_CRED** Invalid security credentials

PMIX\_ERR\_HANDSHAKE\_FAILED Connection handshake failed

PMIX ERR READY FOR HANDSHAKE Ready for handshake

PMIX ERR WOULD BLOCK Operation would block

PMIX ERR UNKNOWN DATA TYPE Unknown data type

PMIX ERR PROC ENTRY NOT FOUND Process not found

PMIX\_ERR\_TYPE\_MISMATCH Invalid type

PMIX\_ERR\_UNPACK\_INADEQUATE\_SPACE Inadequate space to unpack data

| 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  |         | <b>PMIX_PROC_TERMINATED</b> 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 <b>PMIx_Finalize</b> without first calling <b>PMIx_Disconnect</b> (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 (Deprecated in PMIx C) hanged to  |
| 12 |         | PMIX_EVENT_JOB_END  |
| 13 |         | PMIX_ERR_UPDATE_ENDPOINTS Error updating endpoints  |
| 14 |         | PMIX_MODEL_DECLARED Model declared  |
| 15 |         | PMIX_GDS_ACTION_COMPLETE The global data storage (GDS) action has completed                     |
| 16 |         | PMIX_ERR_INVALID_OPERATION The requested operation is supported by the                          |
| 17 |         | implementation and host environment, but fails to meet a requirement (e.g., requesting to       |
| 18 |         | disconnect from processes without first connecting to them).                                    |
| 19 |         | PMIX_PROC_HAS_CONNECTED A tool or client has connected to the PMIx server                       |
| 20 |         | PMIX_CONNECT_REQUESTED Connection has been requested by a PMIx-based tool                       |
| 21 |         | PMIX_MODEL_RESOURCES Resource usage by a programming model has changed                          |
| 22 |         | PMIX_OPENMP_PARALLEL_ENTERED An OpenMP parallel code region has been entered                    |
| 23 |         | PMIX_OPENMP_PARALLEL_EXITED An OpenMP parallel code region has completed                        |
| 24 |         | <b>PMIX_LAUNCH_DIRECTIVE</b> Launcher directives have been received from a PMIx-enabled         |
| 25 |         | tool  |
| 26 |         | <b>PMIX_LAUNCHER_READY</b> Application launcher (e.g., mpiexec) is ready to receive directives  |
| 27 |         | from a PMIx-enabled tool  |
| 28 |         | PMIX_LAUNCH_COMPLETE A job has been launched - the nspace of the launched job will be           |
| 29 |         | included in the notification  |
| 30 |         | PMIX_OPERATION_IN_PROGRESS A requested operation is already in progress                         |
| 31 |         | <b>PMIX_OPERATION_SUCCEEDED</b> The requested operation was performed atomically - no           |
| 32 |         | callback function will be executed  |
| 33 |         | PMIX_ERR_PARTIAL_SUCCESS The operation is considered successful but not all elements            |
| 34 |         | of the operation were concluded (e.g., some members of a group construct operation chose        |
| 35 |         | not to participate)   |
| 36 |         | PMIX_ERR_DUPLICATE_KEY The provided key has already been published on a different               |
| 37 |         | data range  |
| 38 |         | PMIX_ERR_INVALID_OPERATION The requested operation is not valid - this can possibly             |
| 39 |         | indicate the inclusion of conflicting directives or a request to perform an operation that      |
| 40 |         | conflicts with an ongoing one.  |
| 41 |         | <b>PMIX_GROUP_INVITED</b> The process has been invited to join a PMIx Group - the identifier of |
| 42 |         | the group and the ID's of other invited (or already joined) members will be included in the     |
| 43 |         | notification  |
|    |         |   |

| 1  | PMIX_GROUP_LEFT A process has asynchronously left a PMIx Group - the process identifier          |
|----|--|
| 2  | of the departing process will in included in the notification                                    |
| 3  | PMIX_GROUP_MEMBER_FAILED A member of a PMIx Group has abnormally terminated                      |
| 4  | (i.e., without formally leaving the group prior to termination) - the process identifier of the  |
| 5  | failed process will in included in the notification  |
| 6  | PMIX_GROUP_INVITE_ACCEPTED A process has accepted an invitation to join a PMIx                   |
| 7  | Group - the identifier of the group being joined will be included in the notification            |
| 8  | PMIX_GROUP_INVITE_DECLINED A process has declined an invitation to join a PMIx                   |
| 9  | Group - the identifier of the declined group will be included in the notification                |
| 10 | PMIX_GROUP_INVITE_FAILED An invited process failed or terminated prior to responding             |
| 11 | to the invitation - the identifier of the failed process will be included in the notification.   |
| 12 | <b>PMIX_GROUP_MEMBERSHIP_UPDATE</b> The membership of a PMIx group has changed - the             |
| 13 | identifiers of the revised membership will be included in the notification.                      |
| 14 | PMIX_GROUP_CONSTRUCT_ABORT Any participant in a PMIx group construct operation                   |
| 15 | that returns PMIX_GROUP_CONSTRUCT_ABORT from the leader failed event handler will                |
| 16 | cause all participants to receive an event notifying them of that status. Similarly, the leader  |
| 17 | may elect to abort the procedure by either returning this error code from the handler assigned   |
| 18 | to the PMIX_GROUP_INVITE_ACCEPTED or PMIX_GROUP_INVITE_DECLINED                                  |
| 19 | codes, or by generating an event for the abort code. Abort events will be sent to all invited or |
| 20 | existing members of the group.   |
| 21 | <b>PMIX_GROUP_CONSTRUCT_COMPLETE</b> The group construct operation has completed - the           |
| 22 | final membership will be included in the notification.   |
| 23 | <b>PMIX_GROUP_LEADER_FAILED</b> The current <i>leader</i> of a group including this process has  |
| 24 | abnormally terminated - the group identifier will be included in the notification.               |
| 25 | <b>PMIX_GROUP_LEADER_SELECTED</b> A new <i>leader</i> of a group including this process has been |
| 26 | selected - the identifier of the new leader will be included in the notification                 |
| 27 | PMIX_GROUP_CONTEXT_ID_ASSIGNED A new Process Group Context                                       |
| 28 | IDentifier (PGCID) has been assigned by the host environment to a group that includes this       |
| 29 | process - the group identifier will be included in the notification.                             |
| 30 | PMIX_ERR_REPEAT_ATTR_REGISTRATION The attributes for an identical function have                  |
| 31 | already been registered at the specified level (host, server, or client)                         |
| 32 | <b>PMIX_ERR_IOF_FAILURE</b> An IO forwarding operation failed - the affected channel will be     |
| 33 | included in the notification   |
| 34 | PMIX_ERR_IOF_COMPLETE IO forwarding of the standard input for this process has                   |
| 35 | completed - i.e., the stdin file descriptor has closed   |
| 36 | PMIX_ERR_GET_MALLOC_REQD The data returned by PMIx_Get contains values that                      |
| 37 | required dynamic memory allocations (i.e., "malloc"), despite a request for static pointers to   |
| 38 | the values in the key-value store. User is responsible for releasing the memory when done        |
| 39 | with the information.  |
|    |  |

## 3.1.1.3 Job-related constants

40

41

42

The following constants indicate that a session or job has started and normally completed:

PMIX\_EVENT\_JOB\_START The job has started

| 1        |         | PMIX_EVENT_JOB_END The job has normally terminated   |
|----------|---------|--|
| 2        |         | PMIX_EVENT_SESSION_START The session has started   |
| 3        |         | PMIX_EVENT_SESSION_END The session has normally terminated   |
| 4        |         | The next set of constants are used to indicate that a job has abnormally terminated and to convey      |
| 5        |         | some information as to the cause:  |
| 6        |         | PMIX_ERR_JOB_APP_NOT_EXECUTABLE The specified application executable either                            |
| 7        |         | could not be found, or lacks execution privileges.   |
| 8        |         | PMIX_ERR_NO_EXE_SPECIFIED The job request did not specify an executable.                               |
| 9        |         | PMIX_ERR_JOB_FAILED_TO_MAP The launcher was unable to map the processes for the                        |
| 0        |         | specified job request.   |
| 11       |         | PMIX_ERR_JOB_CANCELED The job was canceled by the host environment                                     |
| 12       |         | PMIX_ERR_JOB_FAILED_TO_LAUNCH One or more processes in the job request failed to                       |
| 13       |         | launch   |
| 14       |         | <b>PMIX_ERR_JOB_ABORTED</b> One or more processes in the job called abort, causing the job to          |
| 15       |         | be terminated  |
| 16       |         | PMIX_ERR_JOB_KILLED_BY_CMD The job was killed by user command  |
| 17       |         | PMIX_ERR_JOB_ABORTED_BY_SIG The job was aborted due to receipt of an error signal                      |
| 8        |         | (e.g., SIGKILL)  |
| 19       |         | PMIX_ERR_JOB_TERM_WO_SYNC The job was terminated due to one or more processes                          |
| 20       |         | exiting without first calling PMIx_Finalize  |
| 21       |         | PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED The job was terminated due to one or more                           |
| 22       |         | processes exceeding a specified sensor limit   |
| 23       |         | PMIX_ERR_JOB_NON_ZERO_TERM The job was terminated due to one or more processes                         |
| 24       |         | exiting with a non-zero status   |
| 25       |         | PMIX_ERR_JOB_ALLOC_FAILED The job request could not be executed due to failure to                      |
| 26<br>27 |         | obtain the specified allocation  |
|          |         | PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT The job was aborted due to receipt of a                              |
| 28<br>29 | 3.1.1.4 | System event System error constants  |
| 30       |         | PMIX_ERR_SYS_BASE Mark the beginning of a dedicated range of constants for system event                |
| 31       |         | reporting.   |
| 32       |         | PMIX_ERR_NODE_DOWN A node has gone down - the identifier of the affected node will be                  |
| 33       |         | included in the notification   |
| 34       |         | <b>PMIX_ERR_NODE_OFFLINE</b> A node has been marked as <i>offline</i> - the identifier of the affected |
| 35       |         | node will be included in the notification  |
| 36       |         | <b>PMIX_ERR_SYS_OTHER</b> Mark the end of a dedicated range of constants for system event              |
| 37<br>38 | 3.1.1.5 | reporting.  Event handler error constants  |
|          | 3       |  |
| 39       |         | PMIX_EVENT_NO_ACTION_TAKEN Event handler: No action taken  |
| 10       |         | PMIX_EVENT_PARTIAL_ACTION_TAKEN Event handler: Partial action taken                                    |
| 11       |         | PMIX_EVENT_ACTION_DEFERRED Event handler: Action deferred  |
| 12       |         | PMIX_EVENT_ACTION_COMPLETE Event handler: Action complete  |

| 1                | 3.1.1.6   | User-Defined Error Constants   |
|------------------|-----------|--|
| 2<br>3<br>4      |           | PMIx establishes an error code boundary for constants defined in the PMIx standard. Negative values larger than this (and any positive values greater than zero) are guaranteed not to conflict with PMIx values.  |
| 5<br>6<br>7<br>8 |           | PMIX_EXTERNAL_ERR_BASE A starting point for user-level defined error constants. Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions should always be based on the PMIX_EXTERNAL_ERR_BASE constant and not a specific value as the value of the constant may change. |
| 9                | 3.1.2     | Macros for use with PMIx constants   |
| 10               | 3.1.2.1   | Detect system event constant   |
| 11<br>12         |           | Test a given error constant to see if it falls within the dedicated range of constants for system event reporting.   |
|                  | PMIx v2.2 | C  |
| 13               |           | PMIX_SYSTEM_EVENT(a)   |
| 14<br>15         |           | IN a Error constant to be checked ( pmix_status_t )  |
| 16<br>17         |           | Returns $\verb true $ if the provided values falls within the dedicated range of constants for system event reporting  |
| 18               | 3.2       | Data Types   |
| 19<br>20         |           | This section defines various data types used by the PMIx APIs. The version of the standard in which a particular data type was introduced is shown in the margin.  |
| 21               | 3.2.1     | Key Structure  |
| 22<br>23<br>24   |           | The <code>pmix_key_t</code> structure is a statically defined character array of length <code>PMIX_MAX_KEYLEN</code> +1, thus supporting keys of maximum length <code>PMIX_MAX_KEYLEN</code> while preserving space for a mandatory <code>NULL</code> terminator.  |
|                  | PMIx v2.0 | C  |
| 25               |           | <pre>typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];</pre>   |

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 1 2 **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 3 4 compatibility. 5 Passing a pmix\_key\_t value to the standard *sizeof* utility can result in compiler warnings of 6 incorrect returned value. Users are advised to avoid using sizeof(pmix\_key\_t) and instead rely on 7 the PMIX MAX KEYLEN constant. Key support macro 3.2.1.1 Compare the key in a **pmix\_info\_t** to a given value 9 PMIx v3.0 PMIX\_CHECK\_KEY(a, b) 10 IN 11 12 Pointer to the structure whose key is to be checked (pointer to pmix info t) IN 13 14 String value to be compared against (char\*) 15 Returns **true** if the key matches the given value 3.2.2 Namespace Structure 17 The **pmix\_nspace\_t** structure is a statically defined character array of length 18 PMIX MAX NSLEN +1, thus supporting namespaces of maximum length PMIX MAX NSLEN while preserving space for a mandatory **NULL** terminator. 19 PMIx v2.0 20 typedef char pmix nspace t[PMIX MAX NSLEN+1]; 21 Characters in the namespace must be standard alphanumeric values supported by common utilities 22 such as stremp.

|           | Advice to users  |
|-----------|--|
|           | References to namespace values in PMIx v1 were defined simply as an array of characters of size <b>PMIX_MAX_NSLEN+1</b> . The <b>pmix_nspace_t</b> 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 <b>pmix_nspace_t</b> value to the standard <i>sizeof</i> utility can result in compiler warnings of incorrect returned value. Users are advised to avoid using <i>sizeof(pmix_nspace_t)</i> and instead rely on the <b>PMIX_MAX_NSLEN</b> constant.  |
| 3.2.2.1   | Namespace support macro  |
|           | Compare the string in a pmix_nspace_t to a given value   |
| PMIx v3.0 | C  |
|           | PMIX_CHECK_NSPACE(a, b)  |
|           | C  |
|           | <pre>IN     a      Pointer to the structure whose value is to be checked (pointer to pmix_nspace_t) IN     b      String value to be compared against (char*)</pre>  |
|           | Returns <b>true</b> if the namespace matches the given value   |
| 3.2.3     | Rank Structure   |
|           | The pmix_rank_t structure is a uint32_t type for rank values.  |
| PMIx v1.0 | C  |
|           | <pre>typedef uint32_t pmix_rank_t;</pre>   |
|           | The following constants can be used to set a variable of the type <code>pmix_rank_t</code> . All definitions were introduced in version 1 of the standard unless otherwise marked. Valid rank values start at zero.  |
|           | <pre>PMIX_RANK_UNDEF</pre>   |
|           | <b>PMIX_RANK_LOCAL_NODE</b> Special rank value used to define groups of ranks. This constant defines the group of all ranks on a local node  |
|           | PMIx v3.0  |

Special rank value used to define groups of rankss. This 1 PMIX\_RANK\_LOCAL\_PEERS 2 constant defines the group of all ranks on a local node within the same namespace as the 3 current process. PMIX RANK INVALID An invalid rank value. 4 5 PMIX\_RANK\_VALID Define an upper boundary for valid rank values. 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 v1.0typedef struct pmix\_proc { 9 10 pmix\_nspace\_t nspace; 11 pmix\_rank\_t rank; 12 } pmix\_proc\_t; 3.2.5 **Process structure support macros** The following macros are provided to support the **pmix proc** t structure. 14 3.2.5.1 Initialize the pmix\_proc\_t structure 15 PMIX PROC CONSTRUCT 16 17 Initialize the **pmix\_proc\_t** fields PMIx v1.0PMIX PROC CONSTRUCT (m) 18 IN 19 20 Pointer to the structure to be initialized (pointer to pmix\_proc\_t) 3.2.5.2 Destruct the pmix\_proc\_t structure 21

There is nothing to release here as the fields in **pmix proc t** are all declared *static*. However,

the macro is provided for symmetry in the code and for future-proofing should some allocated field

22

23

24

be included some day.

```
3.2.5.3 Create a pmix_proc_t array
 2
              Allocate and initialize an array of pmix_proc_t structures
   PMIx v1.0
 3
              PMIX_PROC_CREATE(m, n)
              INOUT m
 4
                   Address where the pointer to the array of pmix proc t structures shall be stored (handle)
 5
 6
              IN
 7
                   Number of structures to be allocated (size_t)
    3.2.5.4
              Free a pmix_proc_t array
 9
              Release an array of pmix_proc_t structures
   PMIx v1.0
10
              PMIX_PROC_FREE(m, n)
              IN
11
                   Pointer to the array of pmix_proc_t structures (handle)
12
              IN
13
14
                   Number of structures in the array (size t)
    3.2.5.5
              Load a pmix_proc_t structure
15
16
              Load values into a pmix_proc_t
   PMIx v2.0
17
              PMIX_PROC_LOAD(m, n, r)
              IN
18
19
                   Pointer to the structure to be loaded (pointer to pmix_proc_t)
20
              IN
                   Namespace to be loaded ( pmix_nspace_t )
21
              IN
22
                   Rank to be assigned (pmix rank t)
23
```

### 1 3.2.5.6 Compare identifiers

2 Compare two pmix proc t identifiers PMIx v3.0 3 PMIX CHECK PROCID(a, b) IN 4 5 Pointer to a structure whose ID is to be compared (pointer to pmix proc t) 6 IN 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 3.2.6 **Process State Structure** 12 PMIx v2.0The pmix proc state t structure is a uint8 t type for process state values. The following constants can be used to set a variable of the type pmix\_proc\_state\_t . All values were 13 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 updated process state values during the process lifetime. This is particularly true of states in the 16 17 launch process, and for short-lived processes. 18 PMIX\_PROC\_STATE\_UNDEF Undefined process state 19 PMIX PROC STATE PREPPED Process is ready to be launched Process launch is underway 20 PMIX PROC STATE LAUNCH UNDERWAY 21 PMIX\_PROC\_STATE\_RESTART Process is ready for restart 22 PMIX\_PROC\_STATE\_TERMINATE Process is marked for termination 23 PMIX PROC STATE RUNNING Process has been locally fork'ed by the RM 24 PMIX PROC STATE CONNECTED Process has connected to PMIx server Define a "boundary" between the terminated states 25 PMIX PROC STATE UNTERMINATED

and PMIX\_PROC\_STATE\_CONNECTED so users can easily and quickly determine if a

process is still running or not. Any value less than this constant means that the process has not

terminated abnormally.

terminated.

26 27

28 29

30

31

32

```
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
                                                    Process called PMIx Abort
              PMIX PROC STATE CALLED 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
              PMIX_PROC_STATE_TERM_NON_ZERO
                                                     Process exited with a non-zero status
11
              PMIX_PROC_STATE_FAILED_TO LAUNCH
12
                                                         Unable to launch process
```

### 3.2.7 Process Information Structure

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

```
PMIx v2.0
            typedef struct pmix_proc_info {
16
                /** Process structure */
17
18
                pmix_proc_t proc;
                /** Hostname where process resides */
19
20
                char *hostname;
21
                /** Name of the executable */
22
                char *executable_name;
                /** Process ID on the host */
23
24
                pid t pid;
                /** Exit code of the process. Default: 0 */
25
26
                int exit code;
27
                /** Current state of the process */
28
                pmix proc state t state;
            } pmix proc info t;
29
```

# 3.2.8 Process Information Structure support macros

31

The following macros are provided to support the **pmix\_proc\_info\_t** structure.

```
3.2.8.1
              Initialize the pmix_proc_info_t structure
              Initialize the pmix proc info t fields
   PMIx v2.0
 3
              PMIX PROC INFO CONSTRUCT (m)
              IN
 4
 5
                   Pointer to the structure to be initialized (pointer to pmix_proc_info_t)
    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)
              IN
 9
10
                   Pointer to the structure to be destructed (pointer to pmix_proc_info_t)
    3.2.8.3
              Create a pmix_proc_info_t array
11
12
              Allocate and initialize a pmix proc info t array
   PMIx v2.0
13
              PMIX PROC INFO CREATE (m, n)
              INOUT m
14
                   Address where the pointer to the array of pmix_proc_info_t structures shall be stored
15
16
              IN
17
                   n
                   Number of structures to be allocated (size t)
18
    3.2.8.4
              Free a pmix proc info t array
19
              Release an array of pmix_proc_info_t structures
20
   PMIx v2.0
21
              PMIX_PROC_INFO_FREE(m, n)
22
              IN
                   Pointer to the array of pmix_proc_info_t structures (handle)
23
24
              IN
                   Number of structures in the array (size t)
25
```

## 1 3.2.9 Scope of Put Data

The pmix\_scope\_t structure is a uint8\_t type that defines the scope for data passed to PMIx v1.03 **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. Data marked in this way will not be included in data packages sent to remote requestors — 10 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.

### 19 3.2.10 Job State Structure

PMIX INTERNAL

processes.

17 *PMIx v2.0* 

18

23 24

25

The pmix\_job\_state\_t structure is a uint8\_t type for job state values. The following constants can be used to set a variable of the type pmix\_job\_state\_t. All values were originally defined in version 4 of the standard unless otherwise marked.

#### Advice to users

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

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

```
26
               PMIX_JOB_STATE_UNDEF
                                             Undefined job state
27
               PMIX JOB STATE PREPPED
                                                Job is ready to be launched
28
               PMIX_JOB_STATE_LAUNCH_UNDERWAY
                                                          Job launch is underway
29
                                                All processes in the job have been spawned
               PMIX_JOB_STATE_RUNNING
                                                  All processes in the job have been suspended
30
               PMIX JOB STATE SUSPENDED
               PMIX JOB STATE CONNECTED
                                                  All processes in the job have connected to their PMIx
31
32
                    server
33
               PMIX JOB STATE UNTERMINATED
                                                      Define a "boundary" between the terminated states
34
                    and PMIX JOB STATE TERMINATED so users can easily and quickly determine if a job
35
                    is still running or not. Any value less than this constant means that the job has not terminated.
```

1 PMIX\_JOB\_STATE\_TERMINATED All processes in the job have terminated and are no
2 longer running - typically will be accompanied by the job exit status in response to a query
3 PMIX\_JOB\_STATE\_TERMINATED\_WITH\_ERROR Define a boundary so users can easily
4 and quickly determine if a job abnormally terminated - typically will be accompanied by a
5 job-related error code in response to a query Any value above this constant means that the job
6 terminated abnormally.

# 7 3.2.11 Range of Published Data

The pmix\_data\_range\_t structure is a uint8\_t type that defines a range for data published via functions other than PMIx\_Put - e.g., the PMIx\_Publish API. The following constants can be used to set a variable of the type pmix\_data\_range\_t. Several values were initially defined in version 1 of the standard but subsequently renamed and other values added in version 2. Thus, all values shown below are as they were defined in version 2 except where noted.

**PMIX\_RANGE\_UNDEF** Undefined range

**PMIX\_RANGE\_RM** Data is intended for the host resource manager.

**PMIX\_RANGE\_LOCAL** Data is only available to processes on the local node.

**PMIX\_RANGE\_NAMESPACE** Data is only available to processes in the same namespace.

**PMIX\_RANGE\_SESSION** Data is only available to all processes in the session.

**PMIX\_RANGE\_GLOBAL** Data is available to all processes.

**PMIX\_RANGE\_CUSTOM** Range is specified in the **pmix\_info\_t** associated with this call.

**PMIX RANGE PROC LOCAL** Data is only available to this process.

PMIX RANGE INVALID Invalid value

#### Advice to users

The names of the **pmix\_data\_range\_t** values changed between version 1 and version 2 of the standard, thereby breaking backward compatibility

# 24 3.2.12 Data Persistence Structure

The pmix\_persistence\_t structure is a uint8\_t type that defines the policy for data published by clients via the PMIx\_Publish API. The following constants can be used to set a variable of the type pmix\_persistence\_t. All definitions were introduced in version 1 of the standard unless otherwise marked.

**PMIX\_PERSIST\_INDEF** Retain data until specifically deleted.

**PMIX PERSIST FIRST READ** Retain data until the first access, then the data is deleted.

**PMIX PERSIST PROC** Retain data until the publishing process terminates.

**PMIX PERSIST APP** Retain data until the application terminates.

**PMIX\_PERSIST\_SESSION** Retain data until the session/allocation terminates.

PMIX\_PERSIST\_INVALID Invalid value

13

14

15

16

17

18 19

20

21

29

30

31

32

33

34

# 3.2.13 Data Array Structure

```
PMIx v2.0
              typedef struct pmix_data_array
 2
 3
                   pmix_data_type_t type;
 4
                   size t size;
 5
                   void *array;
 6
               pmix_data_array_t;
 7
              The pmix_data_array_t structure is used to pass arrays of related values. Any PMIx data
 8
              type (including complex structures) can be included in the array.
    3.2.14
               Data array structure support macros
              The following macros are provided to support the pmix_data_array_t structure.
10
    3.2.14.1
              Initialize the pmix data array t structure
              Initialize the pmix_data_array_t fields, allocating memory for the array itself.
12
   PMIx v2.2
              PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)
13
14
              IN
15
                   Pointer to the structure to be initialized (pointer to pmix data array t)
16
              IN
17
                   Number of elements in the array (size_t)
              IN
18
19
                   PMIx data type for the array elements ( pmix_data_type_t )
    3.2.14.2
               Destruct the pmix_data_array_t structure
20
21
              Destruct the pmix_data_array_t fields, releasing the array's memory.
   PMIx v2.2
22
              PMIX DATA ARRAY DESTRUCT (m)
              IN
23
                   m
24
                   Pointer to the structure to be destructed (pointer to pmix data array t)
```

#### 3.2.14.3 Create and initialize a pmix\_data\_array\_t object 2 Allocate and initialize a pmix\_data\_array\_t structure and initialize it, allocating memory for 3 the array itself as well. PMIx v2.2PMIX DATA ARRAY CREATE (m, n, t) 4 INOUT m 5 6 Address where the pointer to the pmix\_data\_array\_t structure shall be stored (handle) 7 IN 8 Number of elements in the array (size\_t) IN 9 PMIx data type for the array elements ( pmix\_data\_type\_t ) 10 3.2.14.4 Free a pmix data array t object 11 12 Release a pmix\_data\_array\_t structure, including releasing the array's memory. PMIx v2.213 PMIX\_DATA\_ARRAY\_FREE (m) 14 IN 15 Pointer to the **pmix** data array t structure (handle) 3.2.15 Value Structure 17 The pmix value t structure is used to represent the value passed to PMIx Put and retrieved by **PMIx Get**, as well as many of the other PMIx functions. 18 19 A collection of values may be specified under a single key by passing a pmix value t containing an array of type pmix data array t, with each array element containing its own 20 object. All members shown below were introduced in version 1 of the standard unless otherwise 21 22 marked.

PMIx v1.0

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

# 7 3.2.16 Value structure support macros

38

The following macros are provided to support the **pmix\_value\_t** structure.

| 1        | 3.2.16.1  | Initialize the pmix_value_t structure   |
|----------|-----------|---|
| 2        |           | Initialize the pmix_value_t fields  |
|          | PMIx v1.0 | C   |
| 3        |           | PMIX_VALUE_CONSTRUCT (m)  |
|          |           | C -   |
| 4<br>5   |           | <pre>IN m Pointer to the structure to be initialized (pointer to pmix_value_t)</pre>                |
| 6        | 3.2.16.2  | Destruct the pmix_value_t structure   |
| 7        | PMIx v1.0 | Destruct the pmix_value_t fields  C   |
| 8        |           | PMIX_VALUE_DESTRUCT (m)   |
| 9        |           | <pre>IN m Pointer to the structure to be destructed (pointer to pmix_value_t)</pre>                 |
| 11       | 3.2.16.3  | Create a pmix_value_t array   |
| 12       |           | Allocate and initialize an array of pmix_value_t structures   |
|          | PMIx v1.0 | C   |
| 13       |           | PMIX_VALUE_CREATE (m, n)  |
| 14<br>15 |           | INOUT m  Address where the pointer to the array of pmix_value_t structures shall be stored (handle) |
| 16<br>17 |           | IN n  Number of structures to be allocated (size_t)   |
| 18       | 3.2.16.4  | Free a pmix_value_t array   |
| 19       |           | Release an array of <b>pmix_value_t</b> structures  |
|          | PMIx v1.0 | C —   |
| 20       |           | PMIX_VALUE_FREE (m, n)  |
| 21       |           | IN m  |
| 22       |           | Pointer to the array of <pre>pmix_value_t</pre> structures (handle)                                 |
| 23<br>24 |           | Number of structures in the array (size_t)  |

40

#### 3.2.16.5 Load a value structure 2 Summary Load data into a **pmix value** t structure. 3 PMIx v2.0 PMIX VALUE\_LOAD(v, d, t); 4 IN 5 6 The pmix value t into which the data is to be loaded (pointer to pmix value t) 7 IN 8 Pointer to the data value to be loaded (handle) IN 9 Type of the provided data value (pmix data type t) 10 **Description** 11 12 This macro simplifies the loading of data into a pmix value t by correctly assigning values to the structure's fields. 13 The data will be copied into the **pmix\_value\_t** - thus, any data stored in the source value can be 14 modified or free'd without affecting the copied data once the macro has completed. 15 3.2.16.6 Unload a pmix\_value\_t structure 17 Summary 18 Unload data from a pmix value t structure. PMIx v2.2 19 PMIX\_VALUE\_UNLOAD(r, v, d, t); OUT r 20 21 Status code indicating result of the operation pmix\_status\_t 22 IN The pmix\_value\_t from which the data is to be unloaded (pointer to pmix\_value\_t) 23 INOUT d 24 Pointer to the location where the data value is to be returned (handle) 25 26 INOUT t 27 Pointer to return the data type of the unloaded value (handle)

```
Description
 1
 2
               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
 3
               pmix_value_t will not be altered.
 4
    3.2.16.7 Transfer data between pmix_value_t structures
 6
               Summary
 7
               Transfer the data value between two pmix value t structures.
   PMIx v2.0
 8
               PMIX VALUE XFER(r, d, s);
               OUT r
 9
                    Status code indicating success or failure of the transfer ( pmix_status_t )
10
               IN
11
                    Pointer to the pmix_value_t destination (handle)
12
13
               IN
                    Pointer to the pmix_value_t source (handle)
14
               Description
15
               This macro simplifies the transfer of data between two pmix_value_t structures, ensuring that
16
17
               all fields are properly copied.

    Advice to users -

18
               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.
19
```

```
3.2.16.8 Retrieve a numerical value from a pmix_value_t
2
              Retrieve a numerical value from a pmix_value_t structure
   PMIx v3.0
 3
              PMIX VALUE GET NUMBER(s, m, n, t)
              OUT s
4
5
                   Status code for the request ( pmix_status_t )
              IN
6
7
                   Pointer to the pmix_value_t structure (handle)
8
              OUT n
9
                   Variable to be set to the value (match expected type)
              IN
10
11
                   Type of number expected in m (pmix data type t)
12
              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
13
              PMIX ERR BAD PARAM if it doesn't
14
    3.2.17
              Info Structure
15
16
              The pmix info t structure defines a key/value pair with associated directive. All fields were
17
              defined in version 1.0 unless otherwise marked.
   PMIx v1.0
              typedef struct pmix_info_t {
18
                   pmix_key_t key;
19
                   pmix_info_directives_t flags; // version 2.0
20
21
                   pmix_value_t value;
22
              } pmix_info_t;
    3.2.18
              Info structure support macros
              The following macros are provided to support the pmix_info_t structure.
24
    3.2.18.1
              Initialize the pmix info t structure
25
26
              Initialize the pmix info t fields
   PMIx v1.0
27
              PMIX_INFO_CONSTRUCT (m)
              IN
28
29
                   Pointer to the structure to be initialized (pointer to pmix info t)
```

```
3.2.18.2 Destruct the pmix_info_t structure
              Destruct the pmix_info_t fields
   PMIx v1.0
3
              PMIX INFO DESTRUCT (m)
              IN
4
                   Pointer to the structure to be destructed (pointer to pmix_info_t)
5
    3.2.18.3 Create a pmix_info_t array
              Allocate and initialize an array of pmix_info_t structures
7
   PMIx v1.0
8
              PMIX_INFO_CREATE(m, n)
              INOUT m
9
                   Address where the pointer to the array of pmix info t structures shall be stored (handle)
10
11
              IN
12
                   Number of structures to be allocated (size t)
   3.2.18.4 Free a pmix_info_t array
13
              Release an array of pmix_info_t structures
14
   PMIx v1.0
15
              PMIX INFO FREE (m, n)
              IN
16
                   Pointer to the array of pmix_info_t structures (handle)
17
              IN
18
                   Number of structures in the array (size_t)
19
```

#### 3.2.18.5 Load key and value data into a pmix\_info\_t PMIx v1.0 2 PMIX INFO LOAD (v, k, d, t); IN 3 4 Pointer to the pmix info t into which the key and data are to be loaded (pointer to pmix info t) 5 IN k 6 7 String key to be loaded - must be less than or equal to PMIX\_MAX\_KEYLEN in length (handle) 8 IN 9 d 10 Pointer to the data value to be loaded (handle) 11 IN Type of the provided data value ( pmix\_data\_type\_t ) 12 This macro simplifies the loading of key and data into a **pmix\_info\_t** by correctly assigning 13 values to the structure's fields. 14 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. 3.2.18.6 Copy data between pmix\_info\_t structures 18 19 Copy all data (including key, value, and directives) between two pmix info t structures. PMIx v2.0 20 PMIX INFO XFER(d, s); IN d 21 Pointer to the destination pmix\_info\_t (pointer to pmix\_info\_t) 22 23 IN Pointer to the source pmix\_info\_t (pointer to pmix\_info\_t) 24 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 thus, the source pmix\_info\_t may be free'd without affecting the copied data once the macro 27 has completed. 28

#### 3.2.18.7 Test a boolean pmix\_info\_t 2 A special macro for checking if a boolean **pmix info** t is true PMIx v2.03 PMIX INFO TRUE (m) IN 4 m 5 Pointer to a pmix\_info\_t structure (handle) A pmix\_info\_t structure is considered to be of type PMIX\_BOOL and value true if: 6 7 • the structure reports a type of **PMIX UNDEF**, or 8 • the structure reports a type of **PMIX BOOL** and the data flag is **true** Info Type Directives 3.2.19 10 *PMIx v2.0* The pmix info directives t structure is a uint32 t type that defines the behavior of command directives via pmix info t arrays. By default, the values in the pmix info t 11 array passed to a PMIx are optional. 12 Advice to users -A PMIx implementation or PMIx-enabled RM may ignore any pmix\_info\_t value passed to a 13 14 PMIx API if it is not explicitly marked as PMIX\_INFO\_REQD. This is because the values specified default to optional, meaning they can be ignored. This may lead to unexpected behavior if 15 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 PMIX INFO REQUIRED macro. 18 — 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

use by implementers. Implementers are advised to use the provided PMIX\_INFO\_IS\_REQUIRED

macro for testing this flag, and must return PMIX ERR NOT SUPPORTED as soon as possible to

the caller if the required behavior is not supported.

21 22

23

The following constants were introduced in version 2.0 (unless otherwise marked) and can be used 1 to set a variable of the type pmix\_info\_directives\_t. 2 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. PMIX INFO ARRAY END Mark that this **pmix\_info\_t** struct is at the end of an array 5 created by the PMIX\_INFO\_CREATE macro. This is a bit-mask value. 6 Advice to PMIx server hosts Host environments are advised to use the provided PMIX\_INFO\_IS\_REQUIRED macro for 7 testing this flag and must return PMIX\_ERR\_NOT\_SUPPORTED as soon as possible to the caller 8 if the required behavior is not supported. 9 3.2.20 Info Directive support macros The following macros are provided to support the setting and testing of **pmix\_info\_t** directives. 11 3.2.20.1 Mark an info structure as required 13 Summary Set the **PMIX\_INFO\_REQD** flag in a **pmix\_info\_t** structure. 14 PMIx v2.015 PMIX INFO\_REQUIRED(info); IN 16 info 17 Pointer to the pmix info t (pointer to pmix info t) This macro simplifies the setting of the PMIX\_INFO\_REQD flag in pmix\_info\_t structures. 18 3.2.20.2 Mark an info structure as optional 19 20 Summary Unsets the **PMIX\_INFO\_REQD** flag in a **pmix\_info\_t** structure. 21 PMIx v2.022 PMIX\_INFO\_OPTIONAL(info); IN 23 info Pointer to the pmix info t (pointer to pmix info t) 24 25 This macro simplifies marking a **pmix info** t structure as *optional*.

#### 3.2.20.3 Test an info structure for required directive Summary 2 Test the PMIX\_INFO\_REQD flag in a pmix\_info\_t structure, returning true if the flag is set. 3 PMIx v2.0PMIX INFO IS REQUIRED (info); 4 5 IN info Pointer to the pmix\_info\_t (pointer to pmix\_info\_t) 6 7 This macro simplifies the testing of the required flag in **pmix info** t structures. 3.2.20.4 Test an info structure for *optional* directive Summary 9 Test a pmix info t structure, returning true if the structure is optional. 10 PMIx v2.0PMIX INFO IS OPTIONAL (info); 11 IN info 12 Pointer to the pmix\_info\_t (pointer to pmix\_info\_t) 13 14 Test the PMIX\_INFO\_REQD flag in a pmix\_info\_t structure, returning true if the flag is not 15 3.2.20.5 Test an info structure for end of array directive 16 17 Summary 18 Test a pmix info t structure, returning true if the structure is at the end of an array created by the PMIX INFO CREATE macro. 19 PMIx v2.2PMIX\_INFO\_IS\_END(info); 20 IN 21 info 22 Pointer to the pmix\_info\_t (pointer to pmix\_info\_t) This macro simplifies the testing of the end-of-array flag in pmix\_info\_t structures. 23

#### **Job Allocation Directives** 1 3.2.21

The pmix\_alloc\_directive\_t structure is a uint8\_t type that defines the behavior of 2 PMIx v2.0allocation requests. The following constants can be used to set a variable of the type 3 4 pmix\_alloc\_directive\_t . All definitions were introduced in version 2 of the standard unless otherwise marked. 5 6 PMIX ALLOC NEW A new allocation is being requested. The resulting allocation will be 7 disjoint (i.e., not connected in a job sense) from the requesting allocation. 8 PMIX ALLOC EXTEND Extend the existing allocation, either in time or as additional 9 resources. 10 PMIX ALLOC RELEASE Release part of the existing allocation. Attributes in the 11 accompanying pmix\_info\_t array may be used to specify permanent release of the 12 identified resources, or "lending" of those resources for some period of time. 13 PMIX\_ALLOC\_REAQUIRE Reacquire resources that were previously "lent" back to the 14 scheduler. 15 PMIX\_ALLOC\_EXTERNAL A value boundary above which implementers are free to define 16 their own directive values.

# 3.2.22 IO Forwarding Channels

- The pmix\_iof\_channel\_t structure is a uint16\_t type that defines a set of bit-mask flags 18 PMIx v3.019 for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple 20 channels. 21 Forward no channels PMIX\_FWD\_NO\_CHANNELS 22 Forward stdin PMIX\_FWD\_STDIN\_CHANNEL
- 23 PMIX\_FWD\_STDOUT\_CHANNEL Forward stdout 24 Forward stderr PMIX FWD STDERR CHANNEL
- 25 Forward stddiag, if available PMIX FWD STDDIAG CHANNEL 26
  - Forward all available channels PMIX FWD ALL CHANNELS

#### **Environmental Variable Structure** 3.2.23

Define a structure for specifying environment variable modifications. Standard environment 28 *PMIx v3.0* 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;
    3.2.24 Environmental variable support macros
7
              The following macros are provided to support the pmix_envar_t structure.
    3.2.24.1
               Initialize the pmix_envar_t structure
9
              Initialize the pmix_envar_t fields
   PMIx v3.0
10
              PMIX_ENVAR_CONSTRUCT (m)
11
              IN
12
                  Pointer to the structure to be initialized (pointer to pmix envar t)
               Destruct the pmix_envar_t structure
    3.2.24.2
13
              Clear the pmix_envar_t fields
14
   PMIx v3.0
15
              PMIX_ENVAR_DESTRUCT (m)
              IN
16
17
                  Pointer to the structure to be destructed (pointer to pmix_envar_t)
   3.2.24.3 Create a pmix_envar_t array
18
19
              Allocate and initialize an array of pmix_envar_t structures
   PMIx v3.0
20
              PMIX_ENVAR_CREATE(m, n)
              INOUT m
21
22
                  Address where the pointer to the array of pmix envar t structures shall be stored (handle)
23
              IN
24
                  Number of structures to be allocated (size t)
```

```
3.2.24.4 Free a pmix_envar_t array
2
              Release an array of pmix_envar_t structures
   PMIx v3.0
 3
              PMIX_ENVAR_FREE(m, n)
              IN
4
5
                  Pointer to the array of pmix_envar_t structures (handle)
6
              IN
7
                  Number of structures in the array (size_t)
   3.2.24.5 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
                  Pointer to the structure to be loaded (pointer to pmix_envar_t)
12
              IN
13
14
                  Environmental variable name (char*)
              IN
15
                  Value of variable (char*)
16
              IN
17
                  Separator character (char)
18
   3.2.25
             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 {
22
                  pmix_proc_t proc;
23
                  pmix_key_t key;
24
                  pmix_value_t value;
25
              } pmix pdata t;
```

# 26 3.2.26 Lookup data structure support macros

27

The following macros are provided to support the **pmix pdata** t structure.

| 1  | 3.2.26.1  | 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 <b>pmix_pdata_t</b> )                      |
| 6  | 3.2.26.2  | Destruct the pmix_pdata_t structure   |
| 7  |           | Destruct the <pre>pmix_pdata_t</pre> fields   |
|    | PMIx v1.0 | C   |
| 8  |           | PMIX_PDATA_DESTRUCT (m)   |
|    |           | C   |
| 9  |           | IN m  |
| 0  |           | Pointer to the structure to be destructed (pointer to <pre>pmix_pdata_t</pre> )                   |
| 1  | 3.2.26.3  | 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 <b>pmix_pdata_t</b> structures shall be stored (handle) |
| 16 |           | IN n  |
| 17 |           | Number of structures to be allocated (size_t)   |
| 8  | 3.2.26.4  | 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 <pre>pmix_pdata_t</pre> structures (handle)                               |
| 23 |           | IN n  |
| 24 |           | Number of structures in the array (size_t)  |

# 1 3.2.26.5 Load a lookup data structure

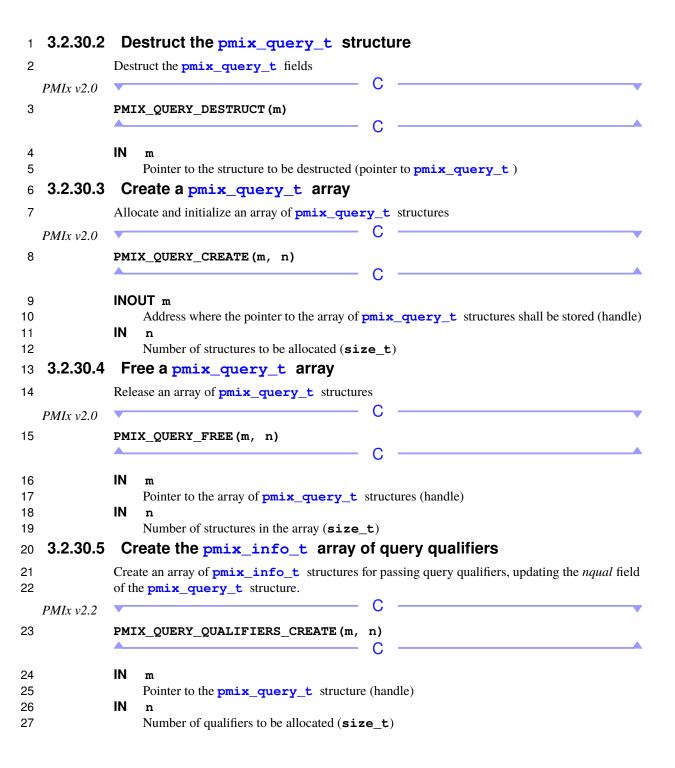
| 2              | Summary  Load key, process identifier, and data value into a pmix_pdata_t structure.   |
|----------------|--|
| PMIx v1.0      | C  |
| 4              | PMIX_PDATA_LOAD(m, p, k, d, t);  |
| 5<br>6<br>7    | IN m  Pointer to the pmix_pdata_t structure into which the key and data are to be loaded (pointer to pmix_pdata_t)   |
| 8<br>9<br>10   | IN p Pointer to the pmix_proc_t structure containing the identifier of the process being referenced (pointer to pmix_proc_t)   |
| 11<br>12<br>13 | IN k String key to be loaded - must be less than or equal to PMIX_MAX_KEYLEN in length (handle)  |
| 14<br>15       | IN d Pointer to the data value to be loaded (handle)   |
| 16<br>17       | <pre>IN t     Type of the provided data value ( pmix_data_type_t )</pre>   |
| 18<br>19       | This macro simplifies the loading of key, process identifier, and data into a pmix_proc_t by correctly assigning values to the structure's fields.   |
|                | Advice to users  |
| 20<br>21<br>22 | Key, process identifier, and data will all be copied into the <pre>pmix_pdata_t</pre> - thus, the source information can be modified or free'd without affecting the copied data once the macro has completed. |

### 3.2.26.6 Transfer a lookup data structure

```
2
             Summary
             Transfer key, process identifier, and data value between two pmix pdata t structures.
 3
   PMIx v2.0
             PMIX PDATA XFER(d, s);
4
5
             IN
                  Pointer to the destination pmix_pdata_t (pointer to pmix_pdata_t)
6
             IN
 7
                  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 —
             Key, process identifier, and data will all be copied into the destination pmix pdata t - thus, the
10
11
             source pmix pdata t may free'd without affecting the copied data once the macro has
12
             completed.
   3.2.27 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
16
             typedef struct pmix_app {
17
                  /** Executable */
18
                  char *cmd;
                  /** Argument set, NULL terminated */
19
20
                  char **argv;
                  /** Environment set, NULL terminated */
21
22
                  char **env;
                  /** Current working directory */
23
24
                  char *cwd;
                  /** Maximum processes with this profile */
25
                  int maxprocs;
26
                  /** Array of info keys describing this application*/
27
                  pmix_info_t *info;
28
                  /** Number of info keys in 'info' array */
29
                  size t ninfo;
30
31
              } pmix app t;
```

```
3.2.28 App structure support macros
              The following macros are provided to support the pmix_app_t structure.
 2
    3.2.28.1 Initialize the pmix_app_t structure
              Initialize the pmix app t fields
 4
   PMIx v1.0
 5
              PMIX_APP_CONSTRUCT (m)
              IN
 6
                   m
 7
                   Pointer to the structure to be initialized (pointer to pmix app t)
    3.2.28.2 Destruct the pmix_app_t structure
              Destruct the pmix_app_t fields
   PMIx v1.0
10
              PMIX_APP_DESTRUCT (m)
11
              IN
                   Pointer to the structure to be destructed (pointer to pmix app t)
12
    3.2.28.3 Create a pmix_app_t array
13
14
              Allocate and initialize an array of pmix_app_t structures
   PMIx v1.0
15
              PMIX APP CREATE (m, n)
              INOUT m
16
                   Address where the pointer to the array of pmix_app_t structures shall be stored (handle)
17
              IN
18
                   Number of structures to be allocated (size t)
19
    3.2.28.4 Free a pmix_app_t array
20
21
              Release an array of pmix_app_t structures
   PMIx v1.0
22
              PMIX APP FREE (m, n)
23
              IN
24
                   Pointer to the array of pmix_app_t structures (handle)
              IN
25
                   Number of structures in the array (size t)
26
```

```
3.2.28.5 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
              PMIX APP INFO CREATE (m, n)
 4
              IN
 5
 6
                  Pointer to the pmix app t structure (handle)
 7
              IN
                  Number of directives to be allocated (size t)
 8
   3.2.29 Query Structure
              The pmix_query_t structure is used by PMIx_Query_info_nb to describe a single query
10
11
              operation.
   PMIx v2.0
12
              typedef struct pmix_query {
13
                   char **keys;
14
                   pmix info t *qualifiers;
                   size_t nqual;
15
              } pmix query t;
16
   3.2.30 Query structure support macros
18
              The following macros are provided to support the pmix_query_t structure.
   3.2.30.1 Initialize the pmix_query_t structure
19
20
              Initialize the pmix query t fields
   PMIx v2.0
21
              PMIX QUERY CONSTRUCT (m)
              IN
22
23
                  Pointer to the structure to be initialized (pointer to pmix query t)
```



# 3.2.31 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;
            } pmix regattr t;;
10
```

Note that in this structure:

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

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

— Advice to PMIx library implementers —

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

11 12

13

14 15

16

17

18

19

20 21

22

23

24

25 26

27 28

29

30

#### Although not strictly required, host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them. 2 3.2.32 Attribute registration structure support macros The following macros are provided to support the **pmix regattr** t structure. 4 3.2.32.1 Initialize the pmix\_regattr\_t structure Initialize the pmix regattr t fields 6 PMIx v4.0 7 PMIX REGATTR CONSTRUCT (m) IN 8 m 9 Pointer to the structure to be initialized (pointer to pmix\_regattr\_t) 3.2.32.2 Destruct the pmix\_regattr\_t structure 11 Destruct the pmix\_regattr\_t fields, releasing all strings. *PMIx v4.0* 12 PMIX REGATTR DESTRUCT (m) IN 13 14 Pointer to the structure to be destructed (pointer to **pmix regattr t**) 3.2.32.3 Create a pmix\_regattr\_t array 15 16 Allocate and initialize an array of **pmix\_regattr\_t** structures PMIx v4.017 PMIX\_REGATTR\_CREATE(m, n) INOUT m 18 Address where the pointer to the array of pmix\_regattr\_t structures shall be stored 19 (handle) 20 IN 21 Number of structures to be allocated (size t) 22

Advice to PMIx server hosts —

```
3.2.32.4 Free a pmix_regattr_t array
2
               Release an array of pmix_regattr_t structures
   PMIx v4.0
 3
               PMIX REGATTR FREE (m, n)
               INOUT m
4
5
                    Pointer to the array of pmix_regattr_t structures (handle)
6
               IN
 7
                    Number of structures in the array (size t)
    3.2.32.5
8
               Load a pmix_regattr_t structure
               Load values into a pmix regattr t structure. The macro can be called multiple times to add
9
               as many strings as desired to the same structure by passing the same address and a NULL key to the
10
               macro. Note that the t type value must be given each time.
11
   PMIx v4.0
12
               PMIX REGATTR LOAD (a, n, k, t, ni, v)
               IN
13
                    Pointer to the structure to be loaded (pointer to pmix_proc_t)
14
               IN
15
16
                    String name of the attribute (string)
               IN
17
                    Key value to be loaded ( pmix_key_t )
18
               IN
19
20
                    Type of data associated with the provided key ( pmix_data_type_t )
21
               IN
                   Number of pmix_info_t elements to be allocated in info(size_t)
22
               IN
23
                    One-line description to be loaded (more can be added separately) (string)
24
    3.2.32.6
                Transfer a pmix_regattr_t to another pmix_regattr_t
25
26
27
               Non-destructively transfer the contents of a pmix regattr t structure to another one.
   PMIx v4.0
               PMIX REGATTR XFER(m, n)
28
               INOUT m
29
                    Pointer to the destination pmix_regattr_t structure (handle)
30
               IN
31
                    Pointer to the source pmix_regattr_t structure (handle)
32
```

```
PMIx Group Directives
   3.2.33
             The pmix_group_opt_t type is an enumerated type used with the PMIx_Group_join API
  PMIx v4.0
3
             to indicate accept or decline of the invitation - these are provided for readability of user code:
             PMIX_GROUP_DECLINE
                                      Decline the invitation
 4
 5
             PMIX_GROUP_ACCEPT
                                     Accept the invitation.
   3.2.34 Byte Object Type
7
             The pmix_byte_object_t structure describes a raw byte sequence.
   PMIx v1.0
             typedef struct pmix_byte_object {
8
9
                  char *bytes;
                  size_t size;
10
11
              } pmix_byte_object_t;
   3.2.35
              Byte object support macros
             The following macros support the pmix_byte_object_t structure.
13
   3.2.35.1
              Initialize the pmix_byte_object_t structure
15
             Initialize the pmix byte object t fields
   PMIx v2.0
16
             PMIX BYTE OBJECT CONSTRUCT (m)
17
             IN
                  Pointer to the structure to be initialized (pointer to pmix_byte_object_t)
18
   3.2.35.2
              Destruct the pmix_byte_object_t structure
19
20
             Clear the pmix byte object t fields
   PMIx v2.0
21
             PMIX BYTE OBJECT DESTRUCT (m)
```

Pointer to the structure to be destructed (pointer to pmix\_byte\_object\_t)

IN

```
3.2.35.3 Create a pmix_byte_object_t structure
2
              Allocate and intitialize an array of pmix_byte_object_t structures
                                                      C
   PMIx v2.0
3
              PMIX_BYTE_OBJECT_CREATE(m, n)
              INOUT m
4
5
                   Address where the pointer to the array of pmix_byte_object_t structures shall be
6
                   stored (handle)
7
              IN
                   n
                   Number of structures to be allocated (size_t)
8
   3.2.35.4 Free a pmix_byte_object_t array
10
              Release an array of pmix_byte_object_t structures
   PMIx v2.0
              PMIX_BYTE_OBJECT_FREE(m, n)
11
              IN
12
                   Pointer to the array of pmix byte object t structures (handle)
13
14
              IN
                   Number of structures in the array (size_t)
15
   3.2.35.5 Load a pmix_byte_object_t structure
16
              Load values into a pmix_byte_object_t
17
   PMIx v2.0
              PMIX BYTE OBJECT LOAD (b, d, s)
18
              IN
19
20
                   Pointer to the structure to be loaded (pointer to pmix_byte_object_t)
              IN
21
                  Pointer to the data to be loaded (char*)
22
              IN
23
                   Number of bytes in the data array (size t)
24
```

# 3.2.36 Data Array Structure

23

```
2
              The pmix_data_array_t structure defines an array data structure.
   PMIx v2.0
 3
              typedef struct pmix_data_array {
 4
                   pmix_data_type_t type;
 5
                   size_t size;
                   void *array;
 6
 7
              } pmix_data_array_t;
    3.2.37
              Data array support macros
              The following macros support the pmix_data_array_t structure.
 9
               Initialize a pmix_data_array_t structure
    3.2.37.1
              Initialize the pmix data array t fields, allocating memory for the array of the indicated type.
11
                                                      C
   PMIx v2.2
              PMIX DATA ARRAY_CONSTRUCT(m, n, t)
12
              IN
13
                  Pointer to the structure to be initialized (pointer to pmix_data_array_t)
14
              IN
15
16
                  Number of elements in the array (size_t)
17
              IN
                  PMIx data type of the array elements ( pmix_data_type_t )
18
   3.2.37.2 Destruct a pmix_data_array_t structure
19
20
              Destruct the pmix data array t, releasing the memory in the array.
   PMIx v2.2
21
              PMIX DATA ARRAY CONSTRUCT (m)
22
              IN
```

Pointer to the structure to be destructed (pointer to pmix\_data\_array\_t)

```
3.2.37.3 Create a pmix_data_array_t structure
               Allocate memory for the pmix_data_array_t object itself, and then allocate memory for the
2
               array of the indicated type.
 3
   PMIx v2.2
               PMIX DATA ARRAY CREATE (m, n, t)
 4
5
               INOUT m
6
                   Variable to be set to the address of the structure (pointer to pmix_data_array_t)
7
               IN
8
                   Number of elements in the array (size_t)
               IN
9
                   PMIx data type of the array elements ( pmix_data_type_t )
10
    3.2.37.4 Free a pmix_data_array_t structure
11
               Release the memory in the array, and then release the pmix data array_t object itself.
12
   PMIx v2.2
13
               PMIX_DATA_ARRAY_FREE (m)
14
               IN
15
                   Pointer to the structure to be released (pointer to pmix data array t)
    3.2.38 Argument Array Macros
17
               The following macros support the construction and release of NULL-terminated argy arrays of
               strings.
18
    3.2.38.1 Argument array extension
19
20
               Summary
               Append a string to a NULL-terminated, argy-style array of strings.
21
22
               PMIX ARGV APPEND(r, a, b);
               OUT r
23
24
                   Status code indicating success or failure of the operation ( pmix_status_t )
25
               INOUT a
                   Argument list (pointer to NULL-terminated array of strings)
26
27
               IN
28
                   Argument to append to the list (string)
```

#### **Description** 1 2 This function helps the caller build the **argy** portion of **pmix app t** structure, arrays of keys for querying, or other places where argy-style string arrays are required in the way that the PRI 3 4 expects it to be constructed. Advice to users 5 The provided argument is copied into the destination array - thus, the source string can be free'd without affecting the array once the macro has completed. 6 3.2.38.2 Argument array extension - unique Summary 8 Append a string to a NULL-terminated, argy-style array of strings, but only if the provided 9 10 argument doesn't already exist somewhere in the array. PMIX ARGV APPEND\_UNIQUE(r, a, b); 11 OUT r 12 Status code indicating success or failure of the operation (pmix\_status\_t) 13 14 INOUT a Argument list (pointer to NULL-terminated array of strings) 15 IN 16 Argument to append to the list (string) 17 **Description** 18 This function helps the caller build the **argv** portion of **pmix\_app\_t** structure, arrays of keys 19 for querying, or other places where argy-style string arrays are required in the way that the PRI 20 expects it to be constructed. 21 — Advice to users — 22 The provided argument is copied into the destination array - thus, the source string can be free'd 23 without affecting the array once the macro has completed.

#### 3.2.38.3 Argument array release 1 2 Summary Free an argy-style array and all of the strings that it contains 3 4 PMIX\_ARGV\_FREE(a); 5 IN Argument list (pointer to NULL-terminated array of strings) 6 7 **Description** 8 This function releases the array and all of the strings it contains. 3.2.38.4 Argument array split 10 Summary Split a string into a NULL-terminated argy array. 11 12 PMIX ARGV SPLIT(a, b, c); 13 OUT a Resulting argy-style array (char\*\*) 14 IN 15 16 String to be split (char\*) IN 17 18 Delimiter character (char) Description 19 Split an input string into a NULL-terminated argy array. Do not include empty strings in the 20 21 resulting array. Advice to users -All strings are inserted into the argy array by value; the newly-allocated array makes no references 22 to the src\_string argument (i.e., it can be freed after calling this function without invalidating the 23 output argy array) 24

```
3.2.38.5 Argument array join
 1
 2
               Summary
               Join all the elements of an argy array into a single newly-allocated string.
 3
               PMIX_ARGV_JOIN(a, b, c);
 4
               OUT a
 5
 6
                    Resulting string (char*)
 7
               IN
                    Argy-style array to be joined (char**)
 8
               IN
 9
10
                    Delimiter character (char)
               Description
11
               Join all the elements of an argy array into a single newly-allocated string.
12
    3.2.38.6 Argument array count
13
               Summary
14
15
               Return the length of a NULL-terminated argy array.
               PMIX_ARGV_COUNT(r, a);
16
               OUT r
17
                    Number of strings in the array (integer)
18
19
               IN
20
                    Argv-style array (char**)
               Description
21
               Count the number of elements in an argy array
22
    3.2.38.7 Argument array copy
23
               Summary
24
25
               Copy an argy array, including copying all off its strings.
               PMIX_ARGV_COPY(a, b);
26
27
               OUT a
28
                    New argy-style array (char**)
29
               IN
                    Argy-style array (char**)
30
```

### Description

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

### 3.2.39 Set Environment Variable

#### Summary

1

4

5

6

7 8

9

10

11 12

13

14

15

16

17 18

19

20

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

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

OUT r

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

IN name

Argument name (string)

IN value

Argument value (string)

INOUT env

Environment array to update (pointer to array of strings)

#### **Description**

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

### Advice to users

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

# 3.3 Generalized Data Types Used for Packing/Unpacking

2

4

5

7

8

9

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

# Advice to PMIx library implementers -

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

```
10
              PMIX_UNDEF
                               Undefined
                             Boolean (converted to/from native true/false) (bool)
11
              PMIX BOOL
12
                             A byte of data (uint8_t)
              PMIX_BYTE
13
              PMIX STRING
                               NULL terminated string (char*)
14
              PMIX SIZE
                             Size size t
15
                            Operating process identifier (PID) (pid t)
              PMIX PID
16
                            Integer (int)
              PMIX INT
17
              PMIX_INT8
                             8-byte integer (int8_t)
18
              PMIX INT16
                               16-byte integer (int16_t)
19
              PMIX INT32
                              32-byte integer (int32_t)
20
                              64-byte integer (int64_t)
              PMIX_INT64
21
                             Unsigned integer (unsigned int)
              PMIX UINT
22
              PMIX UINT8
                               Unsigned 8-byte integer (uint8 t)
23
                                Unsigned 16-byte integer (uint16 t)
              PMIX UINT16
                                Unsigned 32-byte integer (uint32_t)
24
              PMIX_UINT32
25
              PMIX UINT64
                                Unsigned 64-byte integer (uint64_t)
26
              PMIX FLOAT
                              Float (float)
27
                                Double (double)
              PMIX_DOUBLE
28
              PMIX TIMEVAL
                                 Time value (struct timeval)
29
              PMIX TIME
                             Time (time t)
                                Status code pmix status t
30
              PMIX STATUS
31
                               Value ( pmix_value_t )
              PMIX_VALUE
32
              PMIX_PROC
                             Process ( pmix_proc_t )
                            Application context
33
              PMIX APP
34
              PMIX_INFO
                             Info object
35
              PMIX PDATA
                              Pointer to data
                                Buffer
36
              PMIX BUFFER
37
              PMIX_BYTE_OBJECT
                                      Byte object ( pmix_byte_object_t )
38
              PMIX KVAL
                             Key/value pair
```

```
1
                                 Persistance (pmix_persistence_t)
              PMIX_PERSIST
2
                                 Pointer to an object (void*)
              PMIX POINTER
 3
              PMIX SCOPE
                              Scope ( pmix_scope_t )
 4
                                    Range for data ( pmix_data_range_t )
              PMIX DATA RANGE
 5
                                 PMIx command code (used internally)
              PMIX COMMAND
6
              PMIX INFO DIRECTIVES
                                          Directives flag for pmix info t (
7
                  pmix info directives t)
                                   Data type code ( pmix_data_type_t )
8
              PMIX DATA TYPE
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 )
              PMIX QUERY
13
                              Query structure (pmix query t)
14
              PMIX COMPRESSED STRING
                                             String compressed with zlib (char*)
15
              PMIX ALLOC DIRECTIVE
                                          Allocation directive (pmix alloc directive t)
                                     Input/output forwarding channel ( pmix_iof_channel_t )
16
              PMIX_IOF_CHANNEL
17
              PMIX ENVAR
                              Environmental variable structure ( pmix_envar_t )
                              Structure containing fabric coordinates ( pmix_coord_t )
18
              PMIX COORD
19
                                 Structure supporting attribute registrations ( pmix_regattr_t )
              PMIX REGATTR
20
                              Regular expressions - can be a valid NULL-terminated string or an arbitrary
              PMIX REGEX
21
                  array of bytes
```

# 3.4 Reserved attributes

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

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

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

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

```
PMIX ATTR UNDEF NULL (NULL)
```

Constant representing an undefined attribute.

23

24

25 26

27

28

29

30 31

32

33

34

35 36

### 1 3.4.1 Initialization attributes

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

```
2
               These attributes are defined to assist the caller with initialization by passing them into the
               appropriate initialization API - thus, they are not typically accessed via the PMIx Get API.
 3
               PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)
5
                     Pointer to libevent 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.
               PMIX SERVER REMOTE CONNECTIONS "pmix.srvr.remote" (bool)
8
9
                     Allow connections from remote tools. Forces the PMIx server to not exclusively use
10
                     loopback device.
               PMIX SERVER SYSTEM SUPPORT "pmix.srvr.sys" (bool)
11
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
                     PMIx server will place its tool rendezvous point and contact information.
16
               PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
17
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)
                     Enable PMIx internal monitoring by the PMIx server.
21
22
               PMIX SERVER NSPACE "pmix.srv.nspace" (char*)
23
                     Name of the namespace to use for this PMIx server.
               PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
24
                     Rank of this PMIx server
25
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)
   3.4.2 Tool-related attributes
29
               These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server by
30
31
               passing them into the PMIx tool init API - thus, they are not typically accessed via the
               PMIx Get API.
32
33
               PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
                     Name of the namespace to use for this tool.
34
35
               PMIX TOOL RANK "pmix.tool.rank" (uint32 t)
                     Rank of this tool.
36
37
               PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
                     PID of the target PMIx server for a tool.
38
               PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)
39
```

```
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)
                    Maximum number of times to try to connect to PMIx server.
9
              PMIX CONNECT RETRY DELAY "pmix.tool.retry" (uint32 t)
10
                    Time in seconds between connection attempts to a PMIx server.
11
12
              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.
13
14
              PMIX RECONNECT SERVER "pmix.tool.recon" (bool)
15
                    Tool is requesting to change server connections
              PMIX_LAUNCHER "pmix.tool.launcher" (bool)
16
17
                    Tool is a launcher and needs rendezvous files created
   3.4.3 Identification attributes
19
```

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

```
PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

PMIX_DSTPATH "pmix.dstpath" (char*)
```

Path to shared memory data storage (dstore) files.

PMIX\_VERSION\_INFO "pmix.version" (char\*)

PMIx version of contractor.

PMIX\_REQUESTOR\_IS\_TOOL "pmix.req.tool" (bool)

The requesting process is a PMIx tool.

PMIX REQUESTOR IS CLIENT "pmix.req.client" (bool)

The requesting process is a PMIx client.

PMIX PSET\_NAME "pmix.pset.nm" (char\*)

User-assigned name for the process set containing the given process.

20

21

22

23

24 25

26 27

28

29

30 31

32

# 3.4.4 Programming model attributes

25

26

27

28

29

30

31 32

33

34

2 These attributes are associated with programming models. 3 PMIX PROGRAMMING MODEL "pmix.pgm.model" (char\*) Programming model being initialized (e.g., "MPI" or "OpenMP") 4 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.mld.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") PMIX\_MODEL\_NUM\_THREADS "pmix.mdl.nthrds" (uint64\_t) 11 Number of active threads being used by the model 12 PMIX MODEL NUM CPUS "pmix.mdl.ncpu" (uint64 t) 13 14 Number of cpus being used by the model 15 PMIX MODEL CPU TYPE "pmix.mdl.cputype" (char\*) Granularity - "hwthread", "core", etc. 16 PMIX\_MODEL\_PHASE\_NAME "pmix.mdl.phase" (char\*) 17 User-assigned name for a phase in the application execution (e.g., "cfd reduction") 18 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 located on cpu close to master thread), "spread" (threads load-balanced across available cpus) 23 3.4.5 UNIX socket rendezvous socket attributes

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

```
PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)
Disable legacy UNIX socket (usock) support

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)
POSIX mode_t (9 bits valid)

PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)
```

Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

### 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. PMIX TCP\_REPORT\_URI "pmix.tcp.repuri" (char\*) 5 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 include when establishing the TCP connection. 13 PMIX TCP IF EXCLUDE "pmix.tcp.ifexclude" (char\*) 14 15 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. 16 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. PMIX\_TCP\_DISABLE\_IPV4 "pmix.tcp.disipv4" (bool) 21 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. 3.4.7 Global Data Storage (GDS) attributes These attributes are used to define the behavior of the GDS used to manage key/value pairs by 26 passing them into the relevant initialization API - thus, they are not typically accessed via the 27 28 PMIx Get API. 29 PMIX\_GDS\_MODULE "pmix.gds.mod" (char\*) Comma-delimited string of desired modules. 30

# 3.4.8 General process-level attributes

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

```
PMIX_CPUSET "pmix.cpuset" (char*)
   hwloc² bitmap to be applied to the process upon launch.

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

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

32 33

34

35 36

<sup>&</sup>lt;sup>2</sup>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.
   3.4.9
             Scratch directory attributes
5
              These attributes are used to define an application scratch directory and are referenced using the
              PMIX RANK WILDCARD rank.
6
 7
              PMIX TMPDIR "pmix.tmpdir" (char*)
8
                    Full path to the top-level temporary directory assigned to the session.
9
              PMIX NSDIR "pmix.nsdir" (char*)
10
                    Full path to the temporary directory assigned to the namespace, under PMIX_TMPDIR.
11
              PMIX PROCDIR "pmix.pdir" (char*)
                    Full path to the subdirectory under PMIX_NSDIR assigned to the process.
12
13
              PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)
                    Resource Manager will clean session directories
14
   3.4.10 Relative Rank Descriptive Attributes
15
              These attributes are used to describe information about relative ranks as assigned by the RM, and
16
              thus are referenced using the process rank except where noted.
17
18
              PMIX CLUSTER ID "pmix.clid" (char*)
                    A string name for the cluster this proc is executing on
19
              PMIX PROCID "pmix.procid" (pmix proc t)
20
21
                    Process identifier
22
              PMIX_NSPACE "pmix.nspace" (char*)
23
                    Namespace of the job.
24
              PMIX JOBID "pmix.jobid" (char*)
                    Job identifier assigned by the scheduler.
25
26
              PMIX_APPNUM "pmix.appnum" (uint32_t)
27
                    Application number within the job.
28
              PMIX_RANK "pmix.rank" (pmix_rank_t)
                    Process rank within the job.
29
              PMIX GLOBAL_RANK "pmix.grank" (pmix_rank_t)
30
                    Process rank spanning across all jobs in this session.
31
              PMIX_APP_RANK "pmix.apprank" (pmix_rank_t)
32
33
                    Process rank within this application.
34
              PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t)
                    Starting global rank of this job - referenced using PMIX_RANK_WILDCARD.
35
              PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)
36
                    Local rank on this node within this job.
37
              PMIX NODE RANK "pmix.nrank" (uint16 t)
38
39
                    Process rank on this node spanning all jobs.
```

| 1  | <pre>PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)</pre>  |
|----|---|
| 2  | Lowest rank on this node within this job - referenced using PMIX_RANK_WILDCARD.               |
| 3  | <pre>PMIX_APPLDR "pmix.aldr" (pmix_rank_t)</pre>  |
| 4  | Lowest rank in this application within this job - referenced using PMIX_RANK_WILDCARD         |
| 5  | <pre>PMIX_PROC_PID "pmix.ppid" (pid_t)</pre>  |
| 6  | PID of specified process.   |
| 7  | <pre>PMIX_SESSION_ID "pmix.session.id" (uint32_t)</pre>                                       |
| 8  | Session identifier - referenced using PMIX_RANK_WILDCARD.                                     |
| 9  | <pre>PMIX_NODE_LIST "pmix.nlist" (char*)</pre>  |
| 10 | Comma-delimited list of nodes running processes for the specified namespace - referenced      |
| 11 | using PMIX_RANK_WILDCARD.   |
| 12 | <pre>PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)</pre>                                       |
| 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 (e.g., where a specified process is running, or a given device is located).  |
| 17 | PMIX_NODEID "pmix.nodeid" (uint32_t)  |
| 18 | Node identifier expressed as the node's index (beginning at zero) in an array of nodes within |
| 19 | the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME      |
| 20 | of the node - i.e., users can interchangeably reference the same location using either the    |
| 21 | PMIX_HOSTNAME or corresponding PMIX_NODEID .  |
| 22 | PMIX_LOCAL_PEERS "pmix.lpeers" (char*)  |
| 23 | Comma-delimited list of ranks on this node within the specified namespace - referenced        |
| 24 | using PMIX_RANK_WILDCARD.   |
| 25 | <pre>PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)</pre>                                 |
| 26 | Array of <b>pmix_proc_t</b> of all processes on the specified node - referenced using         |
| 27 | PMIX_RANK_WILDCARD  |
| 28 | PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)   |
| 29 | Colon-delimited cpusets of local peers within the specified namespace - referenced using      |
| 30 | PMIX_RANK_WILDCARD.   |
| 31 | <pre>PMIX_PROC_URI "pmix.puri" (char*)</pre>  |
| 32 | URI containing contact information for a given process.                                       |
| 33 | PMIX_LOCALITY "pmix.loc" (uint16_t)   |
| 34 | Relative locality of the specified process to the requestor.                                  |
| 35 | <pre>PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)</pre>   |
| 36 | Process identifier of the parent process of the calling process.                              |
| 37 | PMIX_EXIT_CODE "pmix.exit.code" (int)   |
| 38 | Exit code returned when process terminated  |
|    |   |

### 3.4.11 Information retrieval attributes

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

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

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

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

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX\_JOBID or PMIX\_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

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

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a **PMIX\_APPNUM** attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

#### PMIX NODE INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX\_NODEID** or **PMIX\_HOSTNAME** attribute identifying the desired target.

# 3.4.12 Information storage attributes

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

```
PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)
```

Provide an array of **pmix\_info\_t** containing session-level information. The **PMIX\_SESSION\_ID** attribute is required to be included in the array.

```
PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
```

Provide an array of <code>pmix\_info\_t</code> containing job-level information. The <code>PMIX\_SESSION\_ID</code> attribute of the <code>session</code> containing the <code>job</code> is required to be included in the array whenever the PMIx server library may host multiple sessions (e.g., when executing with a host RM daemon). As information is registered one job (aka namespace) at a time via the <code>PMIx\_server\_register\_nspace</code> API, there is no requirement that the array contain either the <code>PMIX\_NSPACE</code> or <code>PMIX\_JOBID</code> attributes when used in that context (though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is ambiguous.

#### PMIX\_APP\_INFO\_ARRAY "pmix.app.arr" (pmix\_data\_array\_t)

Provide an array of **pmix\_info\_t** containing app-level information. The **PMIX\_NSPACE** or **PMIX\_JOBID** attributes of the **job** containing the application, plus its **PMIX\_APPNUM** attribute, are must to be included in the array when the array is *not* included as part of a call to **PMIx\_server\_register\_nspace** - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

### PMIX\_NODE\_INFO\_ARRAY "pmix.node.arr" (pmix\_data\_array\_t)

Provide an array of <code>pmix\_info\_t</code> containing node-level information. At a minimum, either the <code>PMIX\_NODEID</code> or <code>PMIX\_HOSTNAME</code> attribute is required to be included in the array, though both may be included.

Note that these assemblages can be used hierarchically:

- a PMIX\_JOB\_INFO\_ARRAY might contain multiple PMIX\_APP\_INFO\_ARRAY elements, each describing values for a specific application within the job
- a PMIX\_JOB\_INFO\_ARRAY could contain a PMIX\_NODE\_INFO\_ARRAY for each node hosting processes from that job, each array describing job-level values for that node
- a PMIX\_SESSION\_INFO\_ARRAY might contain multiple PMIX\_JOB\_INFO\_ARRAY
  elements, each describing a job executing within the session. Each job array could, in turn,
  contain both application and node arrays, thus providing a complete picture of the active
  operations within the allocation

# Advice to PMIx library implementers -

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

## 3.4.13 Size information attributes

These attributes are used to describe the size of various dimensions of the PMIx universe - all are referenced using PMIX RANK WILDCARD.

#### PMIX\_UNIV\_SIZE "pmix.univ.size" (uint32\_t)

Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is the equivalent to the combination of **PMIX\_SESSION\_INFO\_ARRAY** with the **PMIX\_MAX\_PROCS** entry in the array - it is included in the Standard for historical reasons.

#### PMIX\_JOB\_SIZE "pmix.job.size" (uint32\_t)

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

```
1
               PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)
2
                    Number of applications in this job.
 3
               PMIX_APP_SIZE "pmix.app.size" (uint32_t)
 4
                    Number of processes in this application.
 5
               PMIX LOCAL SIZE "pmix.local.size" (uint32 t)
6
                    Number of processes in this job or application on this node.
7
               PMIX NODE SIZE "pmix.node.size" (uint32 t)
8
                    Number of processes across all jobs on this node.
               PMIX_MAX_PROCS "pmix.max.size" (uint32_t)
9
10
                    Maximum number of processes that can be executed in this context (session, namespace,
                    application, or node). Typically, this is a constraint imposed by a scheduler or by user
11
                    settings in a hostfile or other resource description.
12
               PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)
13
14
                    Number of slots allocated in this context (session, namespace, application, or node). Note
                    that this attribute is the equivalent to PMIX_MAX_PROCS used in the corresponding
15
                    context - it is included in the Standard for historical reasons.
16
17
               PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)
18
                    Number of nodes in this session, or that are currently executing processes from the
19
                    associated namespace or application.
    3.4.14
               Memory information attributes
20
21
               These attributes are used to describe memory available and used in the system - all are referenced
22
               using PMIX_RANK_WILDCARD.
23
               PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
                    Total available physical memory on this node.
24
               PMIX DAEMON MEMORY "pmix.dmn.mem" (float)
25
26
                    Megabytes of memory currently used by the RM daemon.
27
               PMIX CLIENT AVG MEMORY "pmix.cl.mem.avg" (float)
                    Average Megabytes of memory used by client processes.
28
    3.4.15
               Topology information attributes
30
               These attributes are used to describe topology information in the PMIx universe - all are referenced
               using PMIX RANK WILDCARD except where noted.
31
32
               PMIX_LOCAL_TOPO "pmix.ltopo" (char*)
33
                    eXtensible Markup Language (XML) representation of local node topology.
               PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
34
35
                    Pointer to the PMIx client's internal hwloc topology object.
               PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*)
36
                    XML-based description of topology
37
38
               PMIX TOPOLOGY FILE "pmix.topo.file" (char*)
39
                    Full path to file containing XML topology description
```

| 1  |        | PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)   |
|----|--------|--|
| 2  |        | Topology signature string.   |
| 3  |        | PMIX_LOCALITY_STRING "pmix.locstr" (char*)   |
| 4  |        | String describing a process's bound location - referenced using the process's rank. The string                       |
| 5  |        | is of the form:  |
| 6  |        | NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s   |
| 7  |        | Where the %s is replaced with an integer index or inclusive range for hwloc. NM identifies                           |
| 8  |        | the numa node(s). <b>SK</b> identifies the socket(s). <b>L3</b> identifies the L3 cache(s). <b>L2</b> identifies the |
| 9  |        | L2 cache(s). L1 identifies the L1 cache(s). CR identifies the cores(s). HT identifies the                            |
| 0  |        | hardware thread(s). If your architecture does not have the specified hardware designation                            |
| 11 |        | then it can be omitted from the signature.   |
| 2  |        | For example: NM0: SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39.  |
| 13 |        | This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches                                  |
| 4  |        | 0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39.   |
| 15 |        | PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)  |
| 16 |        | Address of the HWLOC shared memory segment.  |
| 17 |        | PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)  |
| 8  |        | Size of the HWLOC shared memory segment.   |
| 19 |        | PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)   |
| 20 |        | Path to the HWLOC shared memory file.  |
| 21 |        | PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)   |
| 22 |        | XML representation of local topology using HWLOC's v1.x format.  |
| 23 |        | PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)   |
| 24 |        | XML representation of local topology using HWLOC's v2.x format.  |
| 25 |        | PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)  |
| 26 |        | Share the HWLOC topology via shared memory   |
| 27 |        | PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)   |
| 28 |        | Kind of VM "hole" HWLOC should use for shared memory   |
|    |        |  |
| 29 | 3.4.16 | Request-related attributes   |
| 30 |        | These attributes are used to influence the behavior of various PMIx operations - they do not                         |
| 31 |        | represent values accessed using the <b>PMI</b> ×_ <b>Get</b> API.  |
| )  |        |  |
| 32 |        | PMIX_COLLECT_DATA "pmix.collect" (bool)  |
| 33 |        | Collect data and return it at the end of the operation.  |
| 34 |        | PMIX_TIMEOUT "pmix.timeout" (int)  |
| 35 |        | Time in seconds before the specified operation should time out $(0 \text{ indicating infinite})$ in                  |
| 36 |        | error. The timeout parameter can help avoid "hangs" due to programming errors that prevent                           |
| 37 |        | the target process from ever exposing its data.  |
| 88 |        | PMIX_IMMEDIATE "pmix.immediate" (bool)   |
| 39 |        | Specified operation should immediately return an error from the PMIx server if the requested                         |
| 10 |        | data cannot be found - do not request it from the host RM.   |
| 11 |        | PMIX WAIT "pmix.wait" (int)  |

| Caller requests that the PMIx server wait until at least the specified number of values are found (0 indicates all and is the default).  PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)  Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.  PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed_barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIX_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term_status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END_exterm_status" (pmix_status_t)  Status returned by a process as of th | 4  | Collan requests that the DMIv server wait until at least the area feel number of values are |
|--|----|---|
| PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)  Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.  PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.  PMIX_DATA_SCODE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed_barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIX event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIX event payload provided by the host environment upon termination of a process on update rate.  PMIX_PROC_TERM_STATUS "p |    |   |
| 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_REQO_"pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example,  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a pooless.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    | · · · · · · · · · · · · · · · · · · ·   |
| impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.  PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIX_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize does not seecute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the |    |   |
| acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values.  PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATIDS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a pob. Note that generation of the PMIX_PROC_TERM_INTORNET PMIX.PROC_TERM_INTORNET.  Status returned by a process upon its termination. The status wi |    |   |
| check their host environment for supported values.  PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, pmix_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  Status returned by a process us on its termination. The status will be communicated as part of a pMIx event payload provided by the host environment upon termination of a job. Note that generation of the specified process us on its termination. The status will be communicated as part of a PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The sta |    |   |
| PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)   If true, indicates that the requested choice of algorithm is mandatory.   PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)   Notify the parent process upon termination of child job.   PMIX_RANGE "pmix.range" (pmix_data_range_t)   Value for calls to publish/lookup/unpublish or for monitoring event notifications.   PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)   Value for calls to PMIx_Publish.   PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)   Scope of the data to be found in a PMIx_Get call.   PMIX_OPTIONAL "pmix.optional" (bool)   Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.   PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)   Request that any pointers in the returned value point directly to values in the key-value store   PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)   Execute a blocking fence operation before executing the specified operation. For example,   PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.   PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)   Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.    PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)   Status returned by a process as of the last report - may not be the actual current state based on update rate.    PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)   Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host          |    | •   |
| If true, indicates that the requested choice of algorithm is mandatory.  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)  Notify the parent process upon termination of child job.  PMIX_RANCE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx_server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIX_Publish or for monitoring event notifications.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| Notify the parent process upon termination of child job.  PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END_event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    | · · · · · · · · · · · · · · · · · · ·   |
| PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIX_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example,  PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| Value for calls to publish/lookup/unpublish or for monitoring event notifications.  PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t) Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t) Scope of the data to be found in a PMIx_Get call.  PMIX_OPTIONAL "pmix.optional" (bool) Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t) Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t) Status returned by a process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    | • 1 1 1   |
| PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIX_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)  Scope of the data to be found in a PMIx_Get call.  PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)  Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  Status returned by a process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| Value for calls to PMIx_Publish.  PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t) Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool) Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t) Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t) State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| 16 PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t) 17 Scope of the data to be found in a PMIx_Get call. 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 the PMIx server if not found. 20 PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) 21 Request that any pointers in the returned value point directly to values in the key-value store 23 PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) 24 Execute a blocking fence operation before executing the specified operation. For example, 25 PMIX_Finalize does not include an internal barrier operation by default. This attribute 26 would direct PMIx_Finalize to execute a barrier as part of the finalize operation. 27 PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t) 28 Status returned by job upon its termination. The status will be communicated as part of a 29 PMIX event payload provided by the host environment upon termination of a job. Note that 30 generation of the PMIX_EVENT_JOB_END event is optional and host environments may 21 choose to provide it only upon request. 29 PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t) 30 Status feturned by a process as of the last report - may not be the actual current state based 34 on update rate. 35 PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) 36 Status returned by a process upon its termination. The status will be communicated as part 37 of a PMIx event payload provided by the host environment upon termination of a process. 38  |    |   |
| Scope of the data to be found in a PMIx_Get_call.  PMIX_OPTIONAL "pmix.optional" (bool) Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) Execute a blocking fence operation before executing the specified operation. For example, PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| DMIX_OPTIONAL "pmix.optional" (bool) Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) Execute a blocking fence operation before executing the specified operation. For example, PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t) Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t) State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) Execute a blocking fence operation before executing the specified operation. For example, PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    | •   |
| the PMIx server if not found.  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| PMIX_GET_STATIC_VALUES "pmix.get.static" (bool) Request that any pointers in the returned value point directly to values in the key-value store PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool) Execute a blocking fence operation before executing the specified operation. For example, PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| Request that any pointers in the returned value point directly to values in the key-value store  PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example,  PMIX_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a  PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| PMIX_EMBED_BARRIER "pmix.embed.barrier" (bool)  Execute a blocking fence operation before executing the specified operation. For example,  PMIX_Finalize does not include an internal barrier operation by default. This attribute  would direct PMIX_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a  PMIX event payload provided by the host environment upon termination of a job. Note that  generation of the PMIX_EVENT_JOB_END event is optional and host environments may  choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based  on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part  of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| Execute a blocking fence operation before executing the specified operation. For example,  PMIx_Finalize does not include an internal barrier operation by default. This attribute  would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a  PMIx event payload provided by the host environment upon termination of a job. Note that  generation of the PMIX_EVENT_JOB_END event is optional and host environments may  choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based  on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part  of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| PMIx_Finalize does not include an internal barrier operation by default. This attribute would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| would direct PMIx_Finalize to execute a barrier as part of the finalize operation.  PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a  PMIx event payload provided by the host environment upon termination of a job. Note that  generation of the PMIX_EVENT_JOB_END event is optional and host environments may  choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| PMIX_JOB_TERM_STATUS "pmix.job.term.status" (pmix_status_t)  Status returned by job upon its termination. The status will be communicated as part of a  PMIx event payload provided by the host environment upon termination of a job. Note that  generation of the PMIX_EVENT_JOB_END event is optional and host environments may  choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based  on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part  of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    | · ·   |
| Status returned by job upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| PMIx event payload provided by the host environment upon termination of a job. Note that generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t) State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process. Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| generation of the PMIX_EVENT_JOB_END event is optional and host environments may choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    | * * *   |
| choose to provide it only upon request.  PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t)  State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| 32 PMIX_PROC_STATE_STATUS "pmix.proc.state" (pmix_proc_state_t) 33 State of the specified process as of the last report - may not be the actual current state based 34 on update rate. 35 PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) 36 Status returned by a process upon its termination. The status will be communicated as part 37 of a PMIx event payload provided by the host environment upon termination of a process. 38 Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| State of the specified process as of the last report - may not be the actual current state based on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   | 32 |   |
| on update rate.  PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t)  Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  |    |   |
| 35 PMIX_PROC_TERM_STATUS "pmix.proc.term.status" (pmix_status_t) 36 Status returned by a process upon its termination. The status will be communicated as part 37 of a PMIx event payload provided by the host environment upon termination of a process. 38 Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    | 1 1   |
| Status returned by a process upon its termination. The status will be communicated as part of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host  | 35 |   |
| of a PMIx event payload provided by the host environment upon termination of a process.  Note that generation of the PMIX_PROC_TERMINATED event is optional and host   |    |   |
| Note that generation of the <b>PMIX_PROC_TERMINATED</b> event is optional and host   | 37 |   |
|  |    |   |
|  |    |   |
|  |    |   |

# 1 3.4.17 Server-to-PMIx library attributes

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

PMIX\_REGISTER\_NODATA "pmix.reg.nodata" (bool)

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

PMIX\_PROC\_DATA "pmix.pdata" (pmix\_data\_array\_t)

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

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

Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.

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

Regular expression describing processes on each node - see 11.1.3.1 for an explanation of its generation.

PMIX ANL MAP "pmix.anlmap" (char\*)

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

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

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

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

Regular expression describing the result of the process mapping.

### 21 3.4.18 Server-to-Client attributes

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

PMIX PROC BLOB "pmix.pblob" (pmix byte object t)

Packed blob of process data.

PMIX\_MAP\_BLOB "pmix.mblob" (pmix\_byte\_object\_t)

Packed blob of process location.

# 28 3.4.19 Event handler registration and notification attributes

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

PMIX EVENT HDLR NAME "pmix.evname" (char\*)

String name identifying this handler.

PMIX\_EVENT\_HDLR\_FIRST "pmix.evfirst" (bool)

Invoke this event handler before any other handlers.

PMIX\_EVENT\_HDLR\_LAST "pmix.evlast" (bool)

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

PMIX EVENT HDLR FIRST IN CATEGORY "pmix.evfirstcat" (bool)

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

5

6

7

8

9

10

11 12

13 14

15

16

17 18

19

20

22

23

24

25

26 27

29

30 31

32

33

34

35

36 37

```
1
              PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)
2
                    Invoke this event handler after all other handlers in this category have been called.
3
              PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)
4
                    Put this event handler immediately before the one specified in the (char*) value.
5
              PMIX EVENT HDLR AFTER "pmix.evafter" (char*)
6
                    Put this event handler immediately after the one specified in the (char*) value.
7
              PMIX EVENT HDLR PREPEND "pmix.evprepend" (bool)
8
                    Prepend this handler to the precedence list within its category.
              PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool)
9
10
                    Append this handler to the precedence list within its category.
              PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)
11
                    Array of pmix proc t defining range of event notification.
12
              PMIX EVENT AFFECTED PROC "pmix.evproc" (pmix proc t)
13
                    The single process that was affected.
14
              PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)
15
                    Array of pmix_proc_t defining affected processes.
16
17
              PMIX EVENT NON DEFAULT "pmix.evnondef" (bool)
                    Event is not to be delivered to default event handlers.
18
              PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
19
20
                    Object to be returned whenever the registered callback function cbfunc is invoked. The
21
                    object will only be returned to the process that registered it.
22
              PMIX EVENT DO NOT CACHE "pmix.evnocache" (bool)
                    Instruct the PMIx server not to cache the event.
23
24
              PMIX EVENT SILENT TERMINATION "pmix.evsilentterm" (bool)
                    Do not generate an event when this job normally terminates.
25
              PMIX EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)
26
27
                    PMIx server that sourced the event
28
              PMIX EVENT TEXT MESSAGE "pmix.evtext" (char*)
29
                    Text message suitable for output by recipient - e.g., describing the cause of the event
   3.4.20 Fault tolerance attributes
30
31
              Attributes to support fault tolerance behaviors - they are values passed to the event notification API
32
              and therefore are not accessed using the PMIx Get API.
              PMIX EVENT TERMINATE SESSION "pmix.evterm.sess" (bool)
33
                    The RM intends to terminate this session.
34
35
              PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)
36
                    The RM intends to terminate this job.
              PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)
37
                    The RM intends to terminate all processes on this node.
38
39
              PMIX EVENT TERMINATE PROC "pmix.evterm.proc" (bool)
40
                    The RM intends to terminate just this process.
              PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int)
41
```

1 The time in seconds before the RM will execute error response. 2 PMIX EVENT NO TERMINATION "pmix.evnoterm" (bool) Indicates that the handler has satisfactorily handled the event and believes termination of the 3 4 application is not required. 5 PMIX\_EVENT\_WANT\_TERMINATION "pmix.evterm" (bool) 6 Indicates that the handler has determined that the application should be terminated 3.4.21 Spawn attributes 7 Attributes used to describe PMIx Spawn behavior - they are values passed to the PMIx Spawn 8 9 API and therefore are not accessed using the PMIx Get API when used in that context. However, some of the attributes defined in this section can be provided by the host environment for other 10 11 purposes - e.g., the environment might provide the PMIX MAPPER attribute in the job-related information so that an application can use PMIx Get to discover the layout algorithm used for 12 13 determining process locations. Multi-use attributes and their respective access reference rank are denoted below. 14 15 PMIX\_PERSONALITY "pmix.pers" (char\*) 16 Name of personality to use. PMIX HOST "pmix.host" (char\*) 17 Comma-delimited list of hosts to use for spawned processes. 18 19 PMIX HOSTFILE "pmix.hostfile" (char\*) 20 Hostfile to use for spawned processes. PMIX\_ADD\_HOST "pmix.addhost" (char\*) 21 Comma-delimited list of hosts to add to the allocation. 22 23 PMIX ADD HOSTFILE "pmix.addhostfile" (char\*) Hostfile listing hosts to add to existing allocation. 24 25 PMIX PREFIX "pmix.prefix" (char\*) 26 Prefix to use for starting spawned processes. 27 PMIX WDIR "pmix.wdir" (char\*) 28 Working directory for spawned processes. PMIX\_MAPPER "pmix.mapper" (char\*) 29 30 Mapping mechanism to use for placing spawned processes - when accessed using PMIx\_Get, use the PMIX\_RANK\_WILDCARD value for the rank to discover the mapping 31 mechanism used for the provided namespace. 32 33 PMIX\_DISPLAY\_MAP "pmix.dispmap" (bool) 34 Display process mapping upon spawn. PMIX PPR "pmix.ppr" (char\*) 35 Number of processes to spawn on each identified resource. 36 37 PMIX MAPBY "pmix.mapby" (char\*) Process mapping policy - when accessed using PMIx\_Get, use the 38 PMIX\_RANK\_WILDCARD value for the rank to discover the mapping policy used for the 39 40 provided namespace PMIX RANKBY "pmix.rankby" (char\*) 41

| 1  | Process ranking policy - when accessed using <b>PMIx_Get</b> , use the                        |
|----|---|
| 2  | PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the          |
| 3  | provided namespace  |
| 4  | <pre>PMIX_BINDTO "pmix.bindto" (char*)</pre>  |
| 5  | Process binding policy - when accessed using PMIx_Get, use the                                |
| 6  | PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the             |
| 7  | provided namespace  |
| 8  | PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)   |
| 9  | Preload binaries onto nodes.  |
| 0  | <pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)</pre>                                     |
| 1  | Comma-delimited list of files to pre-position on nodes.                                       |
| 2  | PMIX_NON_PMI "pmix.nonpmi" (bool)   |
| 3  | Spawned processes will not call PMIx_Init.  |
| 4  | PMIX_STDIN_TGT "pmix.stdin" (uint32_t)  |
| 5  | Spawned process rank that is to receive <b>stdin</b> .  |
| 6  | PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)  |
| 7  | Forward this process's <b>stdin</b> to the designated process.                                |
| 8  | PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)  |
| 9  | Forward <b>stdout</b> from spawned processes to this process.                                 |
| 0  | PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)  |
| :1 | Forward <b>stderr</b> from spawned processes to this process.                                 |
| 2  | PMIX_FWD_STDDIAG "pmix.fwd.stddiag" (bool)  |
| 3  | If a diagnostic channel exists, forward any output on it from the spawned processes to this   |
| 4  | process (typically used by a tool)  |
| 5  | PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)  |
| 6  | Spawned application consists of debugger daemons.   |
| 7  | PMIX_COSPAWN_APP "pmix.cospawn" (bool)  |
| 8  | Designated application is to be spawned as a disconnected job. Meaning that it is not part of |
| 9  | the "comm_world" of the parent process.   |
| 0  | PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)   |
| 1  | Set the application's current working directory to the session working directory assigned by  |
| 2  | the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for                  |
| 3  | the rank to discover the session working directory assigned to the provided namespace         |
| 4  | PMIX_TAG_OUTPUT "pmix.tagout" (bool)  |
| 5  | Tag application output with the identity of the source process.                               |
| 6  | PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)   |
| 7  | Timestamp output from applications.   |
| 8  | PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)  |
| 9  | Merge stdout and stderr streams from application processes.                                   |
| 0  | PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)  |
| 1  | Output application output to the specified file.  |
| 2  | PMIX_INDEX_ARGV "pmix.indxargv" (bool)  |
| 3  | Mark the <b>argv</b> with the rank of the process.  |
|    |   |

```
1
              PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
2
                    Number of cpus to assign to each rank - when accessed using PMIx Get, use the
3
                    PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the
4
                    provided namespace
5
              PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
6
                    Do not place processes on the head node.
7
              PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
8
                    Do not oversubscribe the cpus.
9
              PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
                    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
14
                    namespace
              PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
15
                    Application supports recoverable operations.
16
              PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
17
18
                    Application is continuous, all failed processes should be immediately restarted.
              PMIX MAX RESTARTS "pmix.maxrestarts" (uint32_t)
19
20
                    Maximum number of times to restart a job - when accessed using PMIx Get, use the
21
                    PMIX RANK WILDCARD value for the rank to discover the max restarts for the provided
22
                    namespace
23
              PMIX SPAWN TOOL "pmix.spwn.tool" (bool)
                    Indicate that the job being spawned is a tool
24
   3.4.22 Query attributes
25
26
              Attributes used to describe PMIx Query info nb behavior - these are values passed to the
              PMIx_Query_info_nb API and therefore are not passed to the PMIx_Get API.
27
28
              PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)
29
                    Retrieve updated information from server.
30
              PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*)
                    Request a comma-delimited list of active namespaces.
31
32
              PMIX QUERY JOB STATUS "pmix.gry.jst" (pmix status t)
                    Status of a specified, currently executing job.
33
              PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*)
34
                    Request a comma-delimited list of scheduler queues.
35
36
              PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD)
                    Status of a specified scheduler queue.
37
38
              PMIX QUERY PROC TABLE "pmix.gry.ptable" (char*)
                    Input namespace of the job whose information is being requested returns (
39
40
                    pmix data array t) an array of pmix proc info t.
41
              PMIX QUERY LOCAL PROC TABLE "pmix.gry.lptable" (char*)
```

```
Input namespace of the job whose information is being requested returns (
1
2
                    pmix data array t) an array of pmix proc info t for processes in job on same
 3
                    node.
 4
              PMIX QUERY AUTHORIZATIONS "pmix.qry.auths" (bool)
                    Return operations the PMIx tool is authorized to perform.
 5
              PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
6
7
                    Return a comma-delimited list of supported spawn attributes.
              PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
8
9
                    Return a comma-delimited list of supported debug attributes.
              PMIX QUERY MEMORY USAGE "pmix.gry.mem" (bool)
10
                    Return information on memory usage for the processes indicated in the qualifiers.
11
              PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
12
                    Constrain the query to local information only.
13
              PMIX QUERY REPORT AVG "pmix.gry.avg" (bool)
14
                    Report only average values for sampled information.
15
              PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
16
17
                    Report minimum and maximum values.
              PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
18
                    String identifier of the allocation whose status is being requested.
19
              PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
20
                    Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
21
22
              PMIX QUERY ATTRIBUTE SUPPORT "pmix.gry.attrs" (bool)
                    Query list of supported attributes for specified APIs
23
              PMIX QUERY NUM PSETS "pmix.qry.psetnum" (size t)
24
25
                    Return the number of psets defined in the specified range (defaults to session).
26
              PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (char*)
                    Return a comma-delimited list of the names of the psets defined in the specified range
27
28
                    (defaults to session).
   3.4.23 Log attributes
              Attributes used to describe PMIx_Log_nb behavior - these are values passed to the
30
              PMIx_Log_nb API and therefore are not accessed using the PMIx_Get API.
31
32
              PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
33
                    ID of source of the log request
              PMIX_LOG_STDERR "pmix.log.stderr" (char*)
34
                    Log string to stderr.
35
36
              PMIX LOG STDOUT "pmix.log.stdout" (char*)
                    Log string to stdout.
37
              PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
38
                    Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
39
40
                    otherwise to local syslog
41
              PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)
```

| 1  | Log data to local syslog. Defaults to <b>ERROR</b> priority.                                    |
|----|---|
| 2  | PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)  |
| 3  | Forward data to system "gateway" and log msg to that syslog Defaults to <b>ERROR</b> priority.  |
| 4  | <pre>PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)</pre>  |
| 5  | Syslog priority level   |
| 6  | PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)  |
| 7  | Timestamp for log report  |
| 8  | PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)  |
| 9  | Generate timestamp for log  |
| 0  | PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)   |
| 1  | Label the output stream with the channel name (e.g., "stdout")                                  |
| 2  | PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)   |
| 3  | Print timestamp in output string  |
| 4  | PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)   |
| 5  | Print the output stream in XML format   |
| 6  | PMIX_LOG_ONCE "pmix.log.once" (bool)  |
| 7  | Only log this once with whichever channel can first support it, taking the channels in priority |
| 8  | order   |
| 9  | <pre>PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)</pre>                                     |
| 20 | Message blob to be sent somewhere.  |
| 21 | <pre>PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)</pre>                                  |
| 22 | Log via email based on <pre>pmix_info_t</pre> containing directives.                            |
| 23 | <pre>PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)</pre>  |
| 24 | Comma-delimited list of email addresses that are to receive the message.                        |
| 25 | <pre>PMIX_LOG_EMAIL_SENDER_ADDR "pmix.log.emfaddr" (char*)</pre>                                |
| 26 | Return email address of sender  |
| 27 | <pre>PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)</pre>                                      |
| 28 | Subject line for email.   |
| 29 | <pre>PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)</pre>  |
| 80 | Message to be included in email.  |
| 31 | <pre>PMIX_LOG_EMAIL_SERVER "pmix.log.esrvr" (char*)</pre>                                       |
| 32 | Hostname (or IP address) of estmp server  |
| 3  | <pre>PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (int32_t)</pre>                               |
| 34 | Port the email server is listening to   |
| 5  | PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)  |
| 86 | Store the log data in a global data store (e.g., database)                                      |
| 37 | <pre>PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)</pre>   |
| 88 | Log the provided information to the host environment's job record                               |

# 1 3.4.24 Debugger attributes

```
2
               Attributes used to assist debuggers - these are values that can be passed to the PMIx_Spawn or
 3
               PMIx Init APIs. Some may be accessed using the PMIx Get API with the
 4
               PMIX RANK WILDCARD rank.
5
               PMIX_DEBUG_STOP_ON_EXEC "pmix.dbg.exec" (bool)
6
                     Passed to PMIx Spawn to indicate that the specified application is being spawned under
7
                     debugger, and that the launcher is to pause the resulting application processes on first
8
                     instruction for debugger attach.
9
               PMIX_DEBUG_STOP_IN_INIT "pmix.dbg.init" (bool)
                     Passed to PMIx_Spawn to indicate that the specified application is being spawned under
10
11
                     debugger, and that the PMIx client library is to pause the resulting application processes
12
                     during PMIx_Init until debugger attach and release.
               PMIX_DEBUG_WAIT_FOR_NOTIFY "pmix.dbg.notify" (bool)
13
                     Passed to PMIx_Spawn to indicate that the specified application is being spawned under
14
15
                     debugger, and that the resulting application processes are to pause at some
16
                     application-determined location until debugger attach and release.
               PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
17
                     Namespace of the job to be debugged - provided to the debugger upon launch.
18
               PMIX DEBUG WAITING FOR_NOTIFY "pmix.dbg.waiting" (bool)
19
20
                     Job to be debugged is waiting for a release - this is not a value accessed using the
21
                     PMIx Get API.
22
               PMIX_DEBUG_JOB_DIRECTIVES "pmix.dbg.jdirs" (pmix_data_array_t*)
23
                     Array of job-level directives
24
               PMIX DEBUG APP DIRECTIVES "pmix.dbg.adirs" (pmix data array t*)
25
                     Array of app-level directives
```

# 3.4.25 Resource manager attributes

27

28

29

30

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

```
31 PMIX_RM_NAME "pmix.rm.name" (char*)
32 String name of the RM.
33 PMIX_RM_VERSION "pmix.rm.version" (char*)
34 RM version string.
```

### 1 3.4.26 Environment variable attributes

```
2
              Attributes used to adjust environment variables - these are values passed to the PMIx_Spawn API
 3
              and are not accessed using the PMIx Get API.
 4
              PMIX_SET_ENVAR "pmix.envar.set" (pmix_envar_t*)
5
                    Set the envar to the given value, overwriting any pre-existing one
6
              PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)
7
                    Unset the environment variable specified in the string.
8
              PMIX ADD ENVAR "pmix.envar.add" (pmix envar t*)
9
                    Add the environment variable, but do not overwrite any pre-existing one
10
              PMIX PREPEND ENVAR "pmix.envar.prepnd" (pmix envar t*)
                    Prepend the given value to the specified environmental value using the given separator
11
12
                    character, creating the variable if it doesn't already exist
13
              PMIX_APPEND_ENVAR "pmix.envar.appnd" (pmix_envar_t*)
14
                    Append the given value to the specified environmental value using the given separator
15
                    character, creating the variable if it doesn't already exist
   3.4.27 Job Allocation attributes
16
              Attributes used to describe the job allocation - these are values passed to and/or returned by the
17
18
              PMIx_Allocation_request_nb and PMIx_Allocation_request APIs and are not
19
              accessed using the PMIx_Get API
20
              PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
21
                    User-provided string identifier for this allocation request which can later be used to query
22
                    status of the request.
23
              PMIX ALLOC ID "pmix.alloc.id" (char*)
                    A string identifier (provided by the host environment) for the resulting allocation which can
24
25
                    later be used to reference the allocated resources in, for example, a call to PMIx Spawn.
26
              PMIX ALLOC NUM NODES "pmix.alloc.nnodes" (uint64 t)
27
                    The number of nodes.
              PMIX ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
28
29
                    Regular expression of the specific nodes.
30
              PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
31
                    Number of cpus.
              PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
32
                    Regular expression of the number of cpus for each node.
33
34
              PMIX ALLOC CPU LIST "pmix.alloc.cpulist" (char*)
35
                    Regular expression of the specific cpus indicating the cpus involved.
              PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
36
37
                    Number of Megabytes.
              PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)
38
39
                    Changed to PMIX ALLOC FABRIC
              PMIX ALLOC FABRIC "pmix.alloc.net" (array)
40
```

```
Array of pmix_info_t describing requested fabric resources. This must include at least:
1
                    PMIX ALLOC FABRIC ID, PMIX ALLOC FABRIC TYPE, and
2
                    PMIX ALLOC FABRIC ENDPTS, plus whatever other descriptors are desired.
3
4
              PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
                    Changed to PMIX_ALLOC_FABRIC_ID
5
              PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
6
                    The key to be used when accessing this requested fabric allocation. The allocation will be
7
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
                    type of the included value depends upon the fabric support. For example, a TCP allocation
10
                    might consist of a comma-delimited string of socket ranges such as
11
                    "32000-32100,33005,38123-38146". Additional entries will consist of any provided
12
                    resource request directives, along with their assigned values. Examples include:
13
                    PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
14
                    PMIX ALLOC FABRIC PLANE - if applicable, what plane the resources were assigned
15
                    from; PMIX ALLOC FABRIC QOS - the assigned QoS; PMIX ALLOC BANDWIDTH -
16
                    the allocated bandwidth; PMIX ALLOC FABRIC SEC KEY - a security key for the
17
18
                    requested fabric allocation. NOTE: the assigned values may differ from those requested,
                    especially if PMIX INFO REQD was not set in the request.
19
20
              PMIX ALLOC BANDWIDTH "pmix.alloc.bw" (float)
21
                    Mbits/sec.
22
              PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
                    Changed to PMIX_ALLOC_FABRIC_QOS
23
24
              PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netgos" (char*)
25
                    Ouality of service level.
              PMIX ALLOC TIME "pmix.alloc.time" (uint32 t)
26
                    Time in seconds.
27
28
              PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)
                    Changed to PMIX_ALLOC_FABRIC_TYPE
29
              PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
30
                   Type of desired transport (e.g., "tcp", "udp")
31
              PMIX ALLOC NETWORK PLANE "pmix.alloc.netplane" (char*)
32
33
                    Changed to PMIX ALLOC FABRIC PLANE
              PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
34
                    ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)
35
              PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)
36
37
                    Changed to PMIX_ALLOC_FABRIC_ENDPTS
              PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
38
39
                    Number of endpoints to allocate per process
              PMIX ALLOC NETWORK ENDPTS NODE "pmix.alloc.endpts.nd" (size t)
40
                    Changed to PMIX ALLOC FABRIC ENDPTS NODE
41
42
              PMIX ALLOC FABRIC 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
                    Changed to PMIX ALLOC FABRIC SEC KEY
3
              PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
 4
                    Fabric security key
    3.4.28 Job control attributes
              Attributes used to request control operations on an executing application - these are values passed
6
7
              to the PMIx Job control nb API and are not accessed using the PMIx Get API.
              PMIX JOB CTRL ID "pmix.jctrl.id" (char*)
8
9
                    Provide a string identifier for this request. The user can provide an identifier for the
10
                    requested operation, thus allowing them to later request status of the operation or to
11
                    terminate it. The host, therefore, shall track it with the request for future reference.
              PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
12
13
                    Pause the specified processes.
14
              PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
                    Resume ("un-pause") the specified processes.
15
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
16
                    Cancel the specified request - the provided request ID must match the
17
18
                    PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
19
                    NULL implies cancel all requests from this requestor.
              PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
20
                    Forcibly terminate the specified processes and cleanup.
21
22
              PMIX JOB CTRL RESTART "pmix.jctrl.restart" (char*)
23
                    Restart the specified processes using the given checkpoint ID.
              PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
24
25
                    Checkpoint the specified processes and assign the given ID to it.
              PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
26
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.ckptsiq" (int)
                    Time in seconds to wait for a checkpoint to complete.
31
32
              PMIX_JOB_CTRL_CHECKPOINT_METHOD
33
              "pmix.jctrl.ckmethod" (pmix data array t)
                    Array of pmix_info_t declaring each method and value supported by this application.
34
              PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
35
36
                    Send given signal to specified processes.
37
              PMIX JOB CTRL PROVISION "pmix.jctrl.pvn" (char*)
                    Regular expression identifying nodes that are to be provisioned.
38
              PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
39
40
                    Name of the image that is to be provisioned.
41
              PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
```

| 1  |        | Indicate that the job can be pre-empted.  |
|----|--------|---|
| 2  |        | PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)  |
| 3  |        | Politely terminate the specified processes.   |
| 4  |        | <pre>PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)</pre>                                       |
| 5  |        | Comma-delimited list of files to be removed upon process termination                              |
| 6  |        | <pre>PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)</pre>                                |
| 7  |        | Comma-delimited list of directories to be removed upon process termination                        |
| 8  |        | PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)  |
| 9  |        | Recursively cleanup all subdirectories under the specified one(s)                                 |
| 0  |        | PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)  |
| 11 |        | Only remove empty subdirectories  |
| 12 |        | <pre>PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)</pre>  |
| 13 |        | Comma-delimited list of filenames that are not to be removed                                      |
| 4  |        | PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)   |
| 15 |        | When recursively cleaning subdirectories, do not remove the top-level directory (the one          |
| 16 |        | given in the cleanup request)   |
| 7  | 3.4.29 | Monitoring attributes   |
| 18 |        | Attributes used to control monitoring of an executing application- these are values passed to the |
| 19 |        | PMIx_Process_monitor_nb API and are not accessed using the PMIx_Get API.                          |
| 20 |        | PMIX_MONITOR_ID "pmix.monitor.id" (char*)   |
| 21 |        | Provide a string identifier for this request.   |
| 22 |        | PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)   |
| 23 |        | Identifier to be canceled ( <b>NULL</b> means cancel all monitoring for this process).            |
| 24 |        | PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)  |
| 25 |        | The application desires to control the response to a monitoring event.                            |
| 26 |        | PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)  |
| 27 |        | Register to have the PMIx server monitor the requestor for heartbeats.                            |
| 28 |        | PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)  |
| 29 |        | Send heartbeat to local PMIx server.  |
| 30 |        | PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)                                       |
| 31 |        | Time in seconds before declaring heartbeat missed.  |
| 32 |        | PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)                                      |
| 33 |        | Number of heartbeats that can be missed before generating the event.                              |
| 34 |        | <pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)</pre>  |
| 35 |        | Register to monitor file for signs of life.   |
| 36 |        | <pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)</pre>                                     |
| 37 |        | Monitor size of given file is growing to determine if the application is running.                 |
| 38 |        | <pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)</pre>                                |
| 39 |        | Monitor time since last access of given file to determine if the application is running.          |
| 10 |        | <pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)</pre>                                   |
| 11 |        | Monitor time since last modified of given file to determine if the application is running.        |

```
1
               PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
2
                     Time in seconds between checking the file.
               PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
 3
 4
                     Number of file checks that can be missed before generating the event.
    3.4.30 Security attributes
               Attributes for managing security credentials
6 PMIx v3.0
               PMIX_CRED_TYPE "pmix.sec.ctype" (char*)
7
8
                     When passed in PMIx Get credential, a prioritized, comma-delimited list of desired
9
                     credential types for use in environments where multiple authentication mechanisms may be
10
                     available. When returned in a callback function, a string identifier of the credential type.
11
               PMIX_CRYPTO_KEY "pmix.sec.key" (pmix_byte_object_t)
12
                     Blob containing crypto key
13 3.4.31
               IO Forwarding attributes
               Attributes used to control IO forwarding behavior
14 PMIx v3.0
15
               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
16
17
                     server is allowed (but not required) to drop all bytes received beyond the max size.
18
               PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)
19
                     In an overflow situation, drop the oldest bytes to make room in the cache.
               PMIX IOF DROP NEWEST "pmix.iof.new" (bool)
20
21
                     In an overflow situation, drop any new bytes received until room becomes available in the
22
                     cache (default).
               PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)
23
24
                     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
25
                     becomes available. Any remaining buffered data will be "flushed" upon call to deregister the
26
27
                     respective channel.
28
               PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)
29
                     Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
                     size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
30
               PMIX_IOF_COMPLETE "pmix.iof.cmp" (bool)
31
                     Indicates whether or not the specified IO channel has been closed by the source.
32
33
               PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool)
34
                     Tag output with the channel it comes from.
               PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts" (bool)
35
36
                     Timestamp output
               PMIX_IOF_XML_OUTPUT "pmix.iof.xml" (bool)
37
38
                     Format output in XML
```

## 1 3.4.32 Application setup attributes

37

```
2 PMIx v3.0
              Attributes for controlling contents of application setup data
              PMIX SETUP APP ENVARS "pmix.setup.env" (bool)
3
                    Harvest and include relevant environmental variables
 4
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
   3.4.33 Attribute support level attributes
10
              PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (bool)
                    Request a list of functions supported by the PMIx client library
11
              PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
12
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
              PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
16
17
                    Request attributes supported by the PMIx server library
18
              PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (bool)
                    Request a list of functions supported by the host environment
19
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
              PMIX TOOL ATTRIBUTES "pmix.setup.env" (bool)
24
25
                    Request attributes supported by the PMIx tool library functions
    3.4.34 Descriptive attributes
              PMIX_MAX_VALUE "pmix.descr.maxval" (varies)
27
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)
                    Used in pmix regattr t to describe the minimum valid value for the associated
31
32
                    attribute.
              PMIX ENUM VALUE "pmix.descr.enum" (char*)
33
                    Used in pmix_regattr_t to describe accepted values for the associated attribute.
34
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
```

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*)
                     User-provided group identifier
 4
 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)
                     Participation is optional - do not return an error if any of the specified processes terminate
 8
 9
                     without having joined. The default is false
10
               PMIX GROUP NOTIFY TERMINATION "pmix.grp.notterm" (bool)
                     Notify remaining members when another member terminates without first leaving the group.
11
12
                     The default is false
13
               PMIX_GROUP_INVITE_DECLINE "pmix.grp.decline" (bool)
                     Notify the inviting process that this process does not wish to participate in the proposed
14
15
                     group The default is true
               PMIX_GROUP_MEMBERSHIP "pmix.grp.mbrs" (pmix_data_array_t*)
16
17
                     Array of group member ID's
18
               PMIX GROUP ASSIGN CONTEXT ID "pmix.grp.actxid" (bool)
19
                     Requests that the RM assign a new context identifier to the newly created group. The
20
                     identifier is an unsigned, size t value that the RM guarantees to be unique across the range
                     specified in the request. Thus, the value serves as a means of identifying the group within
21
22
                     that range. If no range is specified, then the request defaults to PMIX RANGE SESSION.
23
               PMIX GROUP CONTEXT ID "pmix.grp.ctxid" (size t)
                     Context identifier assigned to the group by the host RM.
24
               PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)
25
                     Group operation only involves local processes. PMIx implementations are required to
26
27
                     automatically scan an array of group members for local vs remote processes - if only local
                     processes are detected, the implementation need not execute a global collective for the
28
29
                     operation unless a context ID has been requested from the host environment. This can result
                     in significant time savings. This attribute can be used to optimize the operation by indicating
30
                     whether or not only local processes are represented, thus allowing the implementation to
31
32
                     bypass the scan. The default is false
               PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)
33
                     Data collected to be shared during group construction
```

### 3.5 Callback Functions

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

34

36

37

### 3.5.1 Release Callback Function

```
2
              Summary
 3
              The pmix_release_cbfunc_t is used by the pmix_modex_cbfunc_t and
              pmix info cbfunc t operations to indicate that the callback data may be reclaimed/freed by
 4
              the caller.
 5
              Format
6
   PMIx v1.0
              typedef void (*pmix_release_cbfunc_t)
 7
8
                   (void *cbdata)
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
              that we are done with the data so it can be released.
13
   3.5.2
             Modex Callback Function
              Summary
15
16
              The pmix modex obfunc t is used by the pmix server fencenb fn t and
17
              pmix_server_dmodex_req_fn_t PMIx server operations to return modex business card
18
              exchange (BCX) data.
   PMIx v1.0
19
              typedef void (*pmix_modex_cbfunc_t)
20
                    (pmix_status_t status,
                    const char *data, size_t ndata,
21
22
                    void *cbdata,
23
                    pmix_release_cbfunc_t release_fn,
                    void *release cbdata)
24
              IN
25
                   status
26
                   Status associated with the operation (handle)
27
              IN
                   Data to be passed (pointer)
28
29
              IN ndata
                   size of the data (size t)
30
              IN cbdata
31
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)

### Description

5

6 7

8

10

11

12

16

17

18

19 20

21

22

23

24 25

26

27

28

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

### 3.5.3 Spawn Callback Function

#### Summary

The pmix\_spawn\_cbfunc\_t is used on the PMIx client side by PMIx\_Spawn\_nb and on the PMIx server side by pmix\_server\_spawn\_fn\_t.

```
PMIx v1.0

typedef void (*pmix_spawn_cbfunc_t)

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

#### IN status

Status associated with the operation (handle)

#### IN nspace

Namespace string ( pmix\_nspace\_t )

#### IN cbdata

Callback data passed to original API call (memory reference)

#### **Description**

The callback will be executed upon launch of the specified applications in **PMIx\_Spawn\_nb**, or upon failure to launch any of them.

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

### 3.5.4 Op Callback Function

```
Summarv
 2
 3
              The pmix_op_cbfunc_t is used by operations that simply return a status.
   PMIx v1.0
 4
             typedef void (*pmix op cbfunc t)
                   (pmix_status_t status, void *cbdata);
 5
 6
              IN
                 status
 7
                  Status associated with the operation (handle)
              IN
 8
                 cbdata
 9
                  Callback data passed to original API call (memory reference)
              Description
10
11
              Used by a wide range of PMIx API's including PMIx_Fence_nb,
              pmix_server_client_connected_fn_t,PMIx_server_register_nspace. This
12
              callback function is used to return a status to an often nonblocking operation.
13
   3.5.5
            Lookup Callback Function
              Summary
15
              The pmix_lookup_cbfunc_t is used by PMIx_Lookup_nb to return data.
16
                      _____ C ____
   PMIx v1.0
              typedef void (*pmix_lookup_cbfunc_t)
17
                   (pmix_status_t status,
18
                   pmix_pdata_t data[], size_t ndata,
19
20
                   void *cbdata);
              IN
21
                  status
22
                  Status associated with the operation (handle)
23
              IN
                  Array of data returned ( pmix_pdata_t )
24
              IN
25
                  Number of elements in the data array (size t)
26
             IN cbdata
27
28
                  Callback data passed to original API call (memory reference)
```

#### Description

1

3

4

5

6

7

9

14

15

16

17

18 19

21

22

23

25

26

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

11 typedef void (*pmix_value_cbfunc_t)
12 (pmix_status_t status,
13 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)

### 20 **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.

### 4 3.5.7 Info Callback Function

### Summary

The pmix info\_cbfunc\_t is a general information callback used by various APIs.

|                      |             | •   |
|----------------------|-------------|---|
| 1                    |             | IN status   |
| 2                    |             | Status associated with the operation ( pmix_status_t )  |
| 3                    |             | IN info   |
| 4<br>5               |             | Array of pmix_info_t returned by the operation (pointer)  IN ninfo  |
| 6                    |             | Number of elements in the <i>info</i> array (size_t)  |
| 7                    |             | IN cbdata   |
| 8                    |             | Callback data passed to original API call (memory reference)  |
| 9<br>10              |             | IN release_fn Function to be called when done with the <i>info</i> data (function pointer)  |
| 11                   |             | IN release cbdata   |
| 12                   |             | Callback data to be passed to <i>release_fn</i> (memory reference)  |
| 13                   |             | Description   |
| 14                   |             | The <i>status</i> indicates if requested data was found or not. An array of <b>pmix_info_t</b> will contain   |
| 15                   |             | the key/value pairs.  |
|                      |             |   |
| 16                   | 3.5.8       | <b>Event Handler Registration Callback Function</b>   |
| 17                   |             | The pmix_evhdlr_reg_cbfunc_t callback function.   |
|                      |             | Advice to users   |
| 18<br>19<br>20<br>21 |             | The PMIx <i>ad hoc</i> v1.0 Standard defined an error handler registration callback function with a compatible signature, but with a different type definition function name (pmix_errhandler_reg_cbfunc_t). It was removed from the v2.0 Standard and is not included in this document to avoid confusion. |
|                      | PMIx v2.0   | C   |
| 22                   | 1 W11x V2.0 | 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<br>29             |             | IN evhdlr_ref  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)  |

### **Description**

1

2

Define a callback function for calls to PMIx\_Register\_event\_handler

#### 3.5.9 **Notification Handler Completion Callback Function**

#### Summary 4 5 The pmix event notification cbfunc fn t is called by event handlers to indicate completion of their operations. PMIx v2.0 7 typedef void (\*pmix event notification cbfunc fn t) (pmix status t status, 8 pmix\_info\_t \*results, size\_t nresults, 9 pmix\_op\_cbfunc\_t cbfunc, void \*thiscbdata, 10 void \*notification\_cbdata); 11 IN 12 status Status returned by the event handler's operation (pmix status t) 13 14 IN results Results from this event handler's operation on the event (pmix info t) 15 16 IN nresults 17 Number of elements in the results array (size t) IN cbfunc 18 pmix\_op\_cbfunc\_t function to be executed when PMIx completes processing the 19 callback (function reference) 20 thischdata 21 IN 22 Callback data that was passed in to the handler (memory reference) IN 23 cbdata Callback data to be returned when PMIx executes cbfunc (memory reference) 24 25 **Description** Define a callback by which an event handler can notify the PMIx library that it has completed its 26 response to the notification. The handler is required to execute this callback so the library can 27 determine if additional handlers need to be called. The handler shall return 28 29 PMIX EVENT ACTION COMPLETE if no further action is required. The return status of each 30 event handler and any returned pmix info t structures will be added to the results array of

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

pmix info t passed to any subsequent event handlers to help guide their operation.

31 32

### 3.5.10 Notification Function

#### Summary 2 The **pmix\_notification\_fn\_t** is called by PMIx to deliver notification of an event. 3 Advice to users 4 The PMIx ad hoc v1.0 Standard defined an error notification function with an identical name, but different signature than the v2.0 Standard described below. The ad hoc v1.0 version was removed 5 from the v2.0 Standard is not included in this document to avoid confusion. 6 PMIx v2.0 7 typedef void (\*pmix\_notification\_fn\_t) (size\_t evhdlr\_registration\_id, 8 pmix\_status\_t status, 9 10 const pmix\_proc\_t \*source, pmix\_info\_t info[], size\_t ninfo, 11 pmix\_info\_t results[], size\_t nresults, 12 pmix\_event\_notification\_cbfunc\_fn\_t cbfunc, 13 14 void \*cbdata); 15 IN evhdlr\_registration\_id Registration number of the handler being called (size\_t) 16 IN 17 status Status associated with the operation ( pmix\_status\_t ) 18 19 IN source 20 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 21 IN 22 info Information describing the event ( pmix\_info\_t ). This argument will be NULL if no 23 24 additional information was provided by the event generator. IN ninfo 25 Number of elements in the info array (size\_t) 26 IN 27 Aggregated results from prior event handlers servicing this event ( pmix\_info\_t ). This 28 argument will be **NULL** if this is the first handler servicing the event, or if no prior handlers 29 provided results. 30 IN nresults 31 32 Number of elements in the results array (size\_t) 33 IN cbfunc pmix\_event\_notification\_cbfunc\_fn\_t callback function to be executed upon 34 completion of the handler's operation and prior to handler return (function reference). 35

#### IN cbdata

Callback data to be passed to cbfunc (memory reference)

### **Description**

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

#### Advice to users -

Possible uses of the info array include:

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

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

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

### Advice to PMIx server hosts —

On the server side, the notification function is used to inform the PMIx server library's host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the pmix\_server\_notify\_event\_fn\_t function pointer, if provided.

## 3.5.11 Server Setup Application Callback Function

The PMIx\_server\_setup\_application callback function.

### Summary

Provide a function by which the resource manager can receive application-specific environmental variables and other setup data prior to launch of an application.

```
Format
1
   PMIx v2.0
2
              typedef void (*pmix_setup_application_cbfunc_t)(
 3
                                             pmix status t status,
 4
                                             pmix info t info[], size t ninfo,
                                             void *provided_cbdata,
5
6
                                             pmix_op_cbfunc_t cbfunc, void *cbdata)
              IN
7
                   status
                   returned status of the request ( pmix_status_t )
8
9
              IN
                  info
10
                   Array of info structures (array of handles)
11
                   Number of elements in the info array (integer)
12
13
                   provided_cbdata
14
                   Data originally passed to call to PMIx server setup application (memory
                   reference)
15
              IN
                   cbfunc
16
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
20
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.
               Server Direct Modex Response Callback Function
    3.5.12
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
              posted by local application processes to the host resource manager.
28
              Format
29
   PMIx v1.0
30
              typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,
31
                                             char *data, size t sz,
32
                                             void *cbdata);
```

IN 1 status Returned status of the request ( pmix\_status\_t ) 2 3 IN Pointer to a data "blob" containing the requested information (handle) 4 5 IN 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) 9 Description Define a function to be called by the PMIx server library for return of information posted by a local 10 application process (via PMIx\_Put with subsequent PMIx\_Commit) in response to a request 11 from the host RM. The returned data blob is owned by the PMIx server library and will be free'd 12 13 upon return from the function. 3.5.13 PMIx Client Connection Callback Function Summarv 15 16 Callback function for incoming connection request from a local client 17 Format PMIx v1.018 typedef void (\*pmix\_connection\_cbfunc\_t)( int incoming sd, void \*cbdata) 19 20 IN incoming\_sd 21 (integer) IN cbdata 22 23 (memory reference) **Description** 24 25 Callback function for incoming connection requests from local clients - only used by host 26 environments that wish to directly handle socket connection requests. 3.5.14 PMIx Tool Connection Callback Function Summary 28 29 Callback function for incoming tool connections.

```
Format
1
   PMIx v2.0
2
               typedef void (*pmix_tool_connection_cbfunc_t)(
3
                                                     pmix_status_t status,
 4
                                                     pmix proc t *proc, void *cbdata)
5
               IN
                    status
6
                    pmix_status_t value (handle)
7
               IN
8
                    pmix_proc_t structure containing the identifier assigned to the tool (handle)
9
               IN
                  cbdata
                    Data to be passed (memory reference)
10
               Description
11
               Callback function for incoming tool connections. The host environment shall provide a
12
13
               namespace/rank identifier for the connecting tool.
                               ——— 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

#### Format 1 PMIx v3.0 2 typedef void (\*pmix credential cbfunc t)( 3 pmix\_status\_t status, 4 pmix byte object t \*credential, pmix\_info\_t info[], size\_t ninfo, 5 6 void \*cbdata) C IN status 7 8 pmix\_status\_t value (handle) 9 IN credential pmix\_byte\_object\_t structure containing the security credential (handle) 10 IN 11 12 Array of provided by the system to pass any additional information about the credential - e.g., the identity of the issuing agent. (handle) 13 14 IN ninfo 15 Number of elements in *info* (size t) IN cbdata 16 Object passed in original request (memory reference) 17 **Description** 18 19 Define a callback function to return a requested security credential. Information provided by the issuing agent can subsequently be used by the application for a variety of purposes. Examples 20 21 include: 22 • checking identified authorizations to determine what requests/operations are feasible as a means 23 to steering workflows • compare the credential type to that of the local SMS for compatibility 24 Advice to users 25 The credential is opaque and therefore understandable only by a service compatible with the issuer. The *info* array is owned by the PMIx library and is not to be released or altered by the receiving 26 27 party.

## 8 3.5.16 Credential validation callback function

#### Summary

Callback function for security credential validation

29

```
Format
 1
   PMIx v3.0
 2
                typedef void (*pmix validation cbfunc t) (
 3
                                                        pmix_status_t status,
                                                        pmix info t info[], size t ninfo,
 4
                                                        void *cbdata);
 5
                IN
 6
                     status
 7
                     pmix_status_t value (handle)
 8
                IN
                     info
 9
                     Array of pmix info t provided by the system to pass any additional information about
                     the authentication - e.g., the effective userid and group id of the certificate holder, and any
10
                     related authorizations (handle)
11
                IN
12
                    ninfo
                     Number of elements in info (size_t)
13
14
                IN
                     cbdata
                     Object passed in original request (memory reference)
15
                Description
16
17
                Define a validation callback function to indicate if a provided credential is valid, and any
18
                corresponding information regarding authorizations and other security matters.
                                                    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
                owned by the PMIx library and is not to be released or altered by the receiving party.
22
```

# 3.5.17 IOF delivery function

#### Summary

24 25

Callback function for delivering forwarded IO to a process

```
Format
 1
   PMIx v3.0
 2
               typedef void (*pmix iof cbfunc t) (
 3
                                                      size_t iofhdlr, pmix_iof_channel_t channel,
 4
                                                      pmix proc t *source, char *payload,
                                                      pmix_info_t info[], size_t ninfo);
 5
               IN
                    iofhdlr
 6
 7
                    Registration number of the handler being invoked (size t)
 8
               IN
                    channel
 9
                    bitmask identifying the channel the data arrived on ( pmix_iof_channel_t )
10
               IN
                    source
                    Pointer to a pmix_proc_t identifying the namespace/rank of the process that generated the
11
                    data (char*)
12
13
               IN
                    payload
14
                    Pointer to character array containing the data.
               IN
                    info
15
                    Array of pmix_info_t provided by the source containing metadata about the payload.
16
                    This could include PMIX_IOF_COMPLETE (handle)
17
               IN
                    ninfo
18
19
                    Number of elements in info (size t)
               Description
20
21
               Define a callback function for delivering forwarded IO to a process. This function will be called
22
               whenever data becomes available, or a specified buffering size and/or time has been met.
                                                  Advice to users
23
               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
24
25
               library and is not to be released or altered by the receiving party.
```

# 26 3.5.18 IOF and Event registration function

### Summary

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

27

```
Format
1
   PMIx v3.0
              typedef void (*pmix_hdlr_reg_cbfunc_t) (pmix_status_t status,
2
 3
                                                                size_t refid,
 4
                                                                void *cbdata);
              IN
5
                  status
6
                   PMIX SUCCESS or an appropriate error constant ( pmix status t )
7
              IN
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)
              Description
11
12
              Callback function for calls to register handlers, e.g., event notification and IOF requests.
   3.6
            Constant String Functions
14
              Provide a string representation for several types of values. Note that the provided string is statically
              defined and must NOT be free'd.
15
16
              Summary
17
              String representation of a pmix_status_t .
   PMIx v1.0
18
              const char*
              PMIx_Error_string(pmix_status_t status);
19
20
              Summary
              String representation of a pmix_proc_state_t.
21
   PMIx v2.0
22
              const char*
              PMIx Proc state string(pmix proc state t state);
23
```

```
Summary
1
2
             String representation of a pmix scope t.
  PMIx v2.0
3
             const char*
4
             PMIx_Scope_string(pmix_scope_t scope);
             Summary
5
6
             String representation of a pmix_persistence_t.
  PMIx v2.0
7
             const char*
             PMIx_Persistence_string(pmix_persistence_t persist);
8
             Summary
9
             String representation of a pmix_data_range_t .
10
  PMIx v2.0
11
             const char*
12
             PMIx Data range string(pmix data range t range);
             Summary
13
14
             String representation of a pmix_info_directives_t.
  PMIx v2.0
             const char*
15
             PMIx_Info_directives_string(pmix_info_directives_t directives);
16
                                                 C
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);
```

```
Summary
1
2
            String representation of a pmix alloc directive t.
  PMIx v2.0
3
            const char*
4
            PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);
            Summary
5
6
            String representation of a pmix_iof_channel_t.
  PMIx v3.0
7
            const char*
            PMIx_IOF_channel_string(pmix_iof_channel_t channel);
8
            Summary
9
            String representation of a pmix_job_state_t.
10
  PMIx v4.0
11
            const char*
12
            PMIx Job state string(pmix job state t state);
            Summary
13
14
            String representation of a PMIx attribute
  PMIx v4.0
            const char*
15
            PMIx_Get_attribute_string(char *attributename);
16
17
            Summary
18
            Return the PMIx attribute name corresponding to the given attribute string
  PMIx v4.0
19
            const char*
            PMIx Get attribute name(char *attributestring);
20
```

```
Summary

String representation of a pmix_link_state_t

PMIx v4.0

C

const char*

PMIx_Link_state_string(pmix_link_state_t state);
```

#### CHAPTER 4

# Initialization and Finalization

The PMIx library is required to be initialized and finalized around the usage of most of the APIs. 1 The APIs that may be used outside of the initialized and finalized region are noted. All other APIs 2 must be used inside this region. 3 There are three sets of initialization and finalization functions depending upon the role of the 4 5 process in the PMIx universe. Each of these functional sets are described in this chapter. Note that a process can only call one of the init/finalize functional pairs - e.g., a process that calls the client 6 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 includes overhead and an increased memory footprint. Tool initialization automatically searches for 11 a server to which it can connect — if declared as a *launcher*, the PMIx library sets up the required 12 "hooks" for other tools (e.g., debuggers) to attach to it. 13 14 **4.1** Query 15 The API defined in this section can be used by any PMIx process, regardless of their role in the PMIx universe. 16 4.1.1 PMIx Initialized Format 18 PMIx v1.0 int PMIx Initialized(void) 19 A value of 1 (true) will be returned if the PMIx library has been initialized, and 0 (false) otherwise. 20 21 The return value is an integer for historical reasons as that was the signature of prior PMI libraries.

### **Description** 1 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. 4.1.2 PMIx Get\_version 5 Summary Get the PMIx version information. 6 7 Format *PMIx v1.0* 8 const char\* PMIx Get version(void) **Description** 9 Get the PMIx version string. Note that the provided string is statically defined and must *not* be 10 11 free'd. Client Initialization and Finalization 13 Initialization and finalization routines for PMIx clients. Advice to users The PMIx ad hoc v1.0 Standard defined the PMIx Init function, but modified the function 14 15 signature in the v1.2 version. The ad hoc v1.0 version is not included in this document to avoid confusion. 16 17 **4.2.1** PMIx\_Init

18

19

Summary

Initialize the PMIx client library

|          | Format  |
|----------|---|
| MIx v1.2 |   |
|          | pmix_status_t   |
|          | <pre>PMIx_Init(pmix_proc_t *proc,     pmix_info_t info[], size_t ninfo)</pre>                     |
|          | pmix_inio_t inio[], size_t ninio;   |
|          |   |
|          | INOUT proc  |
|          | proc structure (handle)   |
|          | IN info   |
|          | Array of pmix_info_t structures (array of handles)  IN ninfo                                      |
|          | Number of element in the <i>info</i> array (size_t)   |
|          | Returns <b>PMIX_SUCCESS</b> or a negative value corresponding to a PMIx error constant.           |
|          |   |
|          | 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.                                    |
|          | <pre>PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)</pre>  |
|          | POSIX <i>mode_t</i> (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.                                |
|          | <pre>PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)</pre>  |
|          | 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.                               |
|          | <pre>PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*)</pre>                                       |
|          | 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 <i>not</i> to be used.                           |

1 PMIX\_TCP\_IPV4\_PORT "pmix.tcp.ipv4" (int) The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be 2 3 supported for specifying the port to be used. PMIX\_TCP\_IPV6\_PORT "pmix.tcp.ipv6" (int) 4 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be 5 6 supported for specifying the port to be used. PMIX\_TCP\_DISABLE\_IPV4 "pmix.tcp.disipv4" (bool) 7 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, 8 this attribute may be supported for disabling it. 9 10 PMIX TCP DISABLE IPV6 "pmix.tcp.disipv6" (bool) Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, 11 this attribute may be supported for disabling it. 12 PMIX\_EVENT\_BASE "pmix.evbase" (struct event\_base \*) 13 Pointer to libevent **base** to use in place of the internal progress thread. 14 PMIX\_GDS\_MODULE "pmix.gds.mod" (char\*) 15 Comma-delimited string of desired modules. This attribute is specific to the PRI and 16 controls only the selection of GDS module for internal use by the process. Module selection 17 for interacting with the server is performed dynamically during the connection process. 18 **Description** 19 Initialize the PMIx client, returning the process identifier assigned to this client's application in the 20 provided pmix\_proc\_t struct. Passing a value of **NULL** for this parameter is allowed if the user 21 wishes solely to initialize the PMIx system and does not require return of the identifier at that time. 22 23 When called, the PMIx client shall check for the required connection information of the local PMIx server and establish the connection. If the information is not found, or the server connection fails, 24 then an appropriate error constant shall be returned. 25 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 startup information provided by the resource manager shall be made available to the client process 28 29 via subsequent calls to PMIx\_Get . The PMIx client library shall be reference counted, and so multiple calls to PMIx\_Init are 30 31 allowed by the standard. Thus, one way for an application process to obtain its namespace and rank is to simply call **PMIx Init** with a non-NULL *proc* parameter. Note that each call to 32

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

PMIx Init must be balanced with a call to PMIx Finalize to maintain the reference count.

33 34

<sup>&</sup>lt;sup>1</sup>http://libevent.org/

Multiple calls to PMIx\_Init shall not include conflicting directives. The PMIx\_Init function 1 2 will return an error when directives that conflict with prior directives are encountered. 4.2.2 PMIx Finalize Summary 4 5 Finalize the PMIx client library. Format 6 PMIx v1.07 pmix status t PMIx\_Finalize(const pmix\_info\_t info[], size\_t ninfo) 8 9 IN info Array of pmix info t structures (array of handles) 10 11 IN Number of element in the *info* array (size\_t) 12 13 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: 14 PMIX EMBED\_BARRIER "pmix.embed.barrier" (bool) 15 Execute a blocking fence operation before executing the specified operation. For example, 16 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. **Description** 19 20 Decrement the PMIx client library reference count. When the reference count reaches zero, the library will finalize the PMIx client, closing the connection with the local PMIx server and 21 22 releasing all internally allocated memory. **Tool Initialization and Finalization** 24 Initialization and finalization routines for PMIx tools. 4.3.1 PMIx tool init

Summary

Initialize the PMIx library for operating as a tool.

```
Format
1
   PMIx v2.0
2
             pmix status t
              PMIx_tool_init(pmix_proc_t *proc,
 3
 4
                               pmix info t info[], size t ninfo)
                                              — C
              INOUT proc
5
6
                  pmix proc t structure (handle)
7
              IN
8
                  Array of pmix info t structures (array of handles)
9
              IN ninfo
                  Number of element in the info array (size t)
10
11
              Returns PMIX SUCCESS or a negative value corresponding to a PMIx error constant.
                                           Required Attributes
              _____
12
              The following attributes are required to be supported by all PMIx libraries:
              PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
13
                   Name of the namespace to use for this tool.
14
              PMIX TOOL RANK "pmix.tool.rank" (uint32 t)
15
                   Rank of this tool.
16
17
              PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
18
                   The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.
19
              PMIX SERVER URI "pmix.srvr.uri" (char*)
                   URI of the PMIx server to be contacted.
20
                                            Optional Attributes -----
                -----
              The following attributes are optional for implementers of PMIx libraries:
21
              PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)
22
23
                   The requestor requires that a connection be made only to a local, system-level PMIx server.
24
              PMIX CONNECT SYSTEM FIRST "pmix.cnct.sys.first" (bool)
                   Preferentially, look for a system-level PMIx server first.
25
              PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)
26
27
                   PID of the target PMIx server for a tool.
28
              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
29
30
                   file: < name of file containing it >.
31
              PMIX_CONNECT_RETRY_DELAY "pmix.tool.retry" (uint32_t)
```

1 Time in seconds between connection attempts to a PMIx server. PMIX CONNECT MAX RETRIES "pmix.tool.mretries" (uint32 t) 2 3 Maximum number of times to try to connect to PMIx server. 4 PMIX SOCKET MODE "pmix.sockmode" (uint32 t) POSIX mode\_t (9 bits valid) If the library supports socket connections, this attribute may 5 be supported for setting the socket mode. 6 7 PMIX TCP REPORT URI "pmix.tcp.repuri" (char\*) If provided, directs that the TCP URI be reported and indicates the desired method of 8 reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket 9 10 connections, this attribute may be supported for reporting the URI. PMIX TCP IF INCLUDE "pmix.tcp.ifinclude" (char\*) 11 Comma-delimited list of devices and/or CIDR notation to include when establishing the 12 TCP connection. If the library supports TCP socket connections, this attribute may be 13 supported for specifying the interfaces to be used. 14 15 PMIX TCP IF EXCLUDE "pmix.tcp.ifexclude" (char\*) Comma-delimited list of devices and/or CIDR notation to exclude when establishing the 16 TCP connection. If the library supports TCP socket connections, this attribute may be 17 supported for specifying the interfaces that are *not* to be used. 18 19 PMIX TCP IPV4 PORT "pmix.tcp.ipv4" (int) The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be 20 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 supported for specifying the port to be used. 24 25 PMIX TCP DISABLE IPV4 "pmix.tcp.disipv4" (bool) Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, 26 27 this attribute may be supported for disabling it. PMIX\_TCP\_DISABLE\_IPV6 "pmix.tcp.disipv6" (bool) 28 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, 29 this attribute may be supported for disabling it. 30 PMIX EVENT BASE "pmix.evbase" (struct event base \*) 31 Pointer to libevent<sup>2</sup> **event base** to use in place of the internal progress thread. 32 PMIX GDS MODULE "pmix.gds.mod" (char\*) 33 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.

<sup>&</sup>lt;sup>2</sup>http://libevent.org/

### Description

Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided **pmix\_proc\_t** struct. The *info* array is used to pass user requests pertaining to the init and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives are desired.

If called with the PMIX\_TOOL\_DO\_NOT\_CONNECT attribute, the PMIx tool library will fully initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later point in time, if desired. In all other cases, the PMIx tool library will attempt to connect to according to the following precedence chain:

- if PMIX\_SERVER\_URI or PMIX\_TCP\_URI is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified.
   PMIX\_SERVER\_URI is the preferred method as it is more generalized PMIX\_TCP\_URI is provided for those cases where the user specifically wants to use a TCP transport for the connection and wants to error out if it isn't available or cannot succeed. The PMIx library will return an error if connection fails it will not proceed to check for other connection options as the user specified a particular one to use
- if PMIX\_SERVER\_PIDINFO was provided, then the tool will search under the directory
  provided by the PMIX\_SERVER\_TMPDIR environmental variable for a rendezvous file created
  by the process corresponding to that PID. The PMIx library will return an error if the rendezvous
  file cannot be found, or the connection is refused by the server
- if PMIX\_CONNECT\_TO\_SYSTEM is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX\_SYSTEM\_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. An error is returned if the rendezvous file cannot be found or the connection is refused.
- if PMIX\_CONNECT\_SYSTEM\_FIRST is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the directory specified by the PMIX\_SYSTEM\_TMPDIR environmental variable. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused the PMIx library will silently continue to the next option
- by default, the tool will search the directory tree under the directory provided by the PMIX\_SERVER\_TMPDIR environmental variable for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error.

If successful, the function will return **PMIX\_SUCCESS** and will fill the provided structure (if provided) with the server-assigned namespace and rank of the tool. Note that each connection attempt in the above precedence chain will retry (with delay between each retry) a number of times according to the values of the corresponding attributes. Default is no retries.

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 **PMIx** tool init with a non-NULL parameter. 3 4.3.2 PMIx\_tool\_finalize 5 Summary Finalize the PMIx library for a tool connection. 6 7 Format PMIx v2.0 pmix status t 8 9 PMIx tool finalize(void) 10 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 11 Finalize the PMIx tool library, closing the connection to the server. An error code will be returned 12 if, for some reason, the connection cannot be cleanly terminated — in this case, the connection is 13 14 dropped. 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 specified server. 18 **Format** 19 *PMIx v3.0* 20 pmix\_status\_t 21 PMIx tool\_connect\_to\_server(pmix\_proc\_t \*proc, pmix\_info\_t info[], size\_t ninfo) 22 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. 23

### Required Attributes

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

PMIX\_CONNECT\_TO\_SYSTEM "pmix.cnct.sys" (bool)

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

PMIX\_CONNECT\_SYSTEM\_FIRST "pmix.cnct.sys.first" (bool)

Preferentially, look for a system-level PMIx server first.

PMIX\_SERVER\_URI "pmix.srvr.uri" (char\*)

URI of the PMIx server to be contacted.

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

Name of the namespace to use for this PMIx server.

PMIX\_SERVER\_PIDINFO "pmix.srvr.pidinfo" (pid\_t)

PID of the target PMIx server for a tool.

### **Description**

1

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

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 <code>pmix\_proc\_t</code> struct. Passing a value of <code>NULL</code> for this parameter is allowed if the user wishes solely to connect to the PMIx server and does not require return of the identifier at that time.

### Advice to PMIx library implementers —

PMIx tools and clients are prohibited from being connected to more than one server at a time to avoid confusion in subsystems such as event notification.

When a tool connects to a server that is under a different namespace manager (e.g., host RM) as the prior server, the identifier of the tool must remain unique in the namespaces. This may require the identifier of the tool to be changed on-the-fly, that is, the *proc* parameter would be filled (if non-NULL) with a different nspace/rank from the current tool identifier.

### Advice to users —

Passing a **NULL** value for the *info* pointer is not allowed and will result in returning an error.

Some PMIx implementations (for example, the current PRI) may not support connecting to a server that is not under the same namespace manager (e.g., host RM) as the tool.

### 4.4 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers. 2 4.4.1 3 PMIx\_server\_init Summary 4 Initialize the PMIx server. 5 **Format** 6 PMIx v1.07 pmix status t PMIx server init(pmix server module t \*module, 8 pmix info t info[], size t ninfo) 9 \_\_\_\_ C \_\_\_ INOUT module 10 pmix\_server\_module\_t structure (handle) 11 12 IN info Array of **pmix\_info\_t** structures (array of handles) 13 14 IN ninfo Number of elements in the *info* array (size\_t) 15 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: PMIX\_SERVER\_NSPACE "pmix.srv.nspace" (char\*) 18 Name of the namespace to use for this PMIx server. 19 20 PMIX SERVER RANK "pmix.srv.rank" (pmix rank t) 21 Rank of this PMIx server 22 PMIX SERVER TMPDIR "pmix.srvr.tmpdir" (char\*) Top-level temporary directory for all client processes connected to this server, and where the 23 24 PMIx server will place its tool rendezvous point and contact information. PMIX\_SYSTEM\_TMPDIR "pmix.sys.tmpdir" (char\*) 25 Temporary directory for this system, and where a PMIx server that declares itself to be a 26 system-level server will place a tool rendezvous point and contact information. 27 28 PMIX SERVER TOOL SUPPORT "pmix.srvr.tool" (bool) The host RM wants to declare itself as willing to accept tool connection requests. 29 PMIX SERVER SYSTEM SUPPORT "pmix.srvr.sys" (bool) 30 The host RM wants to declare itself as being the local system server for PMIx connection 31 32 requests.

| <b>A</b>   |         |
|--|---------|
| ▼ 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 be supported for setting the socket mode.  | may     |
| 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 so 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 to 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 to TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are <i>not</i> 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 masupported for specifying the port to be used.   | y be    |
| PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)  The IPv6 port to be used. If the library supports IPV6 connections, this attribute masupported for specifying the port to be used.   | y be    |
| PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)  Set to true to disable IPv4 family of addresses. If the library supports IPV4 connecthis attribute may be supported for disabling it.   | ctions, |
| PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)  Set to true to disable IPv6 family of addresses. If the library supports IPV6 connections attribute may be supported for disabling it.  | ctions, |
| 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 be supported for enabling or disabling it.                       | e may   |

| 1<br>2                  | PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)  Pointer to libevent <sup>3</sup> event_base to use in place of the internal progress thread.   |
|-------------------------|---|
| 3<br>4<br>5<br>6        | PMIX_GDS_MODULE "pmix.gds.mod" (char*)  Comma-delimited string of desired modules. This attribute is specific to the PRI and controls only the selection of GDS module for internal use by the process. Module selection for interacting with the server is performed dynamically during the connection process.  |
| 7<br>8<br>9<br>10<br>11 | Description  Initialize the PMIx server support library, and provide a pointer to a pmix_server_module_t structure containing the caller's callback functions. The array of pmix_info_t structs is used to pass additional info that may be required by the server when initializing. For example, it may include the PMIX_SERVER_TOOL_SUPPORT attribute, thereby indicating that the daemon is willing to accept connection requests from tools. |
| 13<br>14<br>15          | Providing a value of <b>NULL</b> for the <i>module</i> argument is permitted, as is passing an empty <i>module</i> structure. Doing so indicates that the host environment will not provide support for multi-node operations such as <b>PMIx_Fence</b> , but does intend to support local clients access to information.   |
| 16 <b>4.4.2</b>         | PMIx_server_finalize Summary Finalize the PMIx server library.  |
| 19<br><i>PMIx v1.</i> 0 | Format C  |
| 20<br>21                | <pre>pmix_status_t PMIx_server_finalize(void)</pre>   |
| 22                      | Returns <b>PMIX_SUCCESS</b> or a negative value corresponding to a PMIx error constant.   |
| 23<br>24<br>25          | <b>Description</b> Finalize the PMIx server support library, terminating all connections to attached tools and any local clients. All memory usage is released.   |
|                         | <sup>3</sup> http://libevent.org/   |

#### **CHAPTER 5**

1 2

3

4 5

6 7

8

11

24

# **Key/Value Management**

Management of key-value pairs in PMIx is a distributed responsibility. While the stated objective of the PMIx community is to eliminate collective operations, it is recognized that the traditional method of posting/exchanging data must be supported until that objective can be met. This method relies on processes to discover and post their local information which is collected by the local PMIx server library. Global exchange of the posted information is then executed via a collective operation performed by the host SMS servers. The PMIx\_Put and PMIx\_Commit APIs, plus an attribute directing PMIx\_Fence to globally collect the data posted by processes, are provided for this purpose.

# 5.1 Setting and Accessing Key/Value Pairs

### 0 5.1.1 PMIx\_Put

Summary

```
12
               Push a key/value pair into the client's namespace.
               Format
13
   PMIx v1.0
14
               pmix_status_t
15
               PMIx_Put(pmix_scope_t scope,
16
                          const pmix key t key,
17
                          pmix value t *val)
               IN
                    scope
18
19
                   Distribution scope of the provided value (handle)
               IN
20
21
                   key ( pmix_key_t )
               IN
                   value
22
                   Reference to a pmix_value_t structure (handle)
23
```

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

| 1<br>2<br>3      | <b>Description</b> Push a value into the client's namespace. The client's PMIx library will cache the information locally until PMIx_Commit is called.   |
|------------------|--|
| 4<br>5<br>6<br>7 | The provided <i>scope</i> is passed to the local PMIx server, which will distribute the data to other processes according to the provided scope. The <b>pmix_scope_t</b> values are defined in Section 3.2.9 on page 35. Specific implementations may support different scope values, but all implementations must support at least <b>PMIX_GLOBAL</b> . |
| 8<br>9<br>10     | The <b>pmix_value_t</b> 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 <i>value</i> into internal memory.  Advice to PMIx library implementers   |
| 11<br>12<br>13   | The PMIx server library will properly pack/unpack data to accommodate heterogeneous environments. The host SMS is not involved in this action. The <i>value</i> argument must be copied - the caller is free to release it following return from the function.   |
|                  | Advice to users  |
| 14<br>15         | 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.  |
| 16<br>17<br>18   | 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.  |

# 19 **5.1.2 PMIx\_Get**

### 20 **Summary** 21 Retrieve a ke

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

```
Format
1
   PMIx v1.0
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,
                           pmix_value_t **val)
5
               IN
6
                    proc
7
                    process reference (handle)
8
               IN
9
                    key to retrieve (pmix key t)
10
               IN
                   info
                    Array of info structures (array of handles)
11
               IN ninfo
12
13
                    Number of element in the info array (integer)
14
               OUT val
                    value (handle)
15
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)
                     Specified operation should immediately return an error from the PMIx server if the requested
22
23
                     data cannot be found - do not request it from the host RM.
               PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
24
                     Scope of the data to be found in a PMIx Get call.
25
               PMIX SESSION INFO "pmix.ssn.info" (bool)
26
27
                     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
28
                     PMIX_SESSION_ID attribute identifying the desired target.
29
               PMIX JOB INFO "pmix.job.info" (bool)
30
```

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX\_JOBID or PMIX\_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

#### PMIX APP INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a PMIX\_APPNUM attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

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

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX\_NODEID** or **PMIX\_HOSTNAME** attribute identifying the desired target.

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

Request that any pointers in the returned value point directly to values in the key-value store and indicate that the address provided for the return value points to a statically defined memory location. Returned non-pointer values should therefore be copied directly into the provided memory. Pointers in the returned value should point directly to values in the key-value store. User is responsible for *not* releasing memory on any returned pointer value. Note that a return status of **PMIX\_ERR\_GET\_MALLOC\_REQD** indicates that direct pointers could not be supported - thus, the returned data contains allocated memory that the user must release.

### ------ Optional Attributes ------

The following attributes are optional for host environments:

#### PMIX\_TIMEOUT "pmix.timeout" (int)

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

### Advice to PMIx library implementers -

We recommend that implementation of the PMIX\_TIMEOUT attribute be left to the host environment due to race condition considerations between delivery of the data by the host environment versus internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX\_TIMEOUT to the host environment so that multiple competing timeouts are not created.

### Description

Retrieve information for the specified *key* as published by the process identified in the given <code>pmix\_proc\_t</code>, 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 <code>PMIX\_RANK\_WILDCARD</code>. The list of data referenced in this way is maintained on the PMIx web site at <a href="https://pmix.org/support/faq/wildcard-rank-access/">https://pmix.org/support/faq/wildcard-rank-access/</a> but includes items such as the number of processes in the namespace ( <code>PMIX\_JOB\_SIZE</code> ), total available slots in the allocation ( <code>PMIX\_UNIV\_SIZE</code> ), and the number of nodes in the allocation ( <code>PMIX\_NUM\_NODES</code> ).

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              |                  | Format  |  |  |  |  |
|----------------|------------------|---|--|--|--|--|
|                | <i>PMIx v1.0</i> |   |  |  |  |  |
| 2              |                  | pmix_status_t   |  |  |  |  |
| 3              |                  | PMIx_Get_nb(const pmix_proc_t *proc, const char key[],  |  |  |  |  |
| 4<br>5         |                  | <pre>const pmix_info_t info[], size_t ninfo, pmix_value_shfung_t_shfung_vaid_tshdata)</pre>   |  |  |  |  |
| 5              |                  | <pre>pmix_value_cbfunc_t cbfunc, void *cbdata)</pre>  |  |  |  |  |
|                |                  | 0   |  |  |  |  |
| 6              |                  | IN proc   |  |  |  |  |
| 7              |                  | process reference (handle)  |  |  |  |  |
| 8              |                  | IN key  |  |  |  |  |
| 9              |                  | key to retrieve (string)  |  |  |  |  |
| 10<br>11       |                  | IN info Array of info structures (array of handles)   |  |  |  |  |
| 12             |                  | IN ninfo  |  |  |  |  |
| 13             |                  | Number of elements in the <i>info</i> array (integer)   |  |  |  |  |
| 14             |                  | IN cbfunc   |  |  |  |  |
| 15             |                  | Callback function (function reference)  |  |  |  |  |
| 16             |                  | IN cbdata   |  |  |  |  |
| 17             |                  | Data to be passed to the callback function (memory reference)   |  |  |  |  |
| 18             |                  | Returns one of the following:   |  |  |  |  |
| 19<br>20<br>21 |                  | • <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API. |  |  |  |  |
| 22<br>23       |                  | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called   |  |  |  |  |
| 24<br>25       |                  | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called  |  |  |  |  |
| 26<br>27       |                  | If executed, the status returned in the provided callback function will be one of the following 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  |  |  |  |  |
|                |                  | ▼   |  |  |  |  |
| 31             |                  | The following attributes are required to be supported by all PMIx libraries:  |  |  |  |  |
| 32<br>33<br>34 |                  | PMIX_OPTIONAL "pmix.optional" (bool)  Look only in the client's local data store for the requested value - do not request data from the PMIx server if not found.   |  |  |  |  |

#### 1 PMIX\_IMMEDIATE "pmix.immediate" (bool) 2 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. 3 4

### PMIX DATA\_SCOPE "pmix.scope" (pmix\_scope\_t)

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

#### PMIX SESSION INFO "pmix.ssn.info" (bool)

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

#### PMIX JOB INFO "pmix.job.info" (bool)

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX\_JOBID or PMIX\_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

### PMIX APP INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a PMIX APPNUM attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

#### PMIX NODE INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the PMIX NODEID or PMIX HOSTNAME attribute identifying the desired target.

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

Request that any pointers in the returned value point directly to values in the key-value store and indicate that user takes responsibility for properly releasing memory on the returned value (i.e., free'ing the value structure but not the pointer fields). Note that a return status of PMIX\_ERR\_GET\_MALLOC\_REQD indicates that direct pointers could not be supported thus, the returned data contains allocated memory that the user must release.

### 

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 ( $\theta$  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.

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

### 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.

### 20 5.1.4 PMIx\_Store\_internal

#### Summary

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

#### **Format** 1 PMIx v1.0 2 pmix status t 3 PMIx\_Store\_internal(const pmix\_proc\_t \*proc, 4 const pmix key t key, pmix\_value\_t \*val); 5 IN 6 proc 7 process reference (handle) 8 IN 9 key to retrieve (string) 10 IN val Value to store (handle) 11 12 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 13 14 Store some data locally for retrieval by other areas of the proc. This is data that has only internal 15 scope - it will never be "pushed" externally. Accessing information: examples 5.1.5 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. 5.1.5.1 Session-level information 21 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 and a rank of PMIX\_RANK\_WILDCARD; or 26 • for non-specific attributes (e.g., PMIX\_NUM\_NODES ), including the PMIX\_SESSION\_INFO 27 28 attribute to indicate that the session-level information for that attribute is being requested

29

Example requests are shown below:

```
1
             pmix info t info;
2
             pmix value t *value;
3
             pmix_status_t rc;
4
             pmix_proc_t myproc, wildcard;
5
6
             /* initialize the client library */
7
             PMIx_Init(&myproc, NULL, 0);
8
9
             /* get the #slots in our session */
10
             PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
             rc = PMIx Get(&wildcard, PMIX UNIV SIZE, NULL, 0, &value);
11
12
13
             /* get the #nodes in our session */
             PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
14
             rc = PMIx Get(&wildcard, PMIX NUM NODES, &info, 1, &value);
15
16
             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
17
18
             PMIX_SESSION_ID attribute identifying the target session. In the latter case, the proc argument
19
             to PMIx_Get will be ignored:
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);
```

### 5.1.5.2 Job-level information

2

3

4 5

6 7

8

25

26 27

28

29

30

31 32

33

34

35

36

Information regarding a job can be obtained by the following methods:

- for job-level attributes (e.g., PMIX\_JOB\_SIZE or PMIX\_JOB\_NUM\_APPS), specifying the namespace of the job and a rank of PMIX\_RANK\_WILDCARD for the proc argument to PMIx\_Get; or
- for non-specific attributes (e.g., **PMIX\_NUM\_NODES**), including the **PMIX\_JOB\_INFO** attribute to indicate that the job-level information for that attribute is being requested

Example requests are shown below:

```
9
            pmix info t info;
10
            pmix_value_t *value;
            pmix_status_t rc;
11
12
            pmix_proc_t myproc, wildcard;
13
            /* initialize the client library */
14
15
            PMIx Init(&myproc, NULL, 0);
16
            /* get the #apps in our job */
17
            PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
18
19
            rc = PMIx Get(&wildcard, PMIX JOB NUM APPS, NULL, 0, &value);
20
21
            /* get the #nodes in our job */
            PMIX INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
22
            rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
23
```

### 5.1.5.3 Application-level information

Information regarding an application can be obtained by the following methods:

- for application-level attributes (e.g., **PMIX\_APP\_SIZE** ), specifying the namespace and rank of a process within that application;
- for application-level attributes (e.g., PMIX\_APP\_SIZE), including the PMIX\_APPNUM
   attribute specifying the application whose information is being requested. In this case, the
   namespace field of the *proc* argument is used to reference the job containing the application the rank field is ignored;
- or application-level attributes (e.g., PMIX\_APP\_SIZE), including the PMIX\_APPNUM and PMIX\_NSPACE or PMIX\_JOBID attributes specifying the job/application whose information is being requested. In this case, the *proc* argument is ignored;
- for non-specific attributes (e.g., **PMIX\_NUM\_NODES**), including the **PMIX\_APP\_INFO** attribute to indicate that the application-level information for that attribute is being requested

```
Example requests are shown below:
```

1

37

38

39

```
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
            /* initialize the client library */
8
9
            PMIx Init(&myproc, NULL, 0);
10
11
            /* get the #processes in our application */
            rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
12
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
            /* assume for this example that we are in the first application
19
             * and we want the #nodes in the second application - use the
20
             * rank of the first process in that application, remembering
21
22
             * that ranks start at zero */
23
            PMIX PROC LOAD (&otherproc, myproc.nspace, appsize);
24
25
            PMIX INFO LOAD (&info, PMIX APP INFO, NULL, PMIX BOOL);
            rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);
26
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 */
            appnum = 1;
31
32
            PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
33
            PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
            rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
34
35
```

C

### 5.1.5.4 Process-level information

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

#### 5.1.5.5 Node-level information

2

3

5

6 7

8

9

10

Information regarding a node within the system can be obtained by the following methods:

- for node-level attributes (e.g., **PMIX\_NODE\_SIZE** ), specifying the namespace and rank of a process executing on the target node;
- for node-level attributes (e.g., PMIX\_NODE\_SIZE), including the PMIX\_NODEID or PMIX\_HOSTNAME attribute specifying the node whose information is being requested. In this case, the *proc* argument's values are ignored; or
- for non-specific attributes (e.g., PMIX\_MAX\_PROCS), including the PMIX\_NODE\_INFO
  attribute to indicate that the node-level information for that attribute is being requested

Example requests are shown below:

```
pmix_info_t info[2];
11
12
           pmix_value_t *value;
13
           pmix status t rc;
           pmix proc t myproc, otherproc;
14
           uint32 t nodeid;
15
16
17
           /* initialize the client library */
18
           PMIx Init(&myproc, NULL, 0);
19
20
           /* get the #procs on our node */
           rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);
21
22
23
           /* get the #slots on another node */
24
           PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
           PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
25
           rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);
26
27
                  Advice to users
           An explanation of the use of PMIx Get versus PMIx Query info nb is provided in 7.1.4.1.
28
```

# 5.2 Exchanging Key/Value Pairs

The APIs defined in this section push key/value pairs from the client to the local PMIx server, and circulate the data between PMIx servers for subsequent retrieval by the local clients.

30

### 5.2.1 PMIx Commit Summary 2 3 Push all previously **PMIx\_Put** values to the local PMIx server. Format *PMIx v1.0* 5 pmix status t PMIx Commit(void) 6 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 7 This is an asynchronous operation. The PRI will immediately return to the caller while the data is 8 9 transmitted to the local server in the background. Advice to users — The local PMIx server will cache the information locally - i.e., the committed data will not be 10

circulated during PMIx\_Commit . Availability of the data upon completion of PMIx\_Commit is

### 13 5.2.2 PMIx\_Fence

therefore implementation-dependent.

### 14 Summary

11

12

15 16 Execute a blocking barrier across the processes identified in the specified array, collecting information posted via **PMI** as directed.

| FU   | ermat   |
|------|---|
| v1.0 |   |
| pm   | ix_status_t   |
| PM   | <pre>Ix_Fence(const pmix_proc_t procs[], size_t nprocs,</pre>   |
|      | const pmix_info_t info[], size_t ninfo)   |
|      | C   |
| IN   | procs   |
|      | Array of pmix_proc_t structures (array of handles)  |
| IN   | nprocs  |
|      | Number of element in the <i>procs</i> array (integer)   |
| IN   | info  |
| IN   | Array of info structures (array of handles) ninfo   |
|      | Number of element in the <i>info</i> array (integer)  |
| Da   | turns <b>PMIX_SUCCESS</b> or a negative value corresponding to a PMIx error constant.   |
| Re   | · · · · · · · · · · · · · · · · · · ·   |
| •    | Required Attributes   |
| Th   | e following attributes are required to be supported by all PMIx libraries:  |
| PM   | IX_COLLECT_DATA "pmix.collect" (bool)   |
|      | Collect data and return it at the end of the operation.   |
| _    |   |
| •    | Optional Attributes   |
| Th   | e following attributes are optional for host environments:  |
| PM   | IX_TIMEOUT "pmix.timeout" (int)   |
|      | Time in seconds before the specified operation should time out ( $\theta$ 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.   |
| PM   | <pre>IX_COLLECTIVE_ALGO "pmix.calgo" (char*)</pre>  |
|      | 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. |
|      | **  |
| PM   | IX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)   |
|      | If true, indicates that the requested choice of algorithm is mandatory.   |

| A al: a a | 4  |       | 1:1     |     | lementers | _  |
|-----------|----|-------|---------|-----|-----------|----|
| AUMICE    | IO | PIMIX | IIDrary | ımn | lementer  | е. |
|           |    |       |         |     |           |    |

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               |                  | Format  |  |  |
|-----------------|------------------|---|--|--|
|                 | <i>PMIx v1.0</i> | · · · · · · · · · · · · · · · · · · ·   |  |  |
| 2 pmix_status_t |                  |   |  |  |
| 3               |                  | <pre>PMIx_Fence_nb(const pmix_proc_t procs[], size_t nprocs,</pre>  |  |  |
| 4               |                  | <pre>const pmix_info_t info[], size_t ninfo,</pre>  |  |  |
| 5               |                  | <pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>   |  |  |
|                 |                  | C   |  |  |
| 6               |                  | IN procs  |  |  |
| 7               |                  | Array of pmix_proc_t structures (array of handles)  |  |  |
| 8               |                  | IN nprocs   |  |  |
| 9               |                  | Number of element in the <i>procs</i> array (integer)   |  |  |
| 10              |                  | IN info   |  |  |
| 11              |                  | Array of info structures (array of handles)   |  |  |
| 12              |                  | IN ninfo  |  |  |
| 13              |                  | Number of element in the <i>info</i> array (integer)  |  |  |
| 14              |                  | IN cbfunc   |  |  |
| 15              |                  | Callback function (function reference)  |  |  |
| 16              |                  | IN cbdata   |  |  |
| 17              |                  | Data to be passed to the callback function (memory reference)   |  |  |
| 18              |                  | Returns one of the following:   |  |  |
| 19              |                  | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result                       |  |  |
| 20              |                  | will be returned in the provided <i>cbfunc</i> . 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 <i>success</i> - the <i>cbfunc</i> will <i>not</i> 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 <i>cbfunc</i> will <i>not</i> 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.   |  |  |
|                 |                  | <b>A</b>  |  |  |

| ▼ Optional Attributes   |
|---|
| The following attributes are optional for host environments that support this operation:  |
| PMIX_TIMEOUT "pmix.timeout" (int)  Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.   |
| PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)  Comma-delimited list of algorithms to use for the collective operation. PMIx does not impose any requirements on a host environment's collective algorithms. Thus, the acceptable values for this attribute will be environment-dependent - users are encouraged to check their host environment for supported values. |
| PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  |
| Advice to PMIx library implementers   |

We recommend that implementation of the PMIX\_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX\_TIMEOUT to the host environment so that multiple competing timeouts are not created.

Note that PMIx libraries may choose to implement an optimization for the case where only the calling process is involved in the fence operation by immediately returning 
PMIX\_OPERATION\_SUCCEEDED from the client's call in lieu of passing the fence operation to a 
PMIx server. Fence operations involving more than just the calling process must be communicated to the PMIx server for proper execution of the included barrier behavior.

Similarly, fence operations that involve only processes that are clients of the same PMIx server may be resolved by that server without referral to its host environment as no inter-node coordination is required.

### Description

Nonblocking **PMIx\_Fence** routine. Note that the function will return an error if a **NULL** callback function is given.

Note that for scalability reasons, the default behavior for **PMIx\_Fence\_nb** is to not collect the data.

See the **PMIx\_Fence** description for further details.

# 5.3 Publish and Lookup Data

The APIs defined in this section publish data from one client that can be later exchanged and looked 2 3 up by another client. PMIx libraries that support any of the functions in this section are required to support all of them. 4 — Advice to PMIx server hosts — Host environments that support any of the functions in this section are required to support all of 5 6 5.3.1 PMIx Publish Summary 8 Publish data for later access via PMIx\_Lookup. 9 **Format** 10 *PMIx v1.0* 11 pmix\_status\_t PMIx Publish(const pmix\_info\_t info[], size\_t ninfo) 12 IN info 13 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 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is 19 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: 1 PMIX\_TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 the target process from ever exposing its data. 5 PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t) 6 Value for calls to publish/lookup/unpublish or for monitoring event notifications. 7 PMIX\_PERSISTENCE "pmix.persist" (pmix\_persistence\_t) 8 Value for calls to PMIx Publish. 9 Advice to PMIx library implementers — 10 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 11 12 internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT 13 directly in the PMIx server library must take care to resolve the race condition and should avoid 14 passing **PMIX TIMEOUT** to the host environment so that multiple competing timeouts are not 15 created. **Description** 16 17 Publish the data in the *info* array for subsequent lookup. By default, the data will be published into 18 the PMIX RANGE SESSION range and with PMIX PERSIST APP persistence. Changes to 19 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 20 parameter instructs the server as to how long the data is to be retained. 21 22 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 23 looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be 24 25 released upon return from the blocking function call. Advice to users -Publishing duplicate keys is permitted provided they are published to different ranges. 26 —— Advice to PMIx library implementers ———— Implementations should, to the best of their ability, detect duplicate keys being posted on the same 27 data range and protect the user from unexpected behavior by returning the 28 PMIX\_ERR\_DUPLICATE\_KEY error. 29

#### 5.3.2 PMIx Publish nb Summary 3 Nonblocking PMIx Publish routine. 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) IN info 8 9 Array of info structures (array of handles) 10 IN ninfo Number of element in the *info* array (integer) 11 IN cbfunc 12 Callback function **pmix\_op\_cbfunc\_t** (function reference) 13 IN cbdata 14 15 Data to be passed to the callback function (memory reference) Returns one of the following: 16 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 function prior to returning from the API. 19 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 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 that

published the info.

26

|                                  |       | ▼ Optional Attributes   |
|----------------------------------|-------|---|
| 1                                |       | The following attributes are optional for host environments that support this operation:  |
| 2<br>3<br>4<br>5                 |       | 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.   |
| 6<br>7                           |       | PMIX_RANGE "pmix.range" (pmix_data_range_t)  Value for calls to publish/lookup/unpublish or for monitoring event notifications.   |
| 8<br>9                           |       | PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)  Value for calls to PMIx_Publish .   |
|                                  |       | Advice to PMIx library implementers   |
| 10<br>11<br>12<br>13<br>14<br>15 |       | We recommend that implementation of the <b>PMIX_TIMEOUT</b> 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 <b>PMIX_TIMEOUT</b> directly in the PMIx server library must take care to resolve the race condition and should avoid passing <b>PMIX_TIMEOUT</b> to the host environment so that multiple competing timeouts are not created. |
| 16<br>17<br>18                   |       | <b>Description</b> Nonblocking <b>PMIx_Publish</b> routine. The non-blocking form will return immediately, executing the callback when the PMIx server receives confirmation from its host SMS daemon.  |
| 19<br>20                         |       | Note that the function will return an error if a <b>NULL</b> callback function is given, and that the <i>info</i> array must be maintained until the callback is provided.  |
| 21 5                             | 5.3.3 | PMIx_Lookup   |
| 22<br>23<br>24                   |       | Summary Lookup information published by this or another process with PMIx_Publish or PMIx_Publish_nb.   |

| 1      | Format   |
|--------|--|
| PMIx v |  |
| 2      | pmix_status_t  |
| 3      | PMIx_Lookup(pmix_pdata_t data[], size_t ndata,   |
| 4      | <pre>const pmix_info_t info[], size_t ninfo)</pre>   |
|        | C —  |
| 5      | INOUT data   |
| 6      | Array of publishable data structures (array of handles)  |
| 7      | IN ndata   |
| 8      | Number of elements in the <i>data</i> array (integer)  |
| 9      | IN info  |
| 0      | Array of info structures (array of handles)  |
| 1      | IN ninfo   |
| 2      | Number of elements in the <i>info</i> array (integer)  |
| 3      | Returns <b>PMIX_SUCCESS</b> or a negative value corresponding to a PMIx error constant.            |
|        | ▼  |
| 4      | PMIx libraries are not required to directly support any attributes for this function. However, any |
| 5      | provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  |
| 6      | required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process that is        |
| 7      | requesting the info.   |
|        | <b>AA</b>  |
|        | ▼ Optional Attributes  |
| 8      | The following attributes are optional for host environments that support this operation:           |
| 9      | PMIX_TIMEOUT "pmix.timeout" (int)  |
| 0      | Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in  |
| 1      | error. The timeout parameter can help avoid "hangs" due to programming errors that prevent         |
| 2      | the target process from ever exposing its data.  |
| 3      | PMIX_RANGE "pmix.range" (pmix_data_range_t)  |
| 4      | Value for calls to publish/lookup/unpublish or for monitoring event notifications.                 |
| 5      | PMIX_WAIT "pmix.wait" (int)  |
| 6      | Caller requests that the PMIx server wait until at least the specified number of values are        |
| 7      | found (0 indicates all and is the default).  |
|        | AA   |

### Advice to PMIx library implementers -

We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

### **Description**

Lookup information published by this or another process. By default, the search will be conducted across the **PMIX\_RANGE\_SESSION** range. Changes to the range, and any additional directives, can be provided in the **pmix\_info\_t** array. Data is returned provided the following conditions are met:

- the requesting process resides within the range specified by the publisher. For example, data
  published to PMIX\_RANGE\_LOCAL can only be discovered by a process executing on the same
  node
- the provided key matches the published key within that data range
- the data was published by a process with corresponding user and/or group IDs as the one looking up the data. There currently is no option to override this behavior such an option may become available later via an appropriate pmix\_info\_t directive.

The *data* parameter consists of an array of <code>pmix\_pdata\_t</code> 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 <code>PMIX\_UNDEF</code>. The function will return <code>PMIX\_SUCCESS</code> 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 2 Nonblocking version of PMIx Lookup. 3 Format PMIx v1.0 5 pmix status t 6 PMIx Lookup nb(char \*\*keys, 7 const pmix\_info\_t info[], size\_t ninfo, pmix lookup cbfunc t cbfunc, void \*cbdata) 8 9 IN keys 10 Array to be provided to the callback (array of strings) IN 11 Array of info structures (array of handles) 12 IN ninfo 13 14 Number of element in the *info* array (integer) 15 IN cbfunc Callback function (handle) 16 IN cbdata 17 Callback data to be provided to the callback function (pointer) 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 22 function prior to returning from the API. 23 • a PMIx error constant indicating an error in the input - the *cbfunc* will *not* be called 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.

24

25

# 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 ( $\theta$  indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.

#### PMIX RANGE "pmix.range" (pmix data range t)

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

### PMIX\_WAIT "pmix.wait" (int)

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

### Advice to PMIx library implementers

We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

### **Description**

Non-blocking form of the PMIx\_Lookup function. Data for the provided NULL-terminated keys array will be returned in the provided callback function. As with PMIx\_Lookup, the default behavior is to not wait for data to be published. The info array can be used to modify the behavior as previously described by PMIx\_Lookup. Both the info and keys arrays must be maintained until the callback is provided.

### 3 5.3.5 PMIx\_Unpublish

### Summary

Unpublish data posted by this process using the given keys.

| 1                                |           | Format   |
|----------------------------------|-----------|--|
|                                  | PMIx v1.0 |  |
| 2                                |           | pmix_status_t  |
| 3                                |           | PMIx_Unpublish(char **keys,  |
| 4                                |           | const pmix_info_t info[], size_t ninfo)  |
|                                  |           | O  |
| 5                                |           | IN info  |
| 6                                |           | Array of info structures (array of handles)  |
| 7<br>8                           |           | Number of element in the <i>info</i> array (integer)   |
|                                  |           |  |
| 9                                |           | Returns <b>PMIX_SUCCESS</b> or a negative value corresponding to a PMIx error constant.  |
|                                  |           | Required Attributes  |
| 10<br>11<br>12<br>13             |           | 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   |
| 14                               |           | The following attributes are optional for host environments that support this operation:   |
| 15<br>16<br>17<br>18             |           | 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.  |
| 19<br>20                         |           | 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 —  |
| 21<br>22<br>23<br>24<br>25<br>26 |           | 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** 1 2 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 3 4 keys parameter instructs the server to remove all data published by this process. 5 By default, the range is assumed to be **PMIX RANGE SESSION**. Changes to the range, and any 6 additional directives, can be provided in the *info* array. 5.3.6 PMIx Unpublish nb **Summary** 8 Nonblocking version of PMIx\_Unpublish. 9 **Format** 10 PMIx v1.011 pmix\_status\_t 12 PMIx Unpublish nb(char \*\*keys, 13 const pmix\_info\_t info[], size\_t ninfo, pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 14 IN 15 keys 16 (array of strings) 17 IN info 18 Array of info structures (array of handles) 19 IN ninfo Number of element in the *info* array (integer) 20 21 IN cbfunc 22 Callback function **pmix\_op\_cbfunc\_t** (function reference) IN 23 cbdata Data to be passed to the callback function (memory reference) 24 25 Returns one of the following: 26 • 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 27 function prior to returning from the API. 28 29 • PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and 30 returned success - the cbfunc will not be called 31

• a PMIx error constant indicating either an error in the input or that the request was immediately

processed and failed - the cbfunc will not be called

32

# 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 ( $\theta$ 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.

1

2

5 6

7

8

9

10

11

12

13

14 15

16

17

18

19

20

#### **CHAPTER 6**

# **Process Management**

1 This chapter defines functionality used by clients to create and destroy/abort processes in the PMIx universe.

### 3 **6.1 Abort**

PMIx provides a dedicated API by which an application can request that specified processes be aborted by the system.

### 6 6.1.1 PMIx Abort

```
Summary
 8
               Abort the specified processes
               Format
   PMIx v1.0
10
               pmix_status_t
               PMIx_Abort(int status, const char msg[],
11
                             pmix_proc_t procs[], size_t nprocs)
12
                                                    — С
               IN
13
                    Error code to return to invoking environment (integer)
14
15
               IN
16
                    String message to be returned to user (string)
               IN
                    procs
17
                    Array of pmix proc t structures (array of handles)
18
               IN
19
                    nprocs
                    Number of elements in the procs array (integer)
20
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
21
```

### Description

1

3

4

5

6

7

8

9

10 11

12

13

14

15

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
- 19 upon provided and supported attributes.

### 20 6.2.1 PMIx Spawn

### 21 Summary

Spawn a new job.

```
Format
 1
   PMIx v1.0
 2
               pmix status t
 3
               PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
                              const pmix app t apps[], size t napps,
 4
 5
                              char nspace[])
               IN
                     job info
 6
 7
                    Array of info structures (array of handles)
 8
               IN
                    ninfo
 9
                    Number of elements in the job info array (integer)
10
               IN
                    apps
                    Array of pmix_app_t structures (array of handles)
11
               IN
                    napps
12
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
               required to add the following attributes to those provided before passing the request to the host:
19
20
               PMIX_SPAWNED "pmix.spawned" (bool)
                     true if this process resulted from a call to PMIx Spawn.
21
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)
                     The requesting process is a PMIx client.
25
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
               processes so those values can be returned to clients upon connection to the PMIx server. In
31
               addition, they are required to support the following attributes when present in either the job_info or
32
               the info array of an element of the apps array:
33
34
               PMIX_WDIR "pmix.wdir" (char*)
```

| I                    | working directory for spawned processes.   |
|----------------------|--|
| 2<br>3<br>4<br>5     | PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)  Set the application's current working directory to the session working directory assigned by the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the session working directory assigned to the provided namespace |
| 6<br>7               | PMIX_PREFIX "pmix.prefix" (char*) Prefix to use for starting spawned processes.  |
| 8<br>9               | <pre>PMIX_HOST "pmix.host" (char*)     Comma-delimited list of hosts to use for spawned processes.</pre>   |
| 10<br>11             | PMIX_HOSTFILE "pmix.hostfile" (char*)  Hostfile to use for spawned processes.  |
|                      | ▼Optional Attributes   |
| 12                   | The following attributes are optional for host environments that support this operation:   |
| 13<br>14             | <pre>PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation.</pre>  |
| 15<br>16             | PMIX_ADD_HOST "pmix.addhost" (char*)  Comma-delimited list of hosts to add to the allocation.  |
| 17<br>18             | PMIX_PRELOAD_BIN "pmix.preloadbin" (bool) Preload binaries onto nodes.   |
| 19<br>20             | <pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)     Comma-delimited list of files to pre-position on nodes.</pre>  |
| 21<br>22             | PMIX_PERSONALITY "pmix.pers" (char*)  Name of personality to use.  |
| 23<br>24<br>25<br>26 | PMIX_MAPPER "pmix.mapper" (char*)  Mapping mechanism to use for placing spawned processes - when accessed using  PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping mechanism used for the provided namespace.   |
| 27<br>28             | PMIX_DISPLAY_MAP "pmix.dispmap" (bool) Display process mapping upon spawn.   |
| 29<br>30             | PMIX_PPR "pmix.ppr" (char*)  Number of processes to spawn on each identified resource.   |
| 31                   | <pre>PMIX_MAPBY "pmix.mapby" (char*)</pre>   |

| 1<br>2<br>3      | Process mapping policy - when accessed using <b>PMIx_Get</b> , use the <b>PMIX_RANK_WILDCARD</b> value for the rank to discover the mapping policy used for the provided namespace  |
|------------------|---|
| 4<br>5<br>6<br>7 | PMIX_RANKBY "pmix.rankby" (char*)  Process ranking policy - when accessed using PMIx_Get, use the  PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the provided namespace                    |
| 8<br>9<br>0<br>1 | <pre>PMIX_BINDTO "pmix.bindto" (char*)     Process binding policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the     provided namespace</pre> |
| 2<br>3           | PMIX_NON_PMI "pmix.nonpmi" (bool) Spawned processes will not call PMIx_Init.  |
| 4<br>5           | PMIX_STDIN_TGT "pmix.stdin" (uint32_t)  Spawned process rank that is to receive stdin.  |
| 6<br>7           | PMIX_FWD_STDIN "pmix.fwd.stdin" (bool)  Forward this process's stdin to the designated process.   |
| 8<br>9           | PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)  Forward stdout from spawned processes to this process.  |
| 20<br>21         | <pre>PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.</pre>  |
| 22<br>23         | PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.  |
| 24<br>25         | PMIX_TAG_OUTPUT "pmix.tagout" (bool)  Tag application output with the identity of the source process.   |
| 26<br>27         | PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.   |
| .8<br>.9         | <pre>PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool) Merge stdout and stderr streams from application processes.</pre>   |
| 30<br>31         | <pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*) Output application output to the specified file.</pre>  |
| 32<br>33         | PMIX_INDEX_ARGV "pmix.indxargv" (bool)  Mark the argv with the rank of the process.   |
| 34               | PMIX_CPUS_PER_PROC "pmix.cpuperproc" (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 provided namespace 3 4 PMIX NO PROCS ON HEAD "pmix.nolocal" (bool) 5 Do not place processes on the head node. PMIX NO OVERSUBSCRIBE "pmix.noover" (bool) 6 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\*) List of cpus to use for this job - when accessed using PMIx\_Get, use the 11 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) Application supports recoverable operations. 15 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) Maximum number of times to restart a job - when accessed using PMIx\_Get, use the 19 PMIX\_RANK\_WILDCARD value for the rank to discover the max restarts for the provided 20 21 namespace 22 PMIX NOTIFY COMPLETION "pmix.notecomp" (bool) Notify the parent process upon termination of child job. 23

### **Description**

Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspace* parameter. A **NULL** value in that location indicates that the caller doesn't wish to have the namespace returned. The *nspace* array must be at least of size one more than **PMIX\_MAX\_NSLEN**.

By default, the spawned processes will be PMIx "connected" to the parent process upon successful launch (see PMIx\_Connect description for details). Note that this only means that (a) the parent process will be given a copy of the new job's information so it can query job-level info without incurring any communication penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level info, and (c) both the parent process and members of the child job will receive notification of errors from processes in their combined assemblage.

24 25

26 27

28

29

30

31

32

### Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

### 4 6.2.2 PMIx\_Spawn\_nb

#### Summary

Nonblocking version of the **PMIx\_Spawn** routine.

#### **Format**

PMIx v1.0

1

2

3

5

8

9 10

11

12

13 14

15

17 18

19

20

21 22

23

24

25 26 С

- IN job\_info
  - Array of info structures (array of handles)
- IN ninfo
  - Number of elements in the *job\_info* array (integer)
- 16 IN apps
  - Array of pmix\_app\_t structures (array of handles)
  - IN cbfunc
    - Callback function pmix\_spawn\_cbfunc\_t (function reference)
  - IN cbdata
  - Data to be passed to the callback function (memory reference)
  - Returns one of the following:
    - PMIX\_SUCCESS, indicating that the request is being processed by the host environment result
      will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
      function prior to returning from the API.
    - a PMIx error constant indicating an error in the request the *cbfunc* will *not* be called

### Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is required to add the following attributes to those provided before passing the request to the host:

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

true if this process resulted from a call to PMIx\_Spawn.

```
PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
```

Process identifier of the parent process of the calling process.

PMIX REQUESTOR IS CLIENT "pmix.req.client" (bool)

The requesting process is a PMIx client.

PMIX\_REQUESTOR\_IS\_TOOL "pmix.req.tool" (bool)

The requesting process is a PMIx tool.

Host environments that implement support for PMIx\_Spawn are required to pass the PMIX\_SPAWNED and PMIX\_PARENT\_ID attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the <code>job\_info</code> or the <code>info</code> array of an element of the <code>apps</code> array:

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

Working directory for spawned processes.

```
PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
```

Set the application's current working directory to the session working directory assigned by the RM - when accessed using <code>PMIx\_Get</code>, use the <code>PMIX\_RANK\_WILDCARD</code> value for the rank to discover the session working directory assigned to the provided namespace

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

Prefix to use for starting spawned processes.

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

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

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

Hostfile to use for spawned processes.

1

2

3

4

5 6

7

8

9

11 12 13

14

15

16

17 18

19 20

21

22 23

24

25 26

27

28

|                      | → Optional Attributes   |
|----------------------|---|
| 1                    | The following attributes are optional for host environments that support this operation:  |
| 2                    | <pre>PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*) Hostfile listing hosts to add to existing allocation.</pre>   |
| 4<br>5               | PMIX_ADD_HOST "pmix.addhost" (char*)  Comma-delimited list of hosts to add to the allocation.   |
| 6<br>7               | PMIX_PRELOAD_BIN "pmix.preloadbin" (bool) Preload binaries onto nodes.  |
| 8<br>9               | <pre>PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)</pre>   |
| 10<br>11             | <pre>PMIX_PERSONALITY "pmix.pers" (char*) Name of personality to use.</pre>   |
| 12<br>13<br>14<br>15 | <pre>PMIX_MAPPER "pmix.mapper" (char*)     Mapping mechanism to use for placing spawned processes - when accessed using     PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping     mechanism used for the provided namespace.</pre> |
| 16<br>17             | PMIX_DISPLAY_MAP "pmix.dispmap" (bool) Display process mapping upon spawn.  |
| 18<br>19             | PMIX_PPR "pmix.ppr" (char*)  Number of processes to spawn on each identified resource.  |
| 20<br>21<br>22<br>23 | <pre>PMIX_MAPBY "pmix.mapby" (char*)     Process mapping policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the     provided namespace</pre>                                       |
| 24<br>25<br>26<br>27 | <pre>PMIX_RANKBY "pmix.rankby" (char*)     Process ranking policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the     provided namespace</pre>                                  |
| 28<br>29<br>30<br>31 | <pre>PMIX_BINDTO "pmix.bindto" (char*)     Process binding policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the     provided namespace</pre>                                     |
| 32<br>33             | PMIX_NON_PMI "pmix.nonpmi" (bool)  Spawned processes will not call PMIx_Init.   |
| 34<br>35             | PMIX_STDIN_TGT "pmix.stdin" (uint32_t) Spawned process rank that is to receive stdin.   |

| 1<br>2               | PMIX_FWD_STDIN "pmix.fwd.stdin" (bool) Forward this process's stdin to the designated process.   |
|----------------------|--|
| 3<br>4               | PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)  Forward stdout from spawned processes to this process.   |
| 5<br>6               | PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.  |
| 7<br>8               | PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.   |
| 9<br>10              | PMIX_TAG_OUTPUT "pmix.tagout" (bool)  Tag application output with the identity of the source process.  |
| 1<br> 2              | PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.  |
| 13<br>14             | PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)  Merge stdout and stderr streams from application processes.  |
| 15<br>16             | <pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*) Output application output to the specified file.</pre>   |
| 17<br>18             | PMIX_INDEX_ARGV "pmix.indxargv" (bool)  Mark the argv with the rank of the process.  |
| 19<br>20<br>21<br>22 | PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)  Number of cpus to assign to each rank - when accessed using PMIx_Get , use the  PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace |
| 23<br>24             | PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)  Do not place processes on the head node.  |
| 25<br>26             | PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)  Do not oversubscribe the cpus.   |
| 27<br>28             | PMIX_REPORT_BINDINGS "pmix.repbind" (bool) Report bindings of the individual processes.  |
| 29<br>80<br>31<br>32 | <pre>PMIX_CPU_LIST "pmix.cpulist" (char*) List of cpus to use for this job - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided namespace</pre>               |
| 33<br>34             | PMIX_JOB_RECOVERABLE "pmix.recover" (bool) Application supports recoverable operations.  |
| 35<br>36             | PMIX_JOB_CONTINUOUS "pmix.continuous" (bool) Application is continuous, all failed processes should be immediately restarted.  |

1 PMIX\_MAX\_RESTARTS "pmix.maxrestarts" (uint32\_t) 2 Maximum number of times to restart a job - when accessed using **PMIx Get**, use the 3 PMIX RANK WILDCARD value for the rank to discover the max restarts for the provided

namespace

#### **Description**

5 6

7

8

9

10

12

13 14

15

16

17

18

19

20 21

22 23

24

25

26 27

28

29

30

Nonblocking version of the PMIx Spawn routine. The provided callback function will be executed upon successful start of all specified application processes.

#### Advice to users

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.

#### 6.3 **Connecting and Disconnecting Processes**

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of *connected* solely implies that the host environment should treat the failure of any process in the assemblage as a reportable event, taking action on the assemblage as if it were a single application. For example, if the environment defaults (in the absence of any application directives) to terminating an application upon failure of any process in that application, then the environment should terminate all processes in the connected assemblage upon failure of any member.

#### — Advice to PMIx server hosts —

The host environment may choose to assign a new namespace to the connected assemblage and/or assign new ranks for its members for its own internal tracking purposes. However, it is not required to communicate such assignments to the participants (e.g., in response to an appropriate call to PMIx Query info nb). The host environment is required to generate a PMIX ERR INVALID TERMINATION event should any process in the assemblage terminate or call **PMIx\_Finalize** without first *disconnecting* from the assemblage.

The *connect* operation does not require the exchange of job-level information nor the inclusion of information posted by participating processes via PMIx\_Put. Indeed, the callback function utilized in pmix\_server\_connect\_fn\_t cannot pass information back into the PMIx server library. However, host environments are advised that collecting such information at the participating daemons represents an optimization opportunity as participating processes are likely to request such information after the connect operation completes.

#### Advice to users -

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation.

While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

Neither the PMIx implementation nor host environment are required to provide any tracking support for the assemblage. Thus, the application is responsible for maintaining the membership list of the assemblage.

### 6.3.1 PMIx Connect

#### Summary

Connect namespaces.

#### **Format**

PMIx v1.0

4

5 6

8

9

10

11

12

13

14

15

16

17

18

19

20

21 22

23

24

pmix\_status\_t

\_\_\_\_\_ C \_

IN procs

Array of proc structures (array of handles)

IN nprocs

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

IN info

Array of info structures (array of handles)

IN ninfo

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

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

#### Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing.

|                            | → Optional Attributes   |
|----------------------------|---|
| 1                          | The following attributes are optional for host environments that support this operation:  |
| 2<br>3<br>4<br>5           | 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.   |
| 6<br>7<br>8<br>9<br>10     | 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.   |
| 11<br>12                   | PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)  If true, indicates that the requested choice of algorithm is mandatory.  |
|                            | Advice to PMIx library implementers —   |
| 13<br>14<br>15<br>16<br>17 | We recommend that implementation of the <b>PMIX_TIMEOUT</b> 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 <b>PMIX_TIMEOUT</b> directly in the PMIx server library must take care to resolve the race condition and should avoid passing <b>PMIX_TIMEOUT</b> to the host environment so that multiple competing timeouts are not created. |

#### Description

 Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The function will return once all processes identified in *procs* have called either **PMIx\_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

#### Advice to users

All processes engaged in a given **PMIx\_Connect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX\_RANK\_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

### Advice to PMIx library implementers

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

#### Advice to PMIx server hosts —

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

Processes that combine via **PMIx\_Connect** must call **PMIx\_Disconnect** prior to finalizing and/or terminating - any process in the assemblage failing to meet this requirement will cause a **PMIX\_ERR\_INVALID\_TERMINATION** event to be generated.

A process can only engage in one connect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple connect operations, each involving a different *procs* array.

As in the case of the **PMIx\_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

### 6.3.2 PMIx\_Connect\_nb

#### Summary

Nonblocking PMIx Connect nb routine.

| 1           | Format  |
|-------------|---|
| <i>PMIx</i> | v1.0 V  |
| 2           | pmix_status_t   |
| 3           | <pre>PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,</pre>                                |
| 4           | <pre>const pmix_info_t info[], size_t ninfo,</pre>  |
| 5           | <pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>   |
|             | C   |
| 6           | IN procs  |
| 7           | Array of proc structures (array of handles)   |
| 8           | IN nprocs   |
| 9           | Number of elements in the <i>procs</i> array (integer)  |
| 10          | <pre>IN info</pre>  |
| 11          | Array of info structures (array of handles)   |
| 12          | IN ninfo  |
| 13          | Number of element in the <i>info</i> 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 <i>cbfunc</i> . 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 <i>cbfunc</i> will <i>not</i> 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

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 ( $\theta$  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.

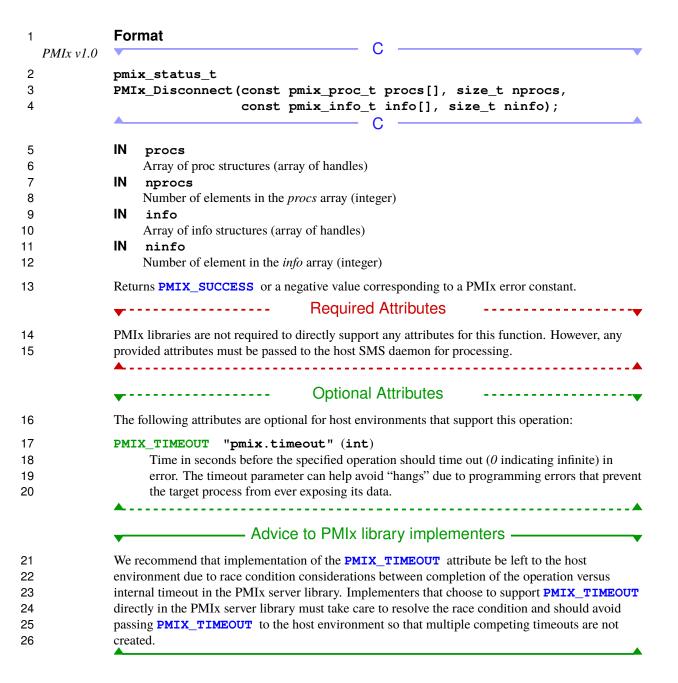
### Description

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.



#### Description

Disconnect a previously connected set of processes. A PMIX\_ERR\_INVALID\_OPERATION error will be returned if the specified set of *procs* was not previously *connected* via a call to PMIx\_Connect or its non-blocking form. The function will return once all processes identified in *procs* have called either PMIx\_Disconnect or its non-blocking version, *and* the host environment has completed any required supporting operations.

#### Advice to users -

All processes engaged in a given **PMIx\_Disconnect** operation must provide the identical *procs* array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX\_RANK\_WILDCARD** versus listing the individual processes) *may* impact the host environment's algorithm for uniquely identifying an operation.

### Advice to PMIx library implementers -

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

### Advice to PMIx server hosts

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

A process can only engage in one disconnect operation involving the identical *procs* array at a time. However, a process can be simultaneously engaged in multiple disconnect operations, each involving a different *procs* array.

As in the case of the **PMIx\_Fence** operation, the *info* array can be used to pass user-level directives regarding the algorithm to be used for any collective operation involved in the operation, timeout constraints, and other options available from the host RM.

### 6.3.4 PMIx\_Disconnect\_nb

#### Summary

Nonblocking PMIx Disconnect routine.

|          |           | Format  |
|----------|-----------|---|
|          | PMIx v1.0 | · · · · · · · · · · · · · · · · · · ·   |
| 2        |           | pmix_status_t   |
| 3        |           | <pre>PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,</pre>   |
| 4        |           | <pre>const pmix_info_t info[], size_t ninfo,</pre>  |
| 5        |           | <pre>pmix_op_cbfunc_t cbfunc, void *cbdata);</pre>  |
|          |           | C —   |
| 6        |           | IN procs  |
| 7        |           | Array of proc structures (array of handles)   |
| 8        |           | IN nprocs   |
| 9        |           | Number of elements in the <i>procs</i> array (integer)  |
| 10       |           | IN info   |
| 11       |           | Array of info structures (array of handles)   |
| 12<br>13 |           | IN ninfo  |
| 13<br>14 |           | Number of element in the <i>info</i> array (integer)  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:   |
| 10       |           |   |
| 19<br>20 |           | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback |
| 21       |           | function prior to returning from the API.   |
|          |           |   |
| 22<br>23 |           | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and<br/>returned success - the cbfunc will not be called</li> </ul>  |
| 23       |           | ·   |
| 24       |           | • a PMIx error constant indicating either an error in the input or that the request was immediately   |
| 25       |           | processed and failed - the <i>cbfunc</i> will <i>not</i> 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.   |
|          |           | <b>A</b>  |
|          |           | ▼ Optional Attributes   |
| 28       |           | The following attributes are optional for host environments that support this operation:  |
|          |           |   |
| 29<br>30 |           | PMIX_TIMEOUT "pmix.timeout" (int)  Time in seconds before the specified operation should time out (0 indicating infinite) in  |
| 30<br>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.

## 2 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

#### 4 Summary

Register to receive output forwarded from a set of remote processes.

#### **Format**

PMIx v3.0

1

5

6

7

8

9 10

11

12

15 16

17 18

19

21 22

23

24

25

26 27

28

29

C

- IN procs
- Array of proc structures identifying desired source processes (array of handles)
- 14 **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)
- 20 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)

1 If regcbfunc is **NULL**, the function call will be treated as a blocking call. In this case, the returned status will be either (a) the IOF handler reference identifier if the value is greater than or equal to 2 zero, or (b) a negative error code indicative of the reason for the failure. 3 4 If the regcbfunc is non-NULL, the function call will be treated as a non-blocking call and will return the following: 5 PMIX SUCCESS indicating that the request has been accepted for processing and the provided 6 callback function will be executed upon completion of the operation. Note that the library 7 must not invoke the callback function prior to returning from the API. The IOF handler 8 9 identifier will be returned in the callback 10 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. 11 Required Attributes \_\_\_\_\_\_ 12 The following attributes are required for PMIx libraries that support IO forwarding: 13 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 14 server is allowed (but not required) to drop all bytes received beyond the max size. 15 16 PMIX IOF DROP OLDEST "pmix.iof.old" (bool) 17 In an overflow situation, drop the oldest bytes to make room in the cache. PMIX IOF DROP\_NEWEST "pmix.iof.new" (bool) 18 In an overflow situation, drop any new bytes received until room becomes available in the 19 cache (default). 20 **▲**----------- Optional Attributes ------21 The following attributes are optional for PMIx libraries that support IO forwarding: 22 PMIX\_IOF\_BUFFERING\_SIZE "pmix.iof.bsize" (uint32\_t) 23 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 24 becomes available. Any remaining buffered data will be "flushed" upon call to deregister the 25 respective channel. 26 27 PMIX\_IOF\_BUFFERING\_TIME "pmix.iof.btime" (uint32\_t) 28 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering 29 size, this prevents IO from being held indefinitely while waiting for another payload to arrive. 30 31 PMIX\_IOF\_TAG\_OUTPUT "pmix.iof.tag" (bool) 32 Tag output with the channel it comes from. 33 PMIX\_IOF\_TIMESTAMP\_OUTPUT "pmix.iof.ts" (bool)

```
Timestamp output
 1
               PMIX IOF XML OUTPUT "pmix.iof.xml" (bool)
2
 3
                     Format output in XML
               Description
4
               Register to receive output forwarded from a set of remote processes.
5
                                                Advice to users
               Providing a NULL function pointer for the cbfunc parameter will cause output for the indicated
6
 7
               channels to be written to their corresponding stdout/stderr file descriptors. Use of
8
               PMIX RANK WILDCARD to specify all processes in a given namespace is supported but should
               be used carefully due to bandwidth considerations.
9
10 6.4.2
             PMIx_IOF_deregister
               Summary
11
12
               Deregister from output forwarded from a set of remote processes.
               Format
13
   PMIx v3.0
14
               pmix status t
               PMIx IOF deregister(size t iofhdlr,
15
                                         const pmix_info_t directives[], size_t ndirs,
16
                                         pmix_op_cbfunc_t cbfunc, void *cbdata)
17
               IN
                    iofhdlr
18
                   Registration number returned from the pmix hdlr req cbfunc t callback from the
19
20
                   call to PMIx IOF pull (size t)
                    directives
               IN
21
22
                   Array of pmix info t structures (array of handles)
23
               IN
                   Number of elements in the directives array (integer)
24
25
               IN
                    cbfunc
                   Callback function to be called when deregistration has been completed. (function reference)
26
27
               IN
                    cbdata
28
                   Data to be passed to the cbfunc callback function (memory reference)
```

If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation 1 returned in the status code. 2 3 If cbfunc is non-NULL, the function will be treated as a non-blocking call and return one of the 4 following: 5 • PMIX\_SUCCESS, indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning 6 from the API. 7 8 • PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and 9 returned success - the cbfunc will not be called • a PMIx error constant indicating either an error in the input or that the request was immediately 10 processed and failed - the cbfunc will not be called 11 The returned status code will be one of the following: 12 13 **PMIX\_SUCCESS** The IOF handler was successfully deregistered. PMIX\_ERR\_BAD\_PARAM The provided *iofhdlr* was unrecognized. 14 PMIX\_ERR\_NOT\_SUPPORTED The PMIx implementation does not support this function. 15 **Description** 16 17 Deregister from output forwarded from a set of remote processes. Advice to PMIx library implementers 18 Any currently buffered IO should be flushed upon receipt of a deregistration request. All received

### 20 6.4.3 PMIx\_IOF\_push

### 21 Summary

19

22

Push data collected locally (typically from stdin or a file) to stdin of the target recipients.

IO after receipt of the request shall be discarded.

| 1        |           | Format  |
|----------|-----------|---|
| 1        | PMIx v3.0 |   |
| 2        |           | pmix_status_t   |
| 3        |           | <pre>PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,</pre>  |
| 4        |           | <pre>pmix_byte_object_t *bo,</pre>  |
| 5        |           | <pre>const pmix_info_t directives[], size_t ndirs,</pre>  |
| 6        |           | <pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>   |
|          |           | C -   |
| 7        |           | IN targets  |
| 8        |           | Array of proc structures identifying desired target processes (array of handles)  |
| 9        |           | IN ntargets   |
| 0        |           | Number of elements in the <i>targets</i> array (integer)  |
| 1        |           | IN bo   |
| 2        |           | Pointer to pmix_byte_object_t containing the payload to be delivered (handle)   |
| 3        |           | IN directives   |
| 4        |           | Array of pmix_info_t structures (array of handles)  |
| 5        |           | IN ndirs  Number of elements in the directives arroy (integer)  |
| 6<br>7   |           | Number of elements in the <i>directives</i> array (integer)  IN directives  |
| 8        |           | Array of pmix_info_t structures (array of handles)  |
| 9        |           | IN cbfunc   |
| 20       |           | Callback function to be called when operation has been completed. (pmix_op_cbfunc_t   |
| 21       |           | function reference)   |
| 2        |           | IN cbdata   |
| 23       |           | Data to be passed to the <i>cbfunc</i> callback function (memory reference)   |
| 24       |           | If <i>cbfunc</i> is <b>NULL</b> , the function will be treated as a <i>blocking</i> call and the result of the operation  |
| 25       |           | returned in the status code.  |
| 26       |           | If <i>cbfunc</i> is non- <b>NULL</b> , the function will be treated as a <i>non-blocking</i> call and return one of the   |
| 27       |           | following:  |
| 28       |           | • PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the   |
| 9        |           | provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning   |
| 0        |           | from the API.   |
| 31       |           | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and   |
| 2        |           | returned success - the cbfunc will not be called  |
| 3        |           | • a PMIx error constant indicating either an error in the input or that the request was immediately   |
| 34       |           | processed and failed - the <i>cbfunc</i> will <i>not</i> be called  |
|          |           |   |
| 15       |           | The returned status code will be one of the following:  |
| 86<br>87 |           | <b>PMIX_SUCCESS</b> The provided data has been accepted for transmission - it is not indicative of the payload being delivered to any member of the provided <i>targets</i> |

**PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function. 1 a PMIx error constant indicating the nature of the error 2 Required Attributes The following attributes are required for PMIx libraries that support IO forwarding: 3 PMIX IOF CACHE SIZE "pmix.iof.csize" (uint32 t) 4 5 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. 6 PMIX IOF DROP OLDEST "pmix.iof.old" (bool) 7 8 In an overflow situation, drop the oldest bytes to make room in the cache. 9 PMIX IOF DROP NEWEST "pmix.iof.new" (bool) In an overflow situation, drop any new bytes received until room becomes available in the 10 cache (default). 11 ------ Optional Attributes -------The following attributes are optional for PMIx libraries that support IO forwarding: 12 13 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 14 IO arrives. The library will execute the callback whenever the specified number of bytes 15 becomes available. Any remaining buffered data will be "flushed" upon call to deregister the 16 respective channel. 17 18 PMIX IOF BUFFERING TIME "pmix.iof.btime" (uint32 t) 19 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 20 21 22 **Description** Push data collected locally (typically from stdin or a file) to stdin of the target recipients. 23 Advice to users -24 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. 25 Use of PMIX RANK WILDCARD to specify all processes in a given namespace is supported but 26 27 should be used carefully due to bandwidth considerations.

#### **CHAPTER 7**

## **Job Management and Reporting**

The job management APIs provide an application with the ability to orchestrate its operation in partnership with the SMS. Members of this category include the

PMIx\_Allocation\_request\_nb, PMIx\_Job\_control\_nb, and

PMIx Process monitor nb APIs.

### 7.1 Query

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

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

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

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

### 7.1.1 PMIx\_Resolve\_peers

#### Summary

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

```
Format
1
   PMIx v1.0
2
               pmix status t
 3
               PMIx Resolve peers (const char *nodename,
                                        const pmix nspace t nspace,
 4
                                        pmix_proc_t **procs, size_t *nprocs)
5
               IN
                    nodename
6
 7
                    Name of the node to query (string)
8
               IN
                   nspace
9
                    namespace (string)
10
               OUT procs
                    Array of process structures (array of handles)
11
               OUT nprocs
12
13
                    Number of elements in the procs array (integer)
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
14
               Description
15
               Given a nodename, return the array of processes within the specified nspace that are executing on
16
               that node. If the nspace is NULL, then all processes on the node will be returned. If the specified
17
               node does not currently host any processes, then the returned array will be NULL, and nprocs will
18
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.
    7.1.2
              PMIx Resolve nodes
               Summary
22
               Return a list of nodes hosting processes within the given namespace.
23
               Format
24
   PMIx v1.0
25
               pmix_status_t
               PMIx_Resolve_nodes(const char *nspace, char **nodelist)
26
27
               IN
                    nspace
                    Namespace (string)
28
29
               OUT nodelist
30
                    Comma-delimited list of nodenames (string)
31
               Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

```
Description
1
2
               Given a nspace, return the list of nodes hosting processes within that namespace. The returned
               string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the
 3
 4
               string when done with it.
    7.1.3
             PMIx Query info
               Summary
6
 7
               Query information about the system in general.
               Format
8
   PMIx v4.0
9
               pmix status t
10
               PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
                                    pmix_info_t *info[], size_t *ninfo)
11
                                                —— C
12
               IN
                    queries
13
                    Array of query structures (array of handles)
               IN
                    nqueries
14
                    Number of elements in the queries array (integer)
15
               INOUT info
16
17
                    Address where a pointer to an array of pmix_info_t containing the results of the query
                    can be returned (memory reference)
18
               INOUT ninfo
19
                    Address where the number of elements in info can be returned (handle)
20
21
               Returns one of the following:
22
               • PMIX SUCCESS All data has been returned
               • PMIX_ERR_NOT_FOUND None of the requested data was available
23
24
               • PMIX ERR PARTIAL SUCCESS Some of the data has been returned
               • PMIX_ERR_NOT_SUPPORTED The host RM does not support this function
25
26
               • 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:
27
28
               PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)
                     Retrieve updated information from server.
29
30
               PMIX_SESSION_INFO "pmix.ssn.info" (bool)
```

1 2 3 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.

4 5

#### PMIX JOB INFO "pmix.job.info" (bool)

6 7

8 9

11 12

10

13 14 15

16 17

18 19 20

21 22 23

24

25 26 27

28 29

30 31 32

33 34

35 36

37

38

39

Return information about the specified job or namespace. If information about a job or

namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX JOBID or PMIX NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

#### PMIX APP INFO "pmix.app.info" (bool)

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a PMIX APPNUM attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

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

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the PMIX\_NODEID or PMIX\_HOSTNAME attribute identifying the desired target.

#### PMIX\_PROCID "pmix.procid" (pmix\_proc\_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the PMIX\_LOCAL\_RANK of a specified process. Only required when the request is for information on a specific process.

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

Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the PMIX\_LOCAL\_RANK of a specified process. Must be accompanied by the PMIX\_RANK attribute. Only required when the request is for information on a specific process.

#### PMIX\_RANK "pmix.rank" (pmix\_rank\_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the PMIX\_LOCAL\_RANK of a specified process. Must be accompanied by the **PMIX\_NSPACE** attribute. Only required when the request is for information on a specific process.

```
PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
     Query list of supported attributes for specified APIs
```

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

```
PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
```

Request attributes supported by the PMIx server library

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

 Request attributes supported by the host environment

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

Request attributes supported by the PMIx tool library functions

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

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

Host environments that support this operation are required to support the following attributes as qualifiers to the request:

#### PMIX\_PROCID "pmix.procid" (pmix\_proc\_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Only required when the request is for information on a specific process.

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

Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_RANK** attribute. Only required when the request is for information on a specific process.

#### PMIX\_RANK "pmix.rank" (pmix\_rank\_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_NSPACE** attribute. Only required when the request is for information on a specific process.

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

|                      | → Optional Attributes  |
|----------------------|--|
| 1                    | The following attributes are optional for host environments that support this operation:   |
| 2<br>3               | <pre>PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*) Request a comma-delimited list of active namespaces.</pre>  |
| 4<br>5               | <pre>PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job.</pre>  |
| 6<br>7               | <pre>PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues.</pre>   |
| 8<br>9               | <pre>PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD) Status of a specified scheduler queue.</pre>   |
| 10<br>11<br>12       | <pre>PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns (     pmix_data_array_t) an array of pmix_proc_info_t.</pre>  |
| 13<br>14<br>15<br>16 | <pre>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.</pre> |
| 17<br>18             | PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)  Return a comma-delimited list of supported spawn attributes.   |
| 19<br>20             | <pre>PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool) Return a comma-delimited list of supported debug attributes.</pre>   |
| 21<br>22             | <pre>PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool) Return information on memory usage for the processes indicated in the qualifiers.</pre>   |
| 23<br>24             | <pre>PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool) Report only average values for sampled information.</pre>   |
| 25<br>26             | <pre>PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values.</pre>  |
| 27<br>28             | <pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*) String identifier of the allocation whose status is being requested.</pre>   |
| 29<br>30<br>31       | <pre>PMIX_TIME_REMAINING "pmix.time.remaining" (char*)</pre>   |
| 32<br>33<br>34       | PMIX_SERVER_URI "pmix.srvr.uri" (char*)  URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.                    |

| 1<br>2<br>3<br>4 | PMIX_PROC_URI "pmix.puri" (char*)  URI containing contact information for a given process. Requests the URI of the specified PMIx server's out-of-band connection. Defaults to requesting the information for the local PMIx server. |
|------------------|--|
| 5                | Description  |
| 6                | Query information about the system in general. This can include a list of active namespaces, fabric  |
| 7                | topology, etc. Also can be used to query node-specific info such as the list of peers executing on a   |
| 8                | given node. We assume that the host RM will exercise appropriate access control on the   |
| 9                | information.   |
| 10               | The returned <i>status</i> indicates if requested data was found or not. The returned array of   |
| 11               | pmix_info_t will contain each key that was provided and the corresponding value that was   |
| 12               | found. Requests for keys that are not found will return the key paired with a value of type  |
| 13               | <b>PMIX_UNDEF</b> . The caller is responsible for releasing the returned array.  |
|                  | Advice to PMIx library implementers  |
| I.A              | Information returned from PMTy, Query, info shall be locally cached so that retrieval by   |

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

### 7.1.4 PMIx\_Query\_info\_nb

### 21 Summary

15

16

17

18

19

22

Query information about the system in general.

| 1                | Format  |
|------------------|---|
| <i>PMIx v2.0</i> |   |
| 2                | pmix_status_t   |
| 3                | PMIx_Query_info_nb(pmix_query_t queries[], size_t nqueries,   |
| 4                | pmix_info_cbfunc_t cbfunc, void *cbdata)  |
|                  | <u> </u>  |
| 5                | IN queries  |
| 6                | Array of query structures (array of handles)  |
| 7<br>8           | IN nqueries  Number of elements in the <i>queries</i> array (integer)   |
| 9                | IN cbfunc   |
| 10               | Callback function pmix_info_cbfunc_t (function reference)   |
| 11               | IN cbdata   |
| 12               | Data to be passed to the callback function (memory reference)   |
| 13               | Returns one of the following:   |
| 14               | • PMIX_SUCCESS indicating that the request has been accepted for processing and the provided  |
| 15               | callback function will be executed upon completion of the operation. Note that the library must   |
| 16               | not invoke the callback function prior to returning from the API.   |
| 17<br>18         | • 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 |
|                  |   |
| 19<br>20         | If executed, the status returned in the provided callback function will be one of the following constants:  |
| 21               | • PMIX_SUCCESS All data has been returned   |
| 22               | • PMIX_ERR_NOT_FOUND None of the requested data was available   |
| 23               | • PMIX_ERR_PARTIAL_SUCCESS Some of the data has been returned   |
| 24               | • PMIX_ERR_NOT_SUPPORTED The host RM does not support this function   |
| 25               | • a non-zero PMIx error constant indicating a reason for the request's failure  |
|                  | ▼ Required Attributes   |
| 26               | PMIx libraries that support this API are required to support the following attributes:  |
| 27               | <pre>PMIX_QUERY_REFRESH_CACHE "pmix.qry.rfsh" (bool)</pre>  |
| 28               | Retrieve updated information from server.   |
| 29               | PMIX_SESSION_INFO "pmix.ssn.info" (bool)  |
| 30               | Return information about the specified session. If information about a session other than the   |
| 31               | one containing the requesting process is desired, then the attribute array must contain a   |
| 32               | <b>PMIX_SESSION_ID</b> attribute identifying the desired target.  |

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

Return information about the specified job or namespace. If information about a job or namespace other than the one containing the requesting process is desired, then the attribute array must contain a PMIX\_JOBID or PMIX\_NSPACE attribute identifying the desired target. Similarly, if information is requested about a job or namespace in a session other than the one containing the requesting process, then an attribute identifying the target session must be provided.

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

Return information about the specified application. If information about an application other than the one containing the requesting process is desired, then the attribute array must contain a <code>PMIX\_APPNUM</code> attribute identifying the desired target. Similarly, if information is requested about an application in a job or session other than the one containing the requesting process, then attributes identifying the target job and/or session must be provided.

#### PMIX NODE INFO "pmix.node.info" (bool)

Return information about the specified node. If information about a node other than the one containing the requesting process is desired, then the attribute array must contain either the **PMIX\_NODEID** or **PMIX\_HOSTNAME** attribute identifying the desired target.

#### PMIX\_PROCID "pmix.procid" (pmix\_proc\_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Only required when the request is for information on a specific process.

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

Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_RANK** attribute. Only required when the request is for information on a specific process.

#### PMIX\_RANK "pmix.rank" (pmix\_rank\_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_NSPACE** attribute. Only required when the request is for information on a specific process.

```
PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
    Query list of supported attributes for specified APIs
```

# PMIX\_CLIENT\_ATTRIBUTES "pmix.client.attrs" (bool) Request attributes supported by the PMIx client library

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

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

Request attributes supported by the host environment

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

Request attributes supported by the PMIx tool library functions

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

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

Host environments that support this operation are required to support the following attributes as qualifiers to the request:

#### PMIX\_PROCID "pmix.procid" (pmix\_proc\_t)

Process identifier Specifies the process ID whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Only required when the request is for information on a specific process.

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

Namespace of the job. Specifies the namespace of the process whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_RANK** attribute. Only required when the request is for information on a specific process.

#### PMIX\_RANK "pmix.rank" (pmix\_rank\_t)

Process rank within the job. Specifies the rank of the process whose information is being requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_NSPACE** attribute. Only required when the request is for information on a specific process.

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

### Optional Attributes

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

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

Request a comma-delimited list of active namespaces.

| 1<br>2           | <pre>PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job.</pre>  |
|------------------|--|
| 3<br>4           | PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues.  |
| 5<br>6           | PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD) Status of a specified scheduler queue.  |
| 7<br>8<br>9      | PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)  Input namespace of the job whose information is being requested returns ( pmix_data_array_t) an array of pmix_proc_info_t.  |
| 0<br>1<br>2<br>3 | 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.                  |
| 4<br>5           | PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool) Return a comma-delimited list of supported spawn attributes.  |
| 6<br>7           | PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool) Return a comma-delimited list of supported debug attributes.  |
| 8<br>9           | PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)  Return information on memory usage for the processes indicated in the qualifiers.   |
| 20<br>21         | PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)  Report only average values for sampled information.   |
| 22<br>23         | PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values.   |
| 24<br>25         | <pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)     String identifier of the allocation whose status is being requested.</pre>   |
| 26<br>27<br>28   | PMIX_TIME_REMAINING "pmix.time.remaining" (char*)  Query number of seconds (uint32_t) remaining in allocation for the specified namespace.   |
| 9<br>80<br>81    | PMIX_SERVER_URI "pmix.srvr.uri" (char*)  URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.                  |
| 33<br>34<br>35   | PMIX_PROC_URI "pmix.puri" (char*)  URI containing contact information for a given process. Requests the URI of the specified PMIx server's out-of-band connection. Defaults to requesting the information for the local PMIx server. |
|                  |  |

#### Description

Non-blocking form of the PMIx Query info API

### 7.1.4.1 Using PMIx\_Get vs PMIx\_Query\_info

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

- information provided by the host environment at time of job start. This includes information on the number of processes in the job, their location, and possibly their communication endpoints
- information posted by processes via the PMIx Put function

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

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

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

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

### 7.1.4.2 Accessing attribute support information

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

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

The desired levels (see 3.4.33) of attribute support are provided as qualifiers. Multiple levels can be requested simultaneously by simply adding elements to the *qualifiers* array. Each qualifier should contain the desired level attribute with the boolean value set to indicate whether or not that level is to be included in the returned information. Failure to provide any levels is equivalent to a request for all levels.

Unlike other queries, queries for attribute support can result in the number of returned <code>pmix\_info\_t</code> 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 <code>key</code> is the function and the <code>value</code> contains a <code>pmix\_data\_array\_t</code> of <code>pmix\_info\_t</code>. Each element of the array is marked by a <code>key</code> indicating the requested attribute <code>level</code> with a <code>value</code> composed of a <code>pmix\_data\_array\_t</code> of <code>pmix\_regattr\_t</code>, each describing a supported attribute for that function, as illustrated in Fig. 7.1 below where the requestor asked for supported attributes of <code>PMIx\_Get</code> at the <code>client</code> and <code>server</code> levels, plus attributes of <code>PMIx\_Allocation\_request</code> at all levels:

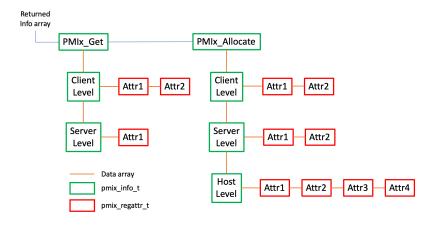


Figure 7.1.: Returned information hierarchy for attribute support request

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

### 7.2 Allocation Requests

This section defines functionality to request new allocations from the RM, and request modifications to existing allocations. These are primarily used in the following scenarios:

- Evolving applications that dynamically request and return resources as they execute
  - *Malleable* environments where the scheduler redirects resources away from executing applications for higher priority jobs or load balancing
    - Resilient applications that need to request replacement resources in the face of failures
    - *Rigid* jobs where the user has requested a static allocation of resources for a fixed period of time, but realizes that they underestimated their required time while executing
    - PMIx attempts to address this range of use-cases with a flexible API.

### 7.2.1 PMIx Allocation request

#### Summary

Request an allocation operation from the host resource manager.

### Format

*PMIx v3.0* 

2

4 5

6 7

9

10

11

12

13

14

15

16

17

18

19 20

21 22

23

24 25

26 27

28

29

#### IN directive

Allocation directive (handle)

#### IN info

Array of pmix info t structures (array of handles)

#### IN ninfo

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

#### **INOUT** results

Address where a pointer to an array of **pmix\_info\_t** containing the results of the request can be returned (memory reference)

#### INOUT nresults

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

#### Returns one of the following:

- PMIX SUCCESS, indicating that the request was processed and returned success
- a PMIx error constant indicating either an error in the input or that the request was refused

| requi        | ded attributes must be passed to the host SMS daemon for processing, and the PMIx library to add the PMIX_USERID and the PMIX_GRPID attributes of the client process measurest.  |
|--------------|--|
|              | environments that implement support for this operation are required to support the follow<br>outes:  |
| PMI:         | K_ALLOC_REQ_ID "pmix.alloc.reqid" (char*) User-provided string identifier for this allocation request which can later be used to que status of the request.  |
| PMI:         | <pre>K_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t) The number of nodes.</pre>   |
| PMI:         | <pre>X_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t) Number of cpus.</pre>  |
| PMI:         | K_ALLOC_TIME "pmix.alloc.time" (uint32_t) Time in seconds.   |
| <b>~</b> - · | Optional Attributes  |
| The          | following attributes are optional for host environments that support this operation:   |
| PMI:         | <pre>K_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*) Regular expression of the specific nodes.</pre>  |
| PMI:         | <pre>K_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*) Regular expression of the number of cpus for each node.</pre>  |
| PMI:         | <pre>K_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*) Regular expression of the specific cpus indicating the cpus involved.</pre>   |
| PMI:         | <pre>X_ALLOC_MEM_SIZE "pmix.alloc.msize" (float) Number of Megabytes.</pre>  |
| PMI:         | <pre>K_ALLOC_FABRIC "pmix.alloc.net" (array) Array of pmix_info_t describing requested fabric resources. This must include at pmix_alloc_fabric_id, pmix_alloc_fabric_type, and PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.</pre> |
| DATE         | <pre>K_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)</pre>  |

```
2
                    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
3
4
                    type of the included value depends upon the fabric support. For example, a TCP allocation
5
                    might consist of a comma-delimited string of socket ranges such as
6
                    "32000-32100,33005,38123-38146". Additional entries will consist of any provided
7
                    resource request directives, along with their assigned values. Examples include:
8
                    PMIX ALLOC FABRIC TYPE - the type of resources provided:
9
                    PMIX ALLOC FABRIC PLANE - if applicable, what plane the resources were assigned
10
                    from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
                    the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
11
                    requested fabric allocation. NOTE: the assigned values may differ from those requested,
12
13
                    especially if PMIX INFO REOD was not set in the request.
               PMIX ALLOC BANDWIDTH "pmix.alloc.bw" (float)
14
15
                    Mbits/sec.
16
               PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
17
                    Quality of service level.
18
               PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
                    Type of desired transport (e.g., "tcp", "udp")
19
20
               PMIX ALLOC FABRIC PLANE "pmix.alloc.netplane" (char*)
                    ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)
21
               PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
22
                    Number of endpoints to allocate per process
23
               PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size t)
24
25
                    Number of endpoints to allocate per node
26
               PMIX ALLOC FABRIC SEC KEY "pmix.alloc.nsec" (pmix byte object t)
27
                    Fabric security key
```

The key to be used when accessing this requested fabric allocation. The allocation will be

### **Description**

Request an allocation operation from the host resource manager. Several broad categories are envisioned, including the ability to:

• Request allocation of additional resources, including memory, bandwidth, and compute. This should be accomplished in a non-blocking manner so that the application can continue to progress while waiting for resources to become available. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.

28 29

30

31

32

33 34

35

- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities. This includes extending the time limit on current resources, and/or requesting additional resources be allocated to the requesting job. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
  - Return no-longer-required resources to the scheduler. This includes the "loan" of resources back to the scheduler with a promise to return them upon subsequent request.

If successful, the returned results for a request for additional resources must include the host resource manager's identifier ( PMIX\_ALLOC\_ID ) that the requester can use to specify the resources in, for example, a call to PMIx\_Spawn .

### 7.2.2 PMIx Allocation request nb

#### Summary

Request an allocation operation from the host resource manager.

### Format

PMIx v2.0

1

2

3

5

6

7

8

9

11 12

13

16

17

18

19

20 21

22

23

24

25

26 27

28

29

30 31

32

33 34

35

- (

C

#### IN directive

Allocation directive (handle)

IN info

Array of pmix info t structures (array of handles)

IN ninfo

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

IN cbfunc

Callback function **pmix info cbfunc t** (function reference)

IN cbdata

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

Returns one of the following:

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

|                      | ▼ Required Attributes  |
|----------------------|--|
| 1<br>2<br>3<br>4     | 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. |
| 6                    | Host environments that implement support for this operation are required to support the following attributes:  |
| 8<br>9<br>10         | <pre>PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*) User-provided string identifier for this allocation request which can later be used to query status of the request.</pre>  |
| 11<br>12             | <pre>PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t) The number of nodes.</pre>  |
| 13<br>14             | <pre>PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t) Number of cpus.</pre>   |
| 15<br>16             | PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t) Time in seconds.  |
|                      | ▼ Optional Attributes  |
| 17                   | The following attributes are optional for host environments that support this operation:   |
| 18<br>19             | <pre>PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*) Regular expression of the specific nodes.</pre>   |
| 20<br>21             | <pre>PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*) Regular expression of the number of cpus for each node.</pre>   |
| 22<br>23             | <pre>PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*) Regular expression of the specific cpus indicating the cpus involved.</pre>  |
| 24<br>25             | PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float) Number of Megabytes.  |
| 26<br>27<br>28<br>29 | PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)  Array of pmix_info_t describing requested fabric resources. This must include at least:  PMIX_ALLOC_FABRIC_ID , PMIX_ALLOC_FABRIC_TYPE , and  PMIX_ALLOC_FABRIC_ENDPTS , plus whatever other descriptors are desired.  |
| 30                   | <pre>PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)</pre>   |

|      | returned/stored as a <b>pmix_data_array_t</b> of <b>pmix_info_t</b> 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 fabric 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_FABRIC_TYPE - the type of resources provided;  PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if PMIX_INFO_REQD was not set in the request. |
| PMIX | ALLOC_BANDWIDTH "pmix.alloc.bw" (float) Mbits/sec.   |
| PMIX | <pre>C_ALLOC_FABRIC_QOS</pre>  |
| PMIX | Type of desired transport (e.g., "tcp", "udp")   |
| PMIX | ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)  ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)   |
| PMIX | ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t) Number of endpoints to allocate per process   |
| PMIX | ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)  Number of endpoints to allocate per node   |
| PMIX | C_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t) Fabric security key  |

Non-blocking form of the **PMIx\_Allocation\_request** API.

# 7.3 Job Control

 This section defines APIs that enable the application and host environment to coordinate the response to failures and other events. This can include requesting termination of the entire job or a subset of processes within a job, but can also be used in combination with other PMIx capabilities (e.g., allocation support and event notification) for more nuanced responses. For example, an application notified of an incipient over-temperature condition on a node could use the PMIx\_Allocation\_request\_nb interface to request replacement nodes while simultaneously using the PMIx\_Job\_control\_nb interface to direct that a checkpoint event be delivered to all processes in the application. If replacement resources are not available, the application might use the PMIx\_Job\_control\_nb interface to request that the job continue at a lower power setting, perhaps sufficient to avoid the over-temperature failure.

The job control APIs can also be used by an application to register itself as available for preemption when operating in an environment such as a cloud or where incentives, financial or otherwise, are provided to jobs willing to be preempted. Registration can include attributes indicating how many resources are being offered for preemption (e.g., all or only some portion), whether the application will require time to prepare for preemption, etc. Jobs that request a warning will receive an event notifying them of an impending preemption (possibly including information as to the resources that will be taken away, how much time the application will be given prior to being preempted, whether the preemption will be a suspension or full termination, etc.) so they have an opportunity to save their work. Once the application is ready, it calls the provided event completion callback function to indicate that the SMS is free to suspend or terminate it, and can include directives regarding any desired restart.

# 7.3.1 PMIx\_Job\_control

```
Summary
24
             Request a job control action.
25
             Format
26
   PMIx v3.0
27
             pmix status t
             PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
28
                                 const pmix_info_t directives[], size_t ndirs,
29
                                 pmix_info_t *results[], size_t *nresults)
30
                                                – C —
31
             IN
                  targets
                  Array of proc structures (array of handles)
32
33
             IN
                  ntargets
                  Number of element in the targets array (integer)
34
35
             IN
                  directives
```

Array of info structures (array of handles)

| 1        | IN ndirs   |
|----------|--|
| 2        | Number of element in the <i>directives</i> array (integer)  INOUT results  |
| 4        | Address where a pointer to an array of pmix_info_t containing the results of the request   |
| 5        | can be returned (memory reference)   |
| 6        | INOUT nresults   |
| 7        | Address where the number of elements in <i>results</i> can be returned (handle)  |
| 8        | Returns one of the following:  |
| 9<br>10  | • PMIX_SUCCESS, indicating that the request was processed by the host environment and returned <i>success</i> . Details of the result will be returned in the <i>results</i> array |
| 11       | • a PMIx error constant indicating either an error in the input or that the request was refused  |
|          | ▼ Required Attributes  |
| 12       | PMIx libraries are not required to directly support any attributes for this function. However, any   |
| 13       | provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  |
| 14<br>15 | required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making the request.  |
|          | the request.   |
| 16       |  |
| 17<br>18 | Host environments that implement support for this operation are required to support the following attributes:  |
| 19       | PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)   |
| 20       | Provide a string identifier for this request. The user can provide an identifier for the   |
| 21       | requested operation, thus allowing them to later request status of the operation or to   |
| 22       | terminate it. The host, therefore, shall track it with the request for future reference.   |
| 23       | PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)  |
| 24       | Pause the specified processes.   |
| 25       | PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)  |
| 26       | Resume ("un-pause") the specified processes.   |
| 27       | PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)  |
| 28       | Forcibly terminate the specified processes and cleanup.  |
| 29       | PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)  |
| 30       | Send given signal to specified processes.  |
| 31       | PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)   |
| 32       | Politely terminate the specified processes.  |
| 33       | PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)   |
| 34       | Comma-delimited list of files to be removed upon process termination   |
| 35       | PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)  |

```
Comma-delimited list of directories to be removed upon process termination
 1
              PMIX CLEANUP RECURSIVE "pmix.clnup.recurse" (bool)
 2
                   Recursively cleanup all subdirectories under the specified one(s)
 3
              PMIX CLEANUP EMPTY "pmix.clnup.empty" (bool)
 4
 5
                   Only remove empty subdirectories
6
              PMIX CLEANUP IGNORE "pmix.clnup.ignore" (char*)
                   Comma-delimited list of filenames that are not to be removed
 7
              PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
8
9
                   When recursively cleaning subdirectories, do not remove the top-level directory (the one
10
                   given in the cleanup request)
              ▲------
                                            Optional Attributes
11
              The following attributes are optional for host environments that support this operation:
              PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
12
                   Cancel the specified request - the provided request ID must match the
13
                   PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
14
15
                   NULL implies cancel all requests from this requestor.
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.
              PMIX JOB CTRL CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
20
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.
              PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
24
25
                   Time in seconds to wait for a checkpoint to complete.
26
              PMIX JOB CTRL CHECKPOINT METHOD
              "pmix.jctrl.ckmethod" (pmix_data_array_t)
27
                   Array of pmix_info_t declaring each method and value supported by this application.
28
29
              PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
                   Regular expression identifying nodes that are to be provisioned.
30
              PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
31
                   Name of the image that is to be provisioned.
32
33
              PMIX JOB CTRL PREEMPTIBLE "pmix.jctrl.preempt" (bool)
```

| 1               |           | <b>A</b> . | Indicate that the job can be pre-empted.   |
|-----------------|-----------|------------|--|
| 2               |           | Des        | scription  |
| 3               |           |            | uest a job control action. The <i>targets</i> array identifies the processes to which the requested job  |
| 4               |           | cont       | rol action is to be applied. A <b>NULL</b> value can be used to indicate all processes in the caller's   |
| 5<br>6          |           |            | espace. The use of <b>PMIX_RANK_WILDCARD</b> can also be used to indicate that all processes in given namespace are to be included.  |
|                 |           |            | •  |
| 7<br>8          |           |            | directives are provided as <b>pmix_info_t</b> structures in the <i>directives</i> array. The callback stion provides a <i>status</i> to indicate whether or not the request was granted, and to provide some |
| 9               |           |            | rmation as to the reason for any denial in the <b>pmix_info_cbfunc_t</b> array of  |
| 10              |           |            | x_info_t structures.   |
| 11              | 7.3.2     | PM1        | [x_Job_control_nb  |
|                 |           |            |  |
| 12              |           |            | mmary  |
| 13              |           | Req        | uest a job control action.   |
| 14              |           | For        | rmat   |
|                 | PMIx v2.0 |            | C  |
| 15              |           | nmi        | .x_status_t  |
| 16              |           | -          | <pre>x_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,</pre>  |
| 17              |           |            | const pmix_info_t directives[], size_t ndirs,  |
| 8               |           |            | <pre>pmix_info_cbfunc_t cbfunc, void *cbdata)</pre>  |
|                 |           |            | C  |
|                 |           |            |  |
| 19              |           | IN         | targets  |
| 20              |           | 18.1       | Array of proc structures (array of handles)  |
| 21              |           | IN         | ntargets Number of allowest in the towards array (interest)  |
| 22              |           | INI        | Number of element in the <i>targets</i> array (integer)  |
| 23<br>24        |           | IN         | Array of info atrustures (array of handles)  |
| 25              |           | IN         | Array of info structures (array of handles) ndirs  |
| 26              |           | 114        | Number of element in the <i>directives</i> array (integer)   |
| 27              |           | IN         | cbfunc   |
| 28              |           | 114        | Callback function pmix_info_cbfunc_t (function reference)  |
| 29              |           | IN         | cbdata   |
| 30              |           |            | Data to be passed to the callback function (memory reference)  |
| 31              |           | Retu       | arns one of the following:   |
|                 |           |            | -  |
| 32              |           |            | MIX_SUCCESS, indicating that the request is being processed by the host environment - result ill be returned in the provided obtains. Note that the library must not invoke the callback                     |
| 33<br>34        |           |            | ill be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback unction prior to returning from the API.  |
| <del>۱</del> ۲ر |           | 10         | menon prior to returning from the Arri.  |

| 1<br>2               | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and<br/>returned success - the cbfunc will not be called</li> </ul>   |
|----------------------|--|
| 3<br>4               | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called   |
|                      | ▼ Required Attributes  |
| 5<br>6<br>7<br>8     | 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.       |
| 10<br>11             | Host environments that implement support for this operation are required to support the following attributes:  |
| 12<br>13<br>14<br>15 | PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)  Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference. |
| 16<br>17             | PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool) Pause the specified processes.   |
| 18<br>19             | PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)  Resume ("un-pause") the specified processes.  |
| 20<br>21             | <pre>PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool) Forcibly terminate the specified processes and cleanup.</pre>   |
| 22<br>23             | <pre>PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int) Send given signal to specified processes.</pre>   |
| 24<br>25             | <pre>PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)     Politely terminate the specified processes.</pre>  |
| 26<br>27             | <pre>PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*) Comma-delimited list of files to be removed upon process termination</pre>   |
| 28<br>29             | <pre>PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*) Comma-delimited list of directories to be removed upon process termination</pre>  |
| 30<br>31             | PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)  Recursively cleanup all subdirectories under the specified one(s)  |
| 32<br>33             | PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool) Only remove empty subdirectories  |
| 34                   | <pre>PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)</pre>   |
|                      |  |

| 1                | Comma-delimited list of filenames that are not to be removed  |
|------------------|---|
| 2<br>3<br>4      | <pre>PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)     When recursively cleaning subdirectories, do not remove the top-level directory (the one given in the cleanup request)</pre>   |
|                  | ▼ Optional Attributes   |
| 5                | The following attributes are optional for host environments that support this operation:  |
| 6<br>7<br>8<br>9 | <pre>PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)     Cancel the specified request - the provided request ID must match the     PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of     NULL implies cancel all requests from this requestor.</pre> |
| 0<br>1           | <pre>PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*) Restart the specified processes using the given checkpoint ID.</pre>  |
| 2<br>3           | <pre>PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)</pre>   |
| 4<br>5           | <pre>PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool) Use event notification to trigger a process checkpoint.</pre>  |
| 6<br>7           | <pre>PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int) Use the given signal to trigger a process checkpoint.</pre>   |
| 8<br>9           | <pre>PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int) Time in seconds to wait for a checkpoint to complete.</pre>  |
| 0<br>1<br>2      | PMIX_JOB_CTRL_CHECKPOINT_METHOD  "pmix.jctrl.ckmethod" (pmix_data_array_t)  Array of pmix_info_t declaring each method and value supported by this application.   |
| 3<br>4           | <pre>PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*) Regular expression identifying nodes that are to be provisioned.</pre>  |
| 5<br>6           | <pre>PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*) Name of the image that is to be provisioned.</pre>   |
| 7<br>8           | PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)  Indicate that the job can be pre-empted.   |

### Description

Non-blocking form of the **PMIx\_Job\_control** API. The *targets* array identifies the processes to which the requested job control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's namespace. The use of **PMIX\_RANK\_WILDCARD** can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as **pmix\_info\_t** structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of **pmix info t** structures.

# 7.4 Process and Job Monitoring

In addition to external faults, a common problem encountered in HPC applications is a failure to make progress due to some internal conflict in the computation. These situations can result in a significant waste of resources as the SMS is unaware of the problem, and thus cannot terminate the job. Various watchdog methods have been developed for detecting this situation, including requiring a periodic "heartbeat" from the application and monitoring a specified file for changes in size and/or modification time.

At the request of SMS vendors and members, a monitoring support interface has been included in the PMIx v2 standard. The defined API allows applications to request monitoring, directing what is to be monitored, the frequency of the associated check, whether or not the application is to be notified (via the event notification subsystem) of stall detection, and other characteristics of the operation. In addition, heartbeat and file monitoring methods have been included in the PRI but are active only when requested.

# 7.4.1 PMIx Process monitor

### Summary

Request that application processes be monitored.

### Format

```
pmix_status_t

PMIx_Process_monitor(const pmix_info_t *monitor, pmix_status_t error,

const pmix_info_t directives[], size_t ndirs,

pmix_info_t *results[], size_t *nresults)
```

PMIx v3.0

| 1<br>2<br>3                | <ul><li>IN directives</li></ul>  |
|----------------------------|--|
| 4<br>5<br>6<br>7<br>8<br>9 | Number of elements in the <i>directives</i> array (integer)  INOUT results  Address where a pointer to an array of pmix_info_t containing the results of the request can be returned (memory reference)  INOUT nresults  Address where the number of elements in results can be returned (handle)  |
| 10                         | Returns one of the following:  |
| 1<br> 2                    | <ul> <li>PMIX_SUCCESS, indicating that the request was processed and returned success. Details of the result will be returned in the results array</li> </ul>  |
| 13                         | • a PMIx error constant indicating either an error in the input or that the request was refused  |
|                            | → Optional Attributes  |
| 4<br> 5<br> 6<br> 7        | The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is <i>required</i> to add the PMIX_USERID and the PMIX_GRPID attributes of the requesting process: |
| 19<br>20                   | <pre>PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.</pre>   |
| 21<br>22                   | <pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)     Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>  |
| 23<br>24                   | <pre>PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.</pre>   |
| 25<br>26                   | PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void) Register to have the PMIx server monitor the requestor for heartbeats.  |
| 27<br>28                   | <pre>PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t) Time in seconds before declaring heartbeat missed.</pre>  |
| 29<br>30                   | PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)  Number of heartbeats that can be missed before generating the event.   |
| 31<br>32                   | <pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>   |
| 33<br>34                   | <pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool) Monitor size of given file is growing to determine if the application is running.</pre>  |
| 35                         | PMIX MONITOR FILE ACCESS "pmix.monitor.faccess" (char*)  |

Monitor time since last access of given file to determine if the application is running. 1 PMIX MONITOR FILE MODIFY "pmix.monitor.fmod" (char\*) 2 Monitor time since last modified of given file to determine if the application is running. 3 PMIX MONITOR FILE CHECK TIME "pmix.monitor.ftime" (uint32 t) 4 5 Time in seconds between checking the file. 6 PMIX\_MONITOR\_FILE\_DROPS "pmix.monitor.fdrop" (uint32\_t) 7 Number of file checks that can be missed before generating the event. **Description** 8 9 Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not 10 become "wedged". When a monitor detects the specified alarm condition, it will generate an event 11 notification using the provided error code and passing along any available relevant information. It 12 13 is up to the caller to register a corresponding event handler. 14 The *monitor* argument is an attribute indicating the type of monitor being requested. For example, PMIX\_MONITOR\_FILE to indicate that the requestor is asking that a file be monitored. 15 The error argument is the status code to be used when generating an event notification alerting that 16 the monitor has been triggered. The range of the notification defaults to 17 PMIX RANGE\_NAMESPACE. This can be changed by providing a PMIX\_RANGE directive. 18 19 The directives argument characterizes the monitoring request (e.g., monitor file size) and frequency 20 of checking to be done 7.4.2 PMIx Process monitor nb Summarv 22 23 Request that application processes be monitored. **Format** 24 PMIx v2.025 pmix\_status\_t 26 PMIx\_Process\_monitor\_nb(const pmix\_info\_t \*monitor, pmix\_status\_t error, const pmix info t directives[], size t ndirs, 27 pmix info cbfunc t cbfunc, void \*cbdata) 28

| C  |       |
|--|-------|
| IN monitor   |       |
| info (handle)  |       |
| IN error   |       |
| status (integer)   |       |
| IN directives  |       |
| Array of info structures (array of handles)  |       |
| IN ndirs   |       |
| Number of elements in the <i>directives</i> 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 - rewill be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.   | esult |
| • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed a returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called  | and   |
| • a PMIx error constant indicating either an error in the input or that the request was immediate processed and failed - the <i>cbfunc</i> will <i>not</i> be called   | ely   |
| ▼ Optional Attributes  |       |
| The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is <i>required</i> to add the PMIX_USERID and the PMIX_GRPID attributes of the requesting process: |       |
| PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.  |       |
| <pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)     Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>  |       |
| PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)  The application desires to control the response to a monitoring event.   |       |
| PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)   |       |

Register to have the PMIx server monitor the requestor for heartbeats.

Time in seconds before declaring heartbeat missed.

| 1<br>2                 | PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)  Number of heartbeats that can be missed before generating the event.  |
|------------------------|---|
| 3<br>4                 | <pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>  |
| 5<br>6                 | <pre>PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)     Monitor size of given file is growing to determine if the application is running.</pre>   |
| 7<br>8                 | <pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*) Monitor time since last access of given file to determine if the application is running.</pre>   |
| 9<br>10                | <pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)     Monitor time since last modified of given file to determine if the application is running.</pre>  |
| 11<br>12               | <pre>PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)</pre>   |
| 13<br>14               | PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)  Number of file checks that can be missed before generating the event.  |
| 15<br>16<br>17<br>18   | <b>Description</b> Non-blocking form of the <b>PMIx_Process_monitor</b> API. The <i>cbfunc</i> function provides a <i>status</i> to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the <b>pmix_info_cbfunc_t</b> array of <b>pmix_info_t</b> structures. |
| 19 <b>7.4.3</b>        | PMIx_Heartbeat  |
| 20<br>21               | Summary Send a heartbeat to the PMIx server library   |
| 22<br><i>PMIx v2.0</i> | Format C  |
| 23                     | PMIx_Heartbeat (void)   |
| 24<br>25<br>26         | <b>Description</b> A simplified macro wrapping <b>PMIx_Process_monitor_nb</b> that sends a heartbeat to the PMIx server library.  |

# ₁ 7.5 Logging

The logging interface supports posting information by applications and SMS elements to persistent storage. This function is *not* intended for output of computational results, but rather for reporting status and saving state information such as inserting computation progress reports into the application's SMS job log or error reports to the local syslog.

# 7.5.1 PMIx\_Log

```
Summary
 7
 8
               Log data to a data service.
               Format
 9
   PMIx v3.0
10
               pmix_status_t
               PMIx_Log(const pmix_info_t data[], size_t ndata,
11
                           const pmix_info_t directives[], size_t ndirs)
12
                                                          C
               IN
13
                    data
                    Array of info structures (array of handles)
14
               IN
                   ndata
15
16
                    Number of elements in the data array (size t)
17
                    directives
                    Array of info structures (array of handles)
18
               IN
                   ndirs
19
20
                    Number of elements in the directives array (size_t)
               Return codes are one of the following:
21
22
                PMIX SUCCESS The logging request was successful.
                PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry.
23
24
                PMIX_ERR_NOT_SUPPORTED The PMIx implementation or host environment does not
25
                    support this function.
                                                Required Attributes
               If the PMIx library does not itself perform this operation, then it is required to pass any attributes
26
27
               provided by the client to the host environment for processing. In addition, it must include the
               following attributes in the passed info array:
28
29
               PMIX_USERID "pmix.euid" (uint32_t)
                     Effective user id.
30
31
               PMIX_GRPID "pmix.egid" (uint32_t)
32
                     Effective group id.
```

```
1
             Host environments or PMIx libraries that implement support for this operation are required to
 2
             support the following attributes:
             PMIX LOG STDERR "pmix.log.stderr" (char*)
 4
5
                   Log string to stderr.
             PMIX LOG STDOUT "pmix.log.stdout" (char*)
6
7
                   Log string to stdout.
             PMIX LOG SYSLOG "pmix.log.syslog" (char*)
8
                   Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
9
                   otherwise to local syslog
10
11
             PMIX LOG LOCAL SYSLOG "pmix.log.lsys" (char*)
                   Log data to local syslog. Defaults to ERROR priority.
12
             PMIX LOG GLOBAL SYSLOG "pmix.log.gsys" (char*)
13
                   Forward data to system "gateway" and log msg to that syslog Defaults to ERROR priority.
14
             PMIX LOG SYSLOG PRI "pmix.log.syspri" (int)
15
16
                   Syslog priority level
17
             PMIX_LOG_ONCE "pmix.log.once" (bool)
                   Only log this once with whichever channel can first support it, taking the channels in priority
18
19

    ▼------ Optional Attributes ------

             The following attributes are optional for host environments or PMIx libraries that support this
20
21
             operation:
22
             PMIX LOG SOURCE "pmix.log.source" (pmix proc t*)
                   ID of source of the log request
23
24
             PMIX LOG TIMESTAMP "pmix.log.tstmp" (time t)
25
                   Timestamp for log report
             PMIX LOG GENERATE TIMESTAMP "pmix.log.gtstmp" (bool)
26
27
                   Generate timestamp for log
             PMIX LOG TAG OUTPUT "pmix.log.tag" (bool)
28
                   Label the output stream with the channel name (e.g., "stdout")
29
             PMIX LOG TIMESTAMP OUTPUT "pmix.log.tsout" (bool)
30
31
                   Print timestamp in output string
32
             PMIX LOG XML OUTPUT "pmix.log.xml" (bool)
33
                   Print the output stream in XML format
```

```
Log via email based on pmix info t containing directives.
 2
 3
               PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
                     Comma-delimited list of email addresses that are to receive the message.
 4
 5
               PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
 6
                     Subject line for email.
               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
               PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
11
                     Store the log data in a global data store (e.g., database)
12
13
               Description
               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.
15
                                           Advice to users
16
               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

PMIX\_LOG\_EMAIL "pmix.log.email" (pmix\_data\_array\_t)

# 21 **7.5.2 PMIx\_Log\_nb**

## 22 Summary

1

17 18

19

20

23

Log data to a data service.

and does not indicate receipt at the final destination.

| 1                | Format C   |
|------------------|--|
| <i>PMIx v2.0</i> |  |
| 2                | pmix_status_t  |
| 3                | <pre>PMIx_Log_nb(const pmix_info_t data[], size_t ndata,</pre>   |
| 4                | <pre>const pmix_info_t directives[], size_t ndirs,</pre>   |
| 5                | <pre>pmix_op_cbfunc_t cbfunc, void *cbdata)</pre>  |
|                  | C  |
| 6                | IN data  |
| 7                | Array of info structures (array of handles)  |
| 8                | IN ndata   |
| 9                | Number of elements in the <i>data</i> array (size_t)   |
| 10               | IN directives  |
| 11               | Array of info structures (array of handles)  |
| 12               | IN ndirs   |
| 13               | Number of elements in the <i>directives</i> array (size_t)   |
| 14               | IN cbfunc  |
| 15               | Callback function <b>pmix_op_cbfunc_t</b> (function reference)   |
| 16               | IN cbdata  |
| 17               | Data to be passed to the callback function (memory reference)  |
| 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 <i>info</i> array:  |
| 31               | <pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>  |
| 32               | Effective user id.   |
|                  |  |
| 33               | PMIX_GRPID "pmix.egid" (uint32_t)  |
| 34               | Effective group id.  |

| 1              |  |
|----------------|--|
| 2              | Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:  |
| 4<br>5         | <pre>PMIX_LOG_STDERR "pmix.log.stderr" (char*) Log string to stderr.</pre>   |
| 6<br>7         | <pre>PMIX_LOG_STDOUT "pmix.log.stdout" (char*) Log string to stdout.</pre>   |
| 8<br>9<br>10   | <pre>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</pre> |
| 11<br>12       | PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)  Log data to local syslog. Defaults to ERROR priority.   |
| 13<br>14       | <pre>PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*) Forward data to system "gateway" and log msg to that syslog Defaults to ERROR priority.</pre>                      |
| 15<br>16       | <pre>PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int) Syslog priority level</pre>   |
| 17<br>18<br>19 | 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  |
| 20<br>21       | The following attributes are optional for host environments or PMIx libraries that support this operation:   |
| 22<br>23       | <pre>PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*) ID of source of the log request</pre>  |
| 24<br>25       | <pre>PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)</pre>  |
| 26<br>27       | <pre>PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool) Generate timestamp for log</pre>   |
| 28<br>29       | PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)  Label the output stream with the channel name (e.g., "stdout")  |
| 30<br>31       | <pre>PMIX_LOG_TIMESTAMP_OUTPUT</pre>   |
| 32             | Print the output stream in XMI format  |

PMIX\_LOG\_EMAIL "pmix.log.email" (pmix\_data\_array\_t) 1 Log via email based on **pmix info** t containing directives. 2 3 PMIX\_LOG\_EMAIL\_ADDR "pmix.log.emaddr" (char\*) Comma-delimited list of email addresses that are to receive the message. 4 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 PMIX\_LOG\_GLOBAL\_DATASTORE "pmix.log.gstore" (bool) 11 Store the log data in a global data store (e.g., database) 12

## **Description**

13

14 15

16 17

18 19

20

21

22

Log data subject to the services offered by the host environment. The data to be logged is provided in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The callback function will be executed when the log operation has been completed. The *data* and *directives* arrays must be maintained until the callback is provided.

### Advice to users -

It is strongly recommended that the PMIx\_Log\_nb API not be used by applications for streaming data as it is not a "performant" transport and can perturb the application since it involves the local PMIx server and host SMS daemon. Note that a return of PMIX\_SUCCESS only denotes that the data was successfully handed to the appropriate system call (for local channels) or the host environment and does not indicate receipt at the final destination.

#### **CHAPTER 8**

# **Event Notification**

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

# 4 8.1 Notification and Management

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

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

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

## Advice to PMIx library implementers –

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

8 9

3

13

19

24

30

31

35

36

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

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

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

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

Users can specify the callback order of a handler within its category at the time of registration. Ordering can be specified either by providing the relevant returned event handler registration ID or using event handler names, if the user specified an event handler name when registering the corresponding event. Thus, users can specify that a given handler be executed before or after another handler should both handlers appear in an event chain (the ordering is ignored if the other handler isn't included). Note that ordering does not imply immediate relationships. For example, multiple handlers registered to be serviced after event handler A will all be executed after A, but are not guaranteed to be executed in any particular order amongst themselves.

In addition, one event handler can be declared as the *first* handler to be executed in the chain. This handler will always be called prior to any other handler, regardless of category, provided the incoming event matches both the specified range and event code. Only one handler can be so designated — attempts to designate additional handlers as first will return an error. Deregistration of the declared *first* handler will re-open the position for subsequent assignment.

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

#### Advice to users

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

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

# 7 8.1.1 PMIx\_Register\_event\_handler

### **Summary**

Register an event handler

#### Format

PMIx v2.0

 IN codes

Array of status codes (array of pmix\_status\_t)

IN ncodes

Number of elements in the *codes* array (size\_t)

IN info

Array of info structures (array of handles)

| 1<br>2<br>3<br>4<br>5<br>6<br>7<br>8 | <ul> <li>IN ninfo         Number of elements in the info array (size_t)</li> <li>IN evhdlr         Event handler to be called pmix_notification_fn_t (function reference)</li> <li>IN cbfunc         Callback function pmix_evhdlr_reg_cbfunc_t (function reference)</li> <li>IN cbdata         Data to be passed to the cbfunc callback function (memory reference)</li> </ul>   |
|--------------------------------------|---|
| 9<br> 0<br> 1                        | If <i>cbfunc</i> is <b>NULL</b> , the function call will be treated as a <i>blocking</i> call. In this case, the returned status will be either (a) the event handler reference identifier if the value is greater than or equal to zero, or (b) a negative error code indicative of the reason for the failure.  |
| 12<br>13                             | If the <i>cbfunc</i> is non- <b>NULL</b> , the function call will be treated as a <i>non-blocking</i> call and will return the following:   |
| 14<br>15<br>16<br>17<br>18           | PMIX_SUCCESS indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API. The event handler identifier will be returned in the callback a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed. |
| 20<br>21<br>22                       | The callback function must not be executed prior to returning from the API, and no events corresponding to this registration may be delivered prior to the completion of the registration callback function ( <i>cbfunc</i> ).  |
|                                      | ▼   |
| 23                                   | The following attributes are required to be supported by all PMIx libraries:  |
| 24<br>25                             | PMIX_EVENT_HDLR_NAME "pmix.evname" (char*) String name identifying this handler.  |
| 26<br>27                             | PMIX_EVENT_HDLR_FIRST "pmix.evfirst" (bool) Invoke this event handler before any other handlers.  |
| 28<br>29                             | PMIX_EVENT_HDLR_LAST "pmix.evlast" (bool) Invoke this event handler after all other handlers have been called.  |
| 30<br>31                             | PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirstcat" (bool) Invoke this event handler before any other handlers in this category.  |
| 32<br>33                             | PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool) Invoke this event handler after all other handlers in this category have been called.  |
| 34<br>35                             | <pre>PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*) Put this event handler immediately before the one specified in the (char*) value.</pre>   |
| 36                                   | <pre>PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)</pre>   |

222

| 1              | Put this event handler immediately after the one specified in the (char*) value.  |
|----------------|---|
| 2              | <pre>PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool) Prepend this handler to the precedence list within its category.</pre>   |
| 4<br>5         | PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool) Append this handler to the precedence list within its category.   |
| 6<br>7         | <pre>PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*) Array of pmix_proc_t defining range of event notification.</pre>   |
| 8<br>9         | <pre>PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.</pre>   |
| 10<br>11<br>12 | <pre>PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)    Object to be returned whenever the registered callback function cbfunc is invoked. The    object will only be returned to the process that registered it.</pre>   |
| 13             |   |
| 14<br>15       | Host environments that implement support for PMIx event notification are required to support the following attributes:  |
| 16<br>17       | <pre>PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t) The single process that was affected.</pre>   |
| 18<br>19       | PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)  Array of pmix_proc_t defining affected processes.   |
|                | ▼ Optional Attributes   |
| 20<br>21<br>22 | Host environments that support PMIx event notification <i>may</i> offer notifications for environmental events impacting the job and for SMS events relating to the job. The following attributes are optional for host environments that support this operation: |
| 23<br>24       | PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool) The RM intends to terminate this session.  |
| 25<br>26       | <pre>PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool) The RM intends to terminate this job.</pre>  |
| 27<br>28       | PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool) The RM intends to terminate all processes on this node.   |
| 29<br>30       | <pre>PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool) The RM intends to terminate just this process.</pre>   |
| 31<br>32       | PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int)  The time in seconds before the RM will execute error response.  |
| 33             | PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)  |

Do not generate an event when this job normally terminates.

### Description

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

## Advice to users

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

# Advice to users

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

# 18 8.1.2 PMIx\_Deregister\_event\_handler

## Summary

Deregister an event handler.

| 1        |                  | Format   |
|----------|------------------|--|
|          | <i>PMIx v2.0</i> | · · · · · · · · · · · · · · · · · · ·  |
| 2        |                  | pmix_status_t  |
| 3        |                  | <pre>PMIx_Deregister_event_handler(size_t evhdlr_ref,</pre>  |
| 4        |                  | <pre>pmix_op_cbfunc_t cbfunc,</pre>  |
| 5        |                  | <pre>void *cbdata);</pre>  |
|          |                  | C -  |
| 6        |                  | <pre>IN evhdlr_ref</pre>   |
| 7        |                  | Event handler ID returned by registration (size_t)   |
| 8        |                  | IN cbfunc  |
| 9        |                  | Callback function to be executed upon completion of operation <pre>pmix_op_cbfunc_t</pre> (function reference)           |
| 10<br>11 |                  | IN cbdata  |
| 12       |                  | Data to be passed to the cbfunc callback function (memory reference)   |
| 13       |                  | If <i>cbfunc</i> is <b>NULL</b> , the function will be treated as a <i>blocking</i> call and the result of the operation |
| 14       |                  | returned in the status code.   |
| 15       |                  | If <i>cbfunc</i> is non- <b>NULL</b> , the function will be treated as a <i>non-blocking</i> call and return one of the  |
| 16       |                  | following:   |
| 17       |                  | • PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the                          |
| 18       |                  | provided cbfunc. Note that the library must not invoke the callback function prior to returning                          |
| 19       |                  | from the API.  |
| 20       |                  | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and                                    |
| 21       |                  | returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> 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 <i>cbfunc</i> will <i>not</i> be called   |
| 24       |                  | The returned status code will be one of the following:   |
| 25       |                  | PMIX_SUCCESS The event handler was successfully deregistered.  |
| 26       |                  | <b>PMIX_ERR_BAD_PARAM</b> The provided <i>evhdlr_ref</i> was unrecognized.   |
| 27       |                  | <b>PMIX_ERR_NOT_SUPPORTED</b> The PMIx implementation does not support event notification.                               |
| 28       |                  | Description  |
| 29       |                  | Deregister an event handler. Note that no events corresponding to the referenced registration may                        |
| 30       |                  | be delivered following completion of the deregistration operation (either return from the API with                       |
| 31       |                  | <b>PMIX_OPERATION_SUCCEEDED</b> or execution of the <i>cbfunc</i> ).   |
|          |                  |  |
| 32       | 8.1.3            | PMIx_Notify_event  |

Report an event for notification via any registered event handler.

Summary

33

34

| 1  |                  | FOI   | rmat  |
|----|------------------|-------|---|
|    | <i>PMIx v2.0</i> |       |   |
| 2  |                  | pmi   | ix_status_t   |
| 3  |                  | PM1   | <pre>Ix_Notify_event(pmix_status_t status,</pre>  |
| 4  |                  |       | <pre>const pmix_proc_t *source,</pre>   |
| 5  |                  |       | <pre>pmix_data_range_t range,</pre>   |
| 6  |                  |       | <pre>pmix_info_t info[], size_t ninfo,</pre>  |
| 7  |                  |       | <pre>pmix_op_cbfunc_t cbfunc, void *cbdata);</pre>  |
|    |                  |       | 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 <i>info</i> 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)                                      |
|    |                  | T.C   | •   |
| 24 |                  |       | bfunc is <b>NULL</b> , the function will be treated as a blocking call and the result of the operation    |
| 25 |                  | retu  | rned in the status code.  |
| 26 |                  | If c  | bfunc is non-NULL, the function will be treated as a non-blocking call and return one of the              |
| 27 |                  | follo | owing:  |
| 28 |                  | DN    | IX_SUCCESS The notification request is valid and is being processed. The callback function                |
| 29 |                  |       | will be called when the process-local operation is complete and will provide the resulting                |
| 30 |                  |       | status of that operation. Note that this does <i>not</i> reflect the success or failure of delivering the |
| 31 |                  |       | event to any recipients. The callback function must not be executed prior to returning from the           |
| 32 |                  |       | API.  |
| 33 |                  | PΝ    | <b>IIX_OPERATION_SUCCEEDED</b> , indicating that the request was immediately processed and                |
| 34 |                  |       | returned success - the cbfunc will not be called  |
| 35 |                  | PM    | IIX_ERR_BAD_PARAM The request contains at least one incorrect entry that prevents it from                 |
| 36 |                  |       | being processed. The callback function will <i>not</i> be called.   |

1 **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support event notification, 2 or in the case of a PMIx server calling the API, the range extended beyond the local node and the host SMS environment does not support event notification. The callback function will not 3 4 be called. Required Attributes 5 The following attributes are required to be supported by all PMIx libraries: 6 PMIX\_EVENT\_NON\_DEFAULT "pmix.evnondef" (bool) Event is not to be delivered to default event handlers. PMIX\_EVENT\_CUSTOM\_RANGE "pmix.evrange" (pmix\_data\_array\_t\*) 8 Array of pmix\_proc\_t defining range of event notification. 9 10 Host environments that implement support for PMIx event notification are required to provide the 11 following attributes for all events generated by the environment: 12 PMIX\_EVENT\_AFFECTED\_PROC "pmix.evproc" (pmix\_proc\_t) 13 14 The single process that was affected. PMIX\_EVENT\_AFFECTED\_PROCS "pmix.evaffected" (pmix\_data\_array\_t\*) 15 Array of **pmix\_proc\_t** defining affected processes. 16

## **Description**

17

18

19

20 21

22

23

24 25

26 27

28

29

30

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.

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**

1

3

4

5 6

7

9

# **Data Packing and Unpacking**

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

# 9.1 Data Buffer Type

The pmix\_data\_buffer\_t structure describes a data buffer used for packing and unpacking.

```
PMIx v2.0
10
            typedef struct pmix_data_buffer {
                /** Start of my memory */
11
                char *base_ptr;
12
                /** Where the next data will be packed to
13
14
                     (within the allocated memory starting
15
                    at base_ptr) */
                char *pack ptr;
16
                /** Where the next data will be unpacked
17
18
                    from (within the allocated memory
19
                    starting as base ptr) */
20
                char *unpack ptr;
                /** Number of bytes allocated (starting
21
22
                    at base_ptr) */
                size_t bytes_allocated;
23
24
                /** Number of bytes used by the buffer
25
                     (i.e., amount of data -- including
                    overhead -- packed in the buffer) */
26
27
                size_t bytes_used;
            } pmix_data_buffer_t;
28
```

# 9.2 Support Macros

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

# 3 9.2.1 PMIX DATA BUFFER CREATE

### 4 Summary

Allocate memory for a pmix\_data\_buffer\_t object and initialize it

#### Format

PMIx v2.0

5

7

8

9

10

11

14

15

16

18

20

23

PMIX\_DATA\_BUFFER\_CREATE(buffer);

OUT buffer

Variable to be assigned the pointer to the allocated **pmix\_data\_buffer\_t** (handle)

### Description

This macro uses *calloc* to allocate memory for the buffer and initialize all fields in it

## 2 9.2.2 PMIX DATA BUFFER RELEASE

### 13 Summary

Free a pmix\_data\_buffer\_t object and the data it contains

#### Format

PMIx v2.0

PMIX\_DATA\_BUFFER\_RELEASE(buffer);

### 17 **IN** buffer

Pointer to the **pmix\_data\_buffer\_t** to be released (handle)

## 19 **Description**

Free's the data contained in the buffer, and then free's the buffer itself

## 1 9.2.3 PMIX DATA BUFFER CONSTRUCT

## 22 Summary

Initialize a statically declared **pmix** data buffer t object

```
Format
1
   PMIx v2.0
2
             PMIX DATA BUFFER CONSTRUCT (buffer);
              IN
3
                 buffer
                  Pointer to the allocated pmix_data_buffer_t that is to be initialized (handle)
 4
              Description
 5
              Initialize a pre-allocated buffer object
 6
   9.2.4
            PMIX DATA BUFFER DESTRUCT
              Summarv
8
              Release the data contained in a pmix_data_buffer_t object
9
10
              Format
   PMIx v2.0
11
             PMIX DATA BUFFER DESTRUCT (buffer);
12
              IN
                  buffer
                  Pointer to the pmix_data_buffer_t whose data is to be released (handle)
13
              Description
14
              Free's the data contained in a pmix_data_buffer_t object
15
   9.2.5
            PMIX DATA BUFFER LOAD
17
              Summary
              Load a blob into a pmix_data_buffer_t object
18
              Format
19
   PMIx v2.0
20
              PMIX DATA BUFFER LOAD (buffer, data, size);
             IN
                buffer
21
                  Pointer to a pre-allocated pmix_data_buffer_t (handle)
22
             IN
23
                  data
24
                  Pointer to a blob (char*)
25
              IN
                  size
26
                  Number of bytes in the blob size_t
```

### Description

1

3

6

7

9

11

12 13

14

15

16 17

18

19

20

22

25

2 Load the given data into the provided **pmix\_data\_buffer\_t** object, usually done in

preparation for unpacking the provided data. Note that the data is *not* copied into the buffer - thus,

the blob must not be released until after operations on the buffer have completed.

# 9.2.6 PMIX DATA BUFFER UNLOAD

### Summary

Unload the data from a pmix\_data\_buffer\_t object

### Format

PMIx v2.0

PMIX\_DATA\_BUFFER\_UNLOAD (buffer, data, size);

10 **IN** buffer

Pointer to the **pmix\_data\_buffer\_t** whose data is to be extracted (handle)

OUT data

Variable to be assigned the pointer to the extracted blob (void\*)

OUT size

Variable to be assigned the number of bytes in the blob size t

### Description

Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to

the provided variables, usually done to transmit the blob to a remote process for unpacking. The buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release -

thus, the user is responsible for releasing the blob when done with it.

# 9.3 General Routines

The following routines are provided to support internode transfers in heterogeneous environments.

# 9.3.1 PMIx\_Data\_pack

## 24 Summary

Pack one or more values of a specified type into a buffer, usually for transmission to another process

| ı  | Format   |  |  |
|--|--|--|--|
| <i>PMIx v2.0</i>                                       | · · · · · · · · · · · · · · · · · · ·  |  |  |
| 2  | pmix_status_t  |  |  |
| <pre>3 PMIx_Data_pack(const pmix_proc_t *target,</pre> |  |  |  |
| 4  | <pre>pmix_data_buffer_t *buffer,</pre>   |  |  |
| 5  | <pre>void *src, int32_t num_vals,</pre>  |  |  |
| 6  | <pre>pmix_data_type_t type);</pre>   |  |  |
|  | C  |  |  |
| 7  | IN target  |  |  |
| 8  | Pointer to a pmix_proc_t containing the nspace/rank of the process that will be unpacking  |  |  |
| 9  | the final buffer. A NULL value may be used to indicate that the target is based on the same  |  |  |
| 10   | PMIx version as the caller. Note that only the target's nspace is relevant. (handle)   |  |  |
| 11   | IN buffer  |  |  |
| 12   | Pointer to a pmix_data_buffer_t where the packed data is to be stored (handle)   |  |  |
| 13   | IN src   |  |  |
| 14   | Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the  |  |  |
| 15   | caller must pass the address of the pointer to the string as the (void*). This allows the caller to  |  |  |
| 16   | pass multiple strings in a single call. (memory reference)   |  |  |
| 17   | IN num_vals  |  |  |
| 18   | Number of elements pointed to by the <i>src</i> pointer. A string value is counted as a single value   |  |  |
| 19   | regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,   |  |  |
|  |  |  |  |
| 20   | string arrays) should be contiguous, although the data pointed to need not be contiguous   |  |  |
| 21   | across array entries.(int32_t)   |  |  |
| 22   | IN type  The type of the data to be realised (remine alot to be realised).   |  |  |
| 23   | The type of the data to be packed ( <b>pmix_data_type_t</b> )  |  |  |
| 24   | Returns one of the following:  |  |  |
| 25   | PMIX_SUCCESS The data has been packed as requested   |  |  |
| 26   | PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.   |  |  |
| 27   | PMIX_ERR_BAD_PARAM The provided buffer or src is NULL  |  |  |
| 28   | PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this  |  |  |
| 29   | implementation   |  |  |
| 30   | PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation  |  |  |
| 31   | PMIX_ERROR General error   |  |  |
| 20   | Description  |  |  |
| 32   | <b>Description</b> The state of the |  |  |
| 33   | The pack function packs one or more values of a specified type into the specified buffer. The buffer   |  |  |
| 34   | must have already been initialized via the PMIX_DATA_BUFFER_CREATE or  |  |  |
| 35   | PMIX_DATA_BUFFER_CONSTRUCT macros — otherwise, PMIx_Data_pack will return an   |  |  |
| 36   | error. Providing an unsupported type flag will likewise be reported as an error.   |  |  |
| 37<br>38   | 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   |  |  |
|  | The first of the superior of t     |  |  |

will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code (generated upon unpacking) — the error cannot be detected during packing.

The namespace of the intended recipient of the packed buffer (i.e., the process that will be unpacking it) is used solely to resolve any data type differences between PMIx versions. The recipient must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the recipient is using. Note that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the target's namespace.

# 9.3.2 PMIx\_Data\_unpack

### Summary

Unpack values from a pmix\_data\_buffer\_t

### **Format**

*PMIx v2.0* 

C

C

#### IN source

Pointer to a **pmix\_proc\_t** structure containing the nspace/rank of the process that packed the provided buffer. A NULL value may be used to indicate that the source is based on the same PMIx version as the caller. Note that only the source's nspace is relevant. (handle)

#### IN buffer

A pointer to the buffer from which the value will be extracted. (handle)

#### **INOUT** dest

A pointer to the memory location into which the data is to be stored. Note that these values will be stored contiguously in memory. For strings, this pointer must be to (char\*\*) to provide a means of supporting multiple string operations. The unpack function will allocate memory for each string in the array - the caller must only provide adequate memory for the array of pointers. (void\*)

#### INOUT max num values

The number of values to be unpacked — upon completion, the parameter will be set to the actual number of values unpacked. In most cases, this should match the maximum number provided in the parameters — but in no case will it exceed the value of this parameter. Note that unpacking fewer values than are actually available will leave the buffer in an unpackable state — the function will return an error code to warn of this condition.(int32\_t)

IN type The type of the data to be unpacked — must be one of the PMIx defined data types ( pmix\_data\_type\_t) Returns one of the following: PMIX\_SUCCESS The data has been unpacked as requested PMIX ERR NOT SUPPORTED The PMIx implementation does not support this function. PMIX ERR BAD PARAM The provided buffer or dest is NULL PMIX\_ERR\_UNKNOWN\_DATA\_TYPE The specified data type is not known to this implementation PMIX ERR OUT OF RESOURCE Not enough memory to support the operation PMIX ERROR General error

### **Description**

The unpack function unpacks the next value (or values) of a specified type from the given buffer. The buffer must have already been initialized via an PMIX\_DATA\_BUFFER\_CREATE or PMIX\_DATA\_BUFFER\_CONSTRUCT call (and assumedly filled with some data) — otherwise, the unpack\_value function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

NOTE: it is possible for the buffer to be corrupted and that PMIx will *think* there is a proper variable type at the beginning of an unpack region — but that the value is bogus (e.g., just a byte field in a string array that so happens to have a value that matches the specified data type flag). Therefore, the data type error check is *not* completely safe.

Unpacking values is a "nondestructive" process — i.e., the values are not removed from the buffer. It is therefore possible for the caller to re-unpack a value from the same buffer by resetting the unpack\_ptr.

Warning: The caller is responsible for providing adequate memory storage for the requested data. The user must provide a parameter indicating the maximum number of values that can be unpacked into the allocated memory. If more values exist in the buffer than can fit into the memory storage, then the function will unpack what it can fit into that location and return an error code indicating that the buffer was only partially unpacked.

Note that any data that was not hard type cast (i.e., not type cast to a specific size) when packed may lose precision when unpacked by a non-homogeneous recipient. PMIx will do its best to deal with heterogeneity issues between the packer and unpacker in such cases. Sending a number larger than can be handled by the recipient will return an error code generated upon unpacking — these errors cannot be detected during packing.

The namespace of the process that packed the buffer is used solely to resolve any data type differences between PMIx versions. The packer must, therefore, be known to the user prior to calling the pack function so that the PMIx library is aware of the version the packer is using. Note

that all processes in a given namespace are *required* to use the same PMIx version — thus, the caller must only know at least one process from the packer's namespace.

# 9.3.3 PMIx\_Data\_copy

### Summary

Copy a data value from one location to another.

#### Format

1

4 5

6

10

11

12 13

14

15 16

17

18

19 20

21

22 23

24

25 26

27

28

29

31

32

```
PMIx v2.0

pmix_status_t

PMIx_Data_copy(void **dest, void *src,

pmix_data_type_t type);
```

#### IN dest

The address of a pointer into which the address of the resulting data is to be stored. (void\*\*)

#### IN src

A pointer to the memory location from which the data is to be copied (handle)

#### IN type

The type of the data to be copied — must be one of the PMIx defined data types. (

pmix\_data\_type\_t)

#### Returns one of the following:

```
PMIX_SUCCESS The data has been copied as requested
```

PMIX\_ERR\_NOT\_SUPPORTED The PMIx implementation does not support this function.

PMIX ERR BAD PARAM The provided src or dest is NULL

PMIX\_ERR\_UNKNOWN\_DATA\_TYPE The specified data type is not known to this implementation

PMIX\_ERR\_OUT\_OF\_RESOURCE Not enough memory to support the operation PMIX\_ERROR General error

# Description

Since registered data types can be complex structures, the system needs some way to know how to copy the data from one location to another (e.g., for storage in the registry). This function, which can call other copy functions to build up complex data types, defines the method for making a copy of the specified data type.

# 9.3.4 PMIx Data print

#### Summary

Pretty-print a data value.

```
Format
 1
   PMIx v2.0
 2
               pmix status t
 3
               PMIx_Data_print(char **output, char *prefix,
 4
                                    void *src, pmix data type t type);
               IN
 5
                    output
 6
                    The address of a pointer into which the address of the resulting output is to be stored.
 7
                    (char**)
 8
               IN
                   prefix
 9
                    String to be prepended to the resulting output (char*)
10
               IN
                    A pointer to the memory location of the data value to be printed (handle)
11
               IN
                    type
12
13
                    The type of the data value to be printed — must be one of the PMIx defined data types. (
                    pmix_data_type_t)
14
               Returns one of the following:
15
                PMIX_SUCCESS The data has been printed as requested
16
17
                PMIX ERR BAD PARAM The provided data type is not recognized.
18
                PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
               Description
19
20
               Since registered data types can be complex structures, the system needs some way to know how to
21
               print them (i.e., convert them to a string representation). Primarily for debug purposes.
    9.3.5
              PMIx Data copy payload
23
               Summary
               Copy a payload from one buffer to another
24
               Format
25
   PMIx v2.0
```

#### **Description**

 This function will append a copy of the payload in one buffer into another buffer. Note that this is *not* a destructive procedure — the source buffer's payload will remain intact, as will any pre-existing payload in the destination's buffer. Only the unpacked portion of the source payload will be copied.

#### **CHAPTER 10**

# **Security**

PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes. *Client* processes (i.e., processes started by the host environment) must be preregistered with the PMIx server library via the PMIx\_server\_register\_client API before they are spawned. This API requires that you pass the expected uid/gid of the client process.

When the client attempts to connect to the PMIx server, the server uses available standard Operating System (OS) methods to determine the effective uid/gid of the process requesting the connection. PMIx implementations shall not rely on any values reported by the client process itself as that would be unsafe. The effective uid/gid reported by the OS is compared to the values provided by the host during registration - if they don't match, the PMIx server is required to drop the connection request. This ensures that the PMIx server does not allow connection from a client that doesn't at least meet some minimal security requirement.

Once the requesting client passes the initial test, the PMIx server can, at the choice of the implementor, perform additional security checks. This may involve a variety of methods such as exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server reports the client connection request to the host via the

pmix\_server\_client\_connected\_fn\_t interface. The host may then perform any
additional checks and operations before responding with either PMIX\_SUCCESS to indicate that
the connection is approved, or a PMIx error constant indicating that the connection request is
refused. In this latter case, the PMIx server is required to drop the connection.

Tools started by the host environment are classed as a subgroup of client processes and follow the client process procedure. However, tools that are not started by the host environment must be handled differently as registration information is not available prior to the connection request. In these cases, the PMIx server library is required to use available standard OS methods to get the effective uid/gid and report them upwards as part of invoking the

**pmix\_server\_tool\_connection\_fn\_t** interface, deferring initial security screening to the host. It is recognized that this may represent a security risk - for this reason, PMIx server libraries must not enable tool connections by default. Instead, the host has to explicitly enable them via the **PMIX\_SERVER\_TOOL\_SUPPORT** attribute, thus recognizing the associated risk. Once the host has completed its authentication procedure, it again informs the PMIx server of the result.

Applications and tools often interact with the host environment in ways that require security beyond just verifying the user's identity - e.g., access to that user's relevant authorizations. This is particularly important when tools connect directly to a system-level PMIx server that may be operating at a privileged level. A variety of system management software packages provide authorization services, but the lack of standardized interfaces makes portability problematic.

This section defines two PMIx client-side APIs for this purpose. These are most likely to be used by user-space applications/tools, but are not restricted to that realm.

# 10.1 Obtaining Credentials

The API for obtaining a credential is a non-blocking operation since the host environment may have to contact a remote credential service. The definition takes into account the potential that the returned credential could be sent via some mechanism to another application that resides in an environment using a different security mechanism. Thus, provision is made for the system to return additional information (e.g., the identity of the issuing agent) outside of the credential itself and visible to the application.

# 10.1.1 PMIx\_Get\_credential

#### Summary

Request a credential from the PMIx server library or the host environment

#### Format

*PMIx v3.0* 

1

4

5

6

7

8

9

11

12

13

14

15

16

17

18 19

20

21 22

23

24 25

26

C

IN info

Array of **pmix\_info\_t** structures (array of handles)

IN ninfo

Number of elements in the *info* array (size\_t)

IN credential

Address of a pmix\_byte\_object\_t within which to return credential (handle)

Returns one of the following:

- PMIX\_SUCCESS, indicating that the credential has been returned in the provided
   pmix\_byte\_object\_t
- a PMIx error constant indicating either an error in the input or that the request is unsupported

|                                  |        | ▼  |
|----------------------------------|--------|--|
| 1<br>2                           |        | PMIx libraries that choose not to support this operation <i>must</i> return <b>PMIX_ERR_NOT_SUPPORTED</b> when the function is called.   |
| 3<br>4                           |        | 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 <i>munge</i> server).  |
| 5<br>6<br>7<br>8                 |        | 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 <i>info</i> array passed from the PMIx library to the host environment:  |
| 9<br>10                          |        | <pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>  |
| 11<br>12                         |        | PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.   |
|                                  |        | ▼ Optional Attributes  |
| 10                               |        | •  |
| 13                               |        | The following attributes are optional for host environments that support this operation:   |
| 14<br>15<br>16<br>17             |        | 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 —  |
| 18<br>19<br>20<br>21<br>22<br>23 |        | We recommend that implementation of the <b>PMIX_TIMEOUT</b> 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 <b>PMIX_TIMEOUT</b> directly in the PMIx server library must take care to resolve the race condition and should avoid passing <b>PMIX_TIMEOUT</b> to the host environment so that multiple competing timeouts are not created.  |
| 24<br>25                         |        | <b>Description</b> Request a credential from the PMIx server library or the host environment   |
| 26                               | 10.1.2 | PMIx_Get_credential_nb   |
| 27                               |        | Summary  Department of the state of the stat |
| 28                               |        | Request a credential from the PMIx server library or the host environment  |

| 1                | Format  |
|------------------|---|
| <i>PMIx v3.0</i> | ▼ C —   |
| 2                | pmix_status_t   |
| 3                | <pre>PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,</pre>   |
| 4                | pmix_credential_cbfunc_t cbfunc, void *cbdata   |
|                  | C   |
| 5                | IN info   |
| 6                | Array of pmix_info_t structures (array of handles)  |
| 7                | IN ninfo  |
| 8                | Number of elements in the <i>info</i> array (size_t)  |
| 9                | IN cbfunc   |
| 10<br>11         | Callback function to return credential ( pmix_credential_cbfunc_t function reference)   |
| 12               | IN cbdata   |
| 13               | Data to be passed to the callback function (memory reference)   |
| 14               | Returns one of the following:   |
| 15<br>16         | • PMIX_SUCCESS, indicating that the request has been communicated to the local PMIx server result will be returned in the provided <i>cbfunc</i>  |
| 17<br>18         | • a PMIx error constant indicating either an error in the input or that the request is unsupported - the <i>cbfunc</i> will <i>not</i> be called  |
|                  | ▼   |
| 19<br>20         | PMIx libraries that choose not to support this operation <i>must</i> return <b>PMIX_ERR_NOT_SUPPORTED</b> when the function is called.  |
| 21<br>22         | 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 <i>munge</i> server). |
| 23               | Implementations that support the operation but cannot directly process the client's request must  |
| 24               | pass any attributes that are provided by the client to the host environment for processing. In  |
| 25               | addition, the following attributes are required to be included in the <i>info</i> array passed from the PMIx  |
| 26               | library to the host environment:  |
| 27               | PMIX_USERID "pmix.euid" (uint32_t)  |
| 28               | Effective user id.  |
| 29               | <pre>PMIX_GRPID "pmix.egid" (uint32_t)</pre>  |
| 30               | Effective group id.   |
|                  | <u> </u>  |

#### Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX\_TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 5 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 6 7 environment due to race condition considerations between completion of the operation versus 8 internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT 9 directly in the PMIx server library must take care to resolve the race condition and should avoid 10 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not created. 11 **Description** 12 13 Request a credential from the PMIx server library or the host environment 10.2 Validating Credentials 15 The API for validating a credential is a non-blocking operation since the host environment may 16 have to contact a remote credential service. Provision is made for the system to return additional 17 information regarding possible authorization limitations beyond simple authentication.

# 18 10.2.1 PMIx\_Validate\_credential

#### 19 **Summary**

20

Request validation of a credential by the PMIx server library or the host environment

| 1        |                  | Format  |
|----------|------------------|---|
|          | <i>PMIx v3.0</i> | · · · · · · · · · · · · · · · · · · ·   |
| 2        |                  | pmix_status_t   |
| 3        |                  | <pre>PMIx_Validate_credential(const pmix_byte_object_t *cred,</pre>   |
| 4        |                  | <pre>const pmix_info_t info[], size_t ninfo,</pre>  |
| 5        |                  | <pre>pmix_info_t **results, size_t *nresults)</pre>   |
|          |                  | C —   |
| 6        |                  | IN cred   |
| 7        |                  | Pointer to pmix_byte_object_t containing the credential (handle)  |
| 8        |                  | IN info   |
| 9        |                  | Array of pmix_info_t structures (array of handles)  |
| 10       |                  | IN ninfo  |
| 11       |                  | Number of elements in the <i>info</i> array (size_t)  |
| 12       |                  | INOUT results   |
| 13<br>14 |                  | Address where a pointer to an array of <b>pmix_info_t</b> containing the results of the request can be returned (memory reference)  |
| 15       |                  | INOUT nresults  |
| 16       |                  | Address where the number of elements in <i>results</i> can be returned (handle)   |
| 17       |                  | Returns one of the following:   |
| 18       |                  | • PMIX_SUCCESS, indicating that the request was processed and returned <i>success</i> . Details of the  |
| 19       |                  | result will be returned in the <i>results</i> array   |
| 20       |                  | • a PMIx error constant indicating either an error in the input or that the request was refused   |
|          |                  | Required Attributes   |
| 21       |                  | PMIx libraries that choose not to support this operation <i>must</i> return   |
| 22       |                  | PMIX_ERR_NOT_SUPPORTED when the function is called.   |
| 23<br>24 |                  | 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 <i>munge</i> server). |
| 25       |                  | Implementations that support the operation but cannot directly process the client's request must  |
| 26       |                  | pass any attributes that are provided by the client to the host environment for processing. In  |
| 27       |                  | addition, the following attributes are required to be included in the <i>info</i> array passed from the PMIx  |
| 28       |                  | library to the host environment:  |
| 29       |                  | <pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>   |
| 30       |                  | Effective user id.  |
| 31       |                  | <pre>PMIX_GRPID "pmix.egid" (uint32_t)</pre>  |
| 32       |                  | Effective group id.   |
|          |                  |   |
|          |                  |   |

#### Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX\_TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 the target process from ever exposing its data. 5 Advice to PMIx library implementers We recommend that implementation of the PMIX\_TIMEOUT attribute be left to the host 6 7 environment due to race condition considerations between completion of the operation versus 8 internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT 9 directly in the PMIx server library must take care to resolve the race condition and should avoid 10 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not 11 created. **Description** 12 13 Request validation of a credential by the PMIx server library or the host environment. 10.2.2 PMIx\_Validate\_credential\_nb Summary 15

Request validation of a credential by the PMIx server library or the host environment

| 1                    | Format  |
|----------------------|---|
| <i>PMIx v3.0</i>     |   |
| 2                    | pmix_status_t   |
| 3<br>4               | PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,   |
| <del>4</del><br>5    | <pre>const pmix_info_t info[], size_t ninfo, pmix_validation_cbfunc_t cbfunc,</pre>   |
| 6                    | void *cbdata)   |
| •                    | C   |
| 7                    | IN cred   |
| 8                    | Pointer to <pre>pmix_byte_object_t</pre> containing the credential (handle)   |
| 9                    | IN info   |
| 10                   | Array of <pre>pmix_info_t</pre> structures (array of handles)   |
| 11                   | IN ninfo  |
| 12                   | Number of elements in the <i>info</i> array (size_t)  |
| 13                   | IN cbfunc   |
| 14                   | Callback function to return result ( pmix_validation_cbfunc_t function reference)   |
| 15                   | IN cbdata   |
| 16                   | Data to be passed to the callback function (memory reference)   |
| 17                   | Returns one of the following:   |
| 18<br>19             | • <b>PMIX_SUCCESS</b> , indicating that the request has been communicated to the local PMIx server - result will be returned in the provided <i>cbfunc</i>  |
| 20<br>21             | • a PMIx error constant indicating either an error in the input or that the request is unsupported - the <i>cbfunc</i> will <i>not</i> be called  |
|                      | Required Attributes   |
| 22<br>23             | PMIx libraries that choose not to support this operation <i>must</i> return <b>PMIX_ERR_NOT_SUPPORTED</b> when the function is called.  |
| 24<br>25             | 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 <i>munge</i> server).   |
| 26<br>27<br>28<br>29 | 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 <i>info</i> array passed from the PMIx library to the host environment: |
| 30<br>31             | PMIX_USERID "pmix.euid" (uint32_t) Effective user id.   |
| 32<br>33             | PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.  |

# The following attributes are optional for host environments that support this operation: PMIX\_TIMEOUT "pmix.timeout" (int) Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data. Advice to PMIx library implementers We recommend that implementation of the PMIX\_TIMEOUT attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support PMIX\_TIMEOUT directly in the PMIx server library must take care to resolve the race condition and should avoid passing PMIX\_TIMEOUT to the host environment so that multiple competing timeouts are not created.

#### **Description**

Request validation of a credential by the PMIx server library or the host environment.

#### **CHAPTER 11**

1 2

3

4 5

6 7

8

9

10

12 13

16

24

# **Server-Specific Interfaces**

The RM daemon that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes generating regular expressions, registering information to be passed to client processes, and requesting information on behalf of a remote process. Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

# 11.1 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

# 14 11.1.1 PMIx\_generate\_regex

#### 15 **Summary**

Generate a compressed representation of the input string.

- 21 String to process (string)
  22 **OUT** output
- Compressed representation of *input* (array of bytes)
  - Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

#### Description

1

3

5

6

7

8

9

10

11

12

13 14

15

16 17

18 19

20

22

23

24

25

26

28 29

30

Given a comma-separated list of *input* values, generate a reduced size representation of the input that can be passed down to the PMIx server library's **PMIx\_server\_register\_nspace** API for parsing. The order of the individual values in the *input* string is preserved across the operation. The caller is responsible for releasing the returned data.

The precise compressed representations will be implementation specific. However, all PMIx implementations are required to include a **NULL**-terminated string in the output representation that can be printed for diagnostic purposes.

#### Advice to PMIx server hosts —

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with "pmix:\0" might indicate that the representation is a PMIx-defined regular expression represented as a NULL-terminated string following the "pmix:\0" prefix. In contrast, an output starting with "blob:\0" might indicate a compressed binary array follows the prefix.

Communicating the resulting output should be done by first packing the returned expression using the <code>PMIx\_Data\_pack</code>, declaring the input to be of type <code>PMIX\_REGEX</code>, and then obtaining the resulting blob to be communicated using the <code>PMIX\_DATA\_BUFFER\_UNLOAD</code> macro. The reciprocal method can be used on the remote end prior to passing the regex into <code>PMIx\_server\_register\_nspace</code>. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

# 11.1.2 PMIx generate ppn

#### Summary

Generate a compressed representation of the input identifying the processes on each node.

#### Format

PMIx v1.0

pmix\_status\_t PMIx\_generate\_ppn(const char \*input, char \*\*ppn)

#### IN input

27 String to process (string)

#### OUT ppn

Compressed representation of *input* (array of bytes)

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

#### Description

The input shall consist of a semicolon-separated list of ranges representing the ranks of processes on each node of the job - e.g., "1-4;2-5;8,10,11,12;6,7,9". Each field of the input must correspond to the node name provided at that position in the input to **PMIx\_generate\_regex**. Thus, in the example, ranks 1-4 would be located on the first node of the comma-separated list of names provided to **PMIx\_generate\_regex**, and ranks 2-5 would be on the second name in the list.

#### Advice to PMIx server hosts -

The returned representation may be an arbitrary array of bytes as opposed to a valid NULL-terminated string. However, the method used to generate the representation shall be identified with a colon-delimited string at the beginning of the output. For example, an output starting with "pmix:" indicates that the representation is a PMIx-defined regular expression represented as a NULL-terminated string. In contrast, an output starting with "blob:\0size=1234:" is a compressed binary array.

Communicating the resulting output should be done by first packing the returned expression using the <code>PMIx\_Data\_pack</code>, declaring the input to be of type <code>PMIX\_REGEX</code>, and then obtaining the blob to be communicated using the <code>PMIX\_DATA\_BUFFER\_UNLOAD</code> macro. The pack/unpack routines will ensure proper handling of the data based on the regex prefix.

# 11.1.3 PMIx\_server\_register\_nspace

#### Summary

Setup the data about a particular namespace.

#### Format

PMIx v1.0

1

3

5

6

7

8

9

10

11 12

13

14

15

16

18 19

20

21

22 23

24 25

26

27

28

29

30 31

32

pmix\_status\_t

pmix\_info\_t info[], size\_t ninfo,

pmix\_op\_cbfunc\_t cbfunc, void \*cbdata)

IN nspace

Character array of maximum size **PMIX\_MAX\_NSLEN** containing the namespace identifier (string)

IN nlocalprocs

number of local processes (integer)

IN info

Array of info structures (array of handles)

| 1<br>2<br>3<br>4<br>5<br>6 | Number of elements in the <i>info</i> array (integer)  IN cbfunc Callback function pmix_op_cbfunc_t (function reference)  IN cbdata Data to be passed to the callback function (memory reference)  |
|----------------------------|--|
| 7                          | Returns one of the following:  |
| 8<br>9<br>0                | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.  |
| 1<br>2                     | • PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called  |
| 3<br>4                     | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called  |
|                            | ▼  |
| 5                          | The following attributes are required to be supported by all PMIx libraries:   |
| 6<br>7<br>8<br>9           | <pre>PMIX_REGISTER_NODATA "pmix.reg.nodata" (bool)     Registration is for this namespace only, do not copy job data - this attribute is not accessed using the PMIx_Get</pre>   |
| 20                         | Host environments are required to provide the following attributes:  |
| <u>!</u> 1                 | • for the session containing the given namespace:  |
| 22<br>23<br>24<br>25<br>26 | - PMIX_UNIV_SIZE "pmix.univ.size" (uint32_t) Number of allocated slots in a session - each slot may or may not be occupied by an executing process. Note that this attribute is the equivalent to the combination of PMIX_SESSION_INFO_ARRAY with the PMIX_MAX_PROCS entry in the array - it is included in the Standard for historical reasons.   |
| 27                         | • for the given namespace:   |
| 28<br>29                   | <ul> <li>- PMIX_JOBID "pmix.jobid" (char*)</li> <li>Job identifier assigned by the scheduler.</li> </ul>   |
| 90<br>91<br>92<br>93<br>94 | <ul> <li>PMIX_JOB_SIZE "pmix.job.size" (uint32_t)</li> <li>Total number of processes in this job across all contained applications. Note that this value can be different from PMIX_MAX_PROCS. For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running job</li> </ul> |
|                            |  |

| 1<br>2                     | on-they-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.   |
|----------------------------|---|
| 3<br>4<br>5<br>6           | <ul> <li>PMIX_MAX_PROCS "pmix.max.size" (uint32_t)</li> <li>Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description.</li> </ul>   |
| 7<br>8                     | <ul> <li>PMIX_NODE_MAP "pmix.nmap" (char*)</li> <li>Regular expression of nodes - see 11.1.3.1 for an explanation of its generation.</li> </ul>   |
| 9<br>10<br>11              | <ul> <li>PMIX_PROC_MAP "pmix.pmap" (char*)</li> <li>Regular expression describing processes on each node - see 11.1.3.1 for an explanation of its generation.</li> </ul>  |
| 12                         | • for its own node:   |
| 13<br>14                   | <ul> <li>- PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)</li> <li>Number of processes in this job or application on this node.</li> </ul>  |
| 15<br>16<br>17             | <ul> <li>PMIX_LOCAL_PEERS "pmix.lpeers" (char*)</li> <li>Comma-delimited list of ranks on this node within the specified namespace - referenced using PMIX_RANK_WILDCARD.</li> </ul>  |
| 18<br>19<br>20             | <ul> <li>PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)</li> <li>Colon-delimited cpusets of local peers within the specified namespace - referenced using PMIX_RANK_WILDCARD.</li> </ul>   |
| 21                         | • for each process in the given namespace:  |
| 22<br>23                   | - PMIX_RANK "pmix.rank" (pmix_rank_t) Process rank within the job.  |
| 24<br>25                   | - PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)  Local rank on this node within this job.   |
| 26<br>27                   | <pre>- PMIX_NODE_RANK "pmix.nrank" (uint16_t) Process rank on this node spanning all jobs.</pre>  |
| 28<br>29<br>30<br>31<br>32 | <ul> <li>PMIX_NODEID "pmix.nodeid" (uint32_t)</li> <li>Node identifier expressed as the node's index (beginning at zero) in an array of nodes within the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME of the node - i.e., users can interchangeably reference the same location using either the PMIX_HOSTNAME or corresponding PMIX_NODEID.</li> </ul> |
| 33<br>34                   | If more than one application is included in the namespace, then the host environment is also required to provide the following attributes:  |
| 35                         | • for each application:   |
| 36                         | - PMTY APPNIM "nmix appnum" (uint32 t)  |

| 1                    | Application number within the job.   |
|----------------------|--|
| 2<br>3<br>4          | <ul> <li>PMIX_APPLDR "pmix.aldr" (pmix_rank_t)</li> <li>Lowest rank in this application within this job - referenced using</li> <li>PMIX_RANK_WILDCARD .</li> </ul>  |
| 5<br>6               | <pre>- PMIX_APP_SIZE "pmix.app.size" (uint32_t)     Number of processes in this application.</pre>   |
| 7                    | • for each process:  |
| 8<br>9               | - PMIX_APP_RANK "pmix.apprank" (pmix_rank_t) Process rank within this application.   |
| 10<br>11             | - PMIX_APPNUM "pmix.appnum" (uint32_t) Application number within the job.  |
|                      | ▼ Optional Attributes  |
| 12                   | The following attributes may be provided by host environments:   |
| 13                   | • for the session containing the given namespace:  |
| 14<br>15             | <pre>- PMIX_SESSION_ID "pmix.session.id" (uint32_t) Session identifier - referenced using PMIX_RANK_WILDCARD.</pre>  |
| 16                   | • for the given namespace:   |
| 17<br>18             | <ul> <li>- PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)</li> <li>Name of the namespace to use for this PMIx server.</li> </ul>   |
| 19<br>20             | - PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t) Rank of this PMIx server  |
| 21<br>22             | <ul> <li>- PMIX_NPROC_OFFSET "pmix.offset" (pmix_rank_t)</li> <li>Starting global rank of this job - referenced using PMIX_RANK_WILDCARD.</li> </ul>   |
| 23<br>24<br>25       | <ul> <li>PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)</li> <li>Comma-delimited list of all nodes in this allocation regardless of whether or not they currently host processes - referenced using PMIX_RANK_WILDCARD.</li> </ul>           |
| 26<br>27             | - PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t) Number of applications in this job.  |
| 28<br>29<br>30<br>31 | <ul> <li>PMIX_MAPBY "pmix.mapby" (char*)</li> <li>Process mapping policy - when accessed using PMIx_Get , use the</li> <li>PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace</li> </ul> |
| 32                   | - PMIX_RANKBY "pmix.rankby" (char*)  |

| 1<br>2<br>3      | Process ranking policy - when accessed using <b>PMIx_Get</b> , use the <b>PMIX_RANK_WILDCARD</b> value for the rank to discover the ranking algorithm used for the provided namespace  |
|------------------|--|
| 4<br>5<br>6<br>7 | <ul> <li>PMIX_BINDTO "pmix.bindto" (char*)</li> <li>Process binding policy - when accessed using PMIx_Get , use the</li> <li>PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the provided namespace</li> </ul> |
| 8<br>9           | <ul> <li>PMIX_ANL_MAP "pmix.anlmap" (char*)</li> <li>Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.</li> </ul>   |
| 10               | • for its own node:  |
| 11<br>12         | - PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)  Total available physical memory on this node.   |
| 13<br>14         | - PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)  XML representation of local topology using HWLOC's v1.x format.  |
| 15<br>16         | - PMIX_HWLOC_XML_V2 "pmix.hwlocxm12" (char*)  XML representation of local topology using HWLOC's v2.x format.  |
| 17<br>18<br>19   | - PMIX_LOCALLDR "pmix.lldr" (pmix_rank_t)  Lowest rank on this node within this job - referenced using PMIX_RANK_WILDCARD.   |
| 20<br>21         | <ul> <li>- PMIX_NODE_SIZE "pmix.node.size" (uint32_t)</li> <li>Number of processes across all jobs on this node.</li> </ul>  |
| 22<br>23<br>24   | - PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)  Array of pmix_proc_t of all processes on the specified node - referenced using PMIX_RANK_WILDCARD.   |
| 25               | • for each process in the given namespace:   |
| 26<br>27         | - PMIX_PROCID "pmix.procid" (pmix_proc_t) Process identifier   |
| 28<br>29         | <ul> <li>- PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t)</li> <li>Process rank spanning across all jobs in this session.</li> </ul>  |
| 30<br>31<br>32   | <ul> <li>PMIX_HOSTNAME "pmix.hname" (char*)</li> <li>Name of the host (e.g., where a specified process is running, or a given device is located).</li> </ul>   |
| 33<br>34<br>35   | Attributes not directly provided by the host environment may be derived by the PMIx server library from other required information and included in the data made available to the server library's clients.                                  |
|                  |  |

The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the namespace. The PMIx server library may utilize this information to customize the environment to fit that model (e.g., adding environmental variables specified by the corresponding standard for that model): • PMIX PROGRAMMING MODEL "pmix.pgm.model" (char\*) Programming model being initialized (e.g., "MPI" or "OpenMP") • PMIX MODEL LIBRARY\_NAME "pmix.mdl.name" (char\*) Programming model implementation ID (e.g., "OpenMPI" or "MPICH") • PMIX MODEL LIBRARY VERSION "pmix.mld.vrs" (char\*) Programming model version string (e.g., "2.1.1") **Description** Pass job-related information to the PMIx server library for distribution to local client processes. Advice to PMIx server hosts -

Host environments are required to execute this operation prior to starting any local application process within the given namespace.

The PMIx server must register all namespaces that will participate in collective operations with local processes. This means that the server must register a namespace even if it will not host any local processes from within that namespace if any local process of another namespace might at some point perform an operation involving one or more processes from the new namespace. This is necessary so that the collective operation can identify the participants and know when it is locally complete.

The caller must also provide the number of local processes that will be launched within this namespace. This is required for the PMIx server library to correctly handle collectives as a collective operation call can occur before all the local processes have been started.

#### Advice to users -

The number of local processes for any given namespace is generally fixed at the time of application launch. Calls to <code>PMIx\_Spawn</code> result in processes launched in their own namespace, not that of their parent. However, it is possible for processes to *migrate* to another node via a call to <code>PMIx\_Job\_control\_nb</code>, thus resulting in a change to the number of local processes on both the initial node and the node to which the process moved. It is therefore critical that applications not migrate processes without first ensuring that <code>PMIx-based</code> collective operations are not in progress, and that no such operations be initiated until process migration has completed.

#### 11.1.3.1 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it is describes how information provided in the *info* parameter of the PMIx\_server\_register\_nspace shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

#### Advice to PMIx server hosts -

Creating the *info* array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single <code>pmix\_info\_t</code> structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final *info* array can be allocated based on the number of elements on the list, and then the values in the list object <code>pmix\_info\_t</code> structures transferred to the corresponding array element utilizing the <code>PMIX\_INFO\_XFER</code> macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a **session** or **job**. PMIx provides several tools to facilitate this operation, beginning by constructing an argy-like array of node names. This array is then passed to the **PMIx\_generate\_regex** function to create a regular expression parseable by the PMIx server library, as shown below:

```
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
             * name to the array */
27
28
            for (n=0; n < num_nodes; n++)</pre>
29
                /* filter the nodes to ignore those not included
                 * in the target range (session, job, etc.). In
30
                 * this example, all nodes are accepted */
31
32
                PMIX_ARGV_APPEND(&nodes, node[n]->name);
33
34
35
            /* join into a comma-delimited string */
            nodelist = PMIX ARGV JOIN(nodes, ',');
36
37
```

```
/* release the array */
PMIX_ARGV_FREE(nodes);

/* generate regex */
rc = PMIx_generate_regex(nodelist, &regex);

/* release list */
free(nodelist);

/* pass the regex as the value to the PMIX_NODE_MAP key */
PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
/* release the regex */
free(regex);
```

Changing the filter criteria allows the construction of node maps for any level of information.

A similar method is used to construct the map of processes on each node from the namespace being registered. This may be done for each information level of interest (e.g., to identify the process map for the entire job or for each application in the job) by changing the search criteria. An example is shown below for the case of creating the process map for a job:

```
char **ndppn;
char rank[30];
char **ppnarray = NULL;
char *ppn;
char *localranks;
char *regex;
size t n, m;
pmix_status_t rc;
pmix_info_t info;
/* loop over an array of nodes */
for (n=0; n < num_nodes; n++)</pre>
    /* for each node, construct an array of ranks on that node */
    ndppn = NULL;
    for (m=0; m < node[n]->num_procs; m++)
        /* ignore processes that are not part of the target job */
        if (!PMIX_CHECK_NSPACE(targetjob, node[n]->proc[m].nspace))
            continue;
        snprintf(rank, 30, "%d", node[n]->proc[m].rank);
        PMIX ARGV APPEND (&ndppn, rank);
```

```
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
```

```
/* convert the array into a comma-delimited string of ranks */
    localranks = PMIX ARGV JOIN(ndppn, ',');
    /* release the local array */
    PMIX ARGV FREE (ndppn);
    /* add this node's contribution to the overall array */
    PMIX ARGV APPEND (&ppnarray, localranks);
    /* release the local list */
    free(localranks);
/* join into a semicolon-delimited string */
ppn = PMIX_ARGV_JOIN(ppnarray, ';');
/* release the array */
PMIX_ARGV_FREE (ppnarray);
/* generate ppn regex */
rc = PMIx generate ppn(ppn, &regex);
/* release list */
free (ppn);
/* pass the regex as the value to the PMIX PROC MAP key */
PMIX INFO LOAD (&info, PMIX PROC MAP, regex, PMIX STRING);
/* release the regex */
free(regex);
```

Note that the PMIX\_NODE\_MAP and PMIX\_PROC\_MAP attributes are linked in that the order of entries in the process map must match the ordering of nodes in the node map - i.e., there is no provision in the PMIx process map regular expression generator/parser pair supporting an out-of-order node or a node that has no corresponding process map entry (e.g., a node with no processes on it). Armed with these tools, the registration *info* array can be constructed as follows:

Session-level information includes all session-specific values. In many cases, only two values (
 <u>PMIX\_SESSION\_ID</u> and <u>PMIX\_UNIV\_SIZE</u>) are included in the registration array. Since
 both of these values are session-specific, they can be specified independently - i.e., in their own
 <u>pmix\_info\_t</u> elements of the *info* array. Alternatively, they can be provided as a
 <u>pmix\_data\_array\_t</u> array of <u>pmix\_info\_t</u> using the <u>PMIX\_SESSION\_INFO\_ARRAY</u>
 attribute and identifed by including the <u>PMIX\_SESSION\_ID</u> attribute in the array - this is
 required in cases where non-specific attributes (e.g., <u>PMIX\_NUM\_NODES</u> or <u>PMIX\_NODE\_MAP</u>

 ) are passed to describe aspects of the session. Note that the node map can include nodes not used by the job being registered as no corresponding process map is specified.

The *info* array at this point might look like (where the labels identify the corresponding attribute - e.g., "Session ID" corresponds to the **PMIX\_SESSION\_ID** attribute):

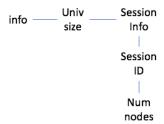


Figure 11.1.: Session-level information elements

Upon conclusion of this step, the *info* array might look like:

Note that in this example, **PMIX\_NUM\_NODES** is not required as that information is contained in the **PMIX\_NODE\_MAP** attribute. Similarly, **PMIX\_JOB\_SIZE** is not technically required as that information is contained in the **PMIX\_PROC\_MAP** when combined with the corresponding node map - however, there is no issue with including the job size as a separate entry.

The example also illustrates the hierarchical use of the PMIX\_NODE\_INFO\_ARRAY attribute. In this case, we have chosen to pass several job-related values for each node - since those values are non-unique across the job, they must be passed in a node-info container. Note that the choice of what information to pass into the PMIx server library versus what information to derive from other values at time of request is left to the host environment. PMIx implementors in turn may, if they choose, pre-parse registration data to create expanded views (thus enabling faster response to requests at the expense of memory footprint) or to compress views into tighter representations (thus trading minimized footprint for longer response times).

Application-level information includes all application-specific values such as PMIX\_APP\_SIZE and PMIX\_APPLDR. If the job contains only a single application, then the application-specific values can be specified independently - i.e., in their own pmix\_info\_t



2

3

4

5



15

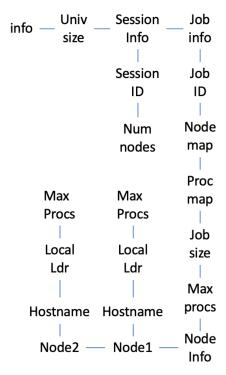


Figure 11.2.: Job-level information elements

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,

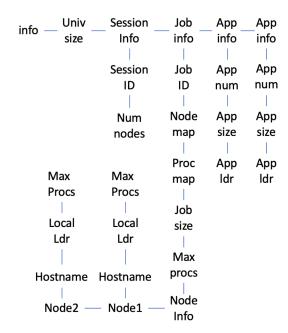


Figure 11.3.: Application-level information elements

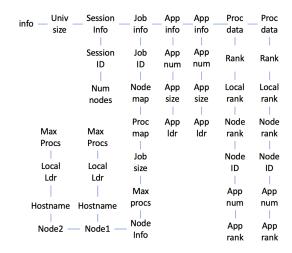


Figure 11.4.: Process-level information elements

the values included in this level are unique to it and can be specified independently - i.e., in their own pmix\_info\_t elements of the *info* array. Alternatively, they can be provided as a pmix\_data\_array\_t array of pmix\_info\_t using the PMIX\_NODE\_INFO\_ARRAY

attribute - this is required in cases where non-specific attributes are passed to describe aspects of the node, or where values for multiple nodes are being provided.

The node-level information requires two elements that must be constructed in a manner similar to that used for the node map. The **PMIX\_LOCAL\_PEERS** value is computed based on the processes on the local node, filtered to select those from the job being registered, as shown below using the tools provided by PMIx:

```
C _____
char **ndppn = NULL;
char rank[30];
char *localranks:
size t m;
pmix info t info;
for (m=0; m < mynode->num_procs; m++)
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob, mynode->proc[m].nspace))
        continue;
    snprintf(rank, 30, "%d", mynode->proc[m].rank);
    PMIX_ARGV_APPEND(&ndppn, rank);
/* convert the array into a comma-delimited string of ranks */
localranks = PMIX ARGV JOIN(ndppn, ',');
/* release the local array */
PMIX ARGV FREE (ndppn);
/* pass the string as the value to the PMIX_LOCAL_PEERS key */
PMIX INFO LOAD (&info, PMIX LOCAL PEERS, localranks, PMIX STRING);
/* release the list */
free(localranks);
```

The PMIX\_LOCAL\_CPUSETS value is constructed in a similar manner. In the provided example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a comma-delimited string of processor IDs) of the processors assigned to each process has previously been generated and stored on the process description. Thus, the value can be constructed as shown below:

1

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

```
C
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_NSPACE(targetjob, mynode->proc[m].nspace))
9
                       continue;
10
                  PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
11
12
13
              /* convert the array into a colon-delimited string */
              localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
14
              /* release the local array */
15
16
              PMIX ARGV FREE (ndcpus);
17
18
              /* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
              PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);
19
              /* release the list */
20
21
              free(localcpus);
22
                                              C
23
              Note that for efficiency, these two values can be computed at the same time.
```

The final *info* array might therefore look like the diagram in 11.5:

# 25 11.1.4 PMIx\_server\_deregister\_nspace

#### Summary

Deregister a namespace.

#### Format

PMIx v1.0

24

26 27

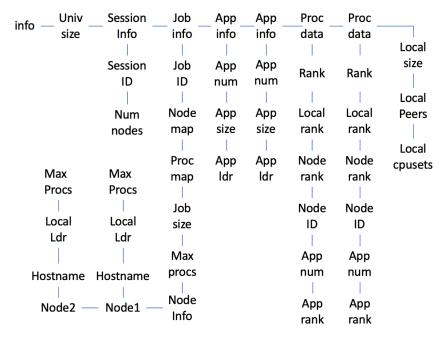


Figure 11.5.: Final information array

#### Description

Deregister the specified *nspace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library must not invoke the callback function prior to returning from the API.

# 11.1.5 PMIx\_server\_register\_client

#### Summary Register a client process with the PMIx server library. 3 **Format** *PMIx v1.0* 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) 10 IN proc pmix\_proc\_t structure (handle) 11 IN uid 12 user id (integer) 13 14 IN qid 15 group id (integer) IN server\_object 16 (memory reference) 17 IN cbfunc 18 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 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 26 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and 27 returned success - the cbfunc will not be called 28 • a PMIx error constant indicating either an error in the input or that the request was immediately 29 processed and failed - the *cbfunc* will not be called **Description** 30 Register a client process with the PMIx server library. 31 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

object to the host server during subsequent calls related to that client, such as a

pmix\_server\_client\_connected\_fn\_t function. This allows the host server to access 1 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. 11.1.6 PMIx server deregister client 12 Summary Deregister a client and purge all data relating to it. 13 Format 14 PMIx v1.0void 15 16 PMIx server deregister client(const pmix proc t \*proc, 17 pmix op cbfunc t cbfunc, void \*cbdata) IN 18 proc pmix\_proc\_t structure (handle) 19 20 Callback function **pmix** op **cbfunc t** (function reference) 21 cbdata 22 IN 23 Data to be passed to the callback function (memory reference) **Description** 24 The PMIx\_server\_deregister\_nspace API will delete all client information for that 25 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.

# 11.1.7 PMIx server setup fork

# Summary Setup the environment of a child process to be forked by the host. Format

PMIx v1.0

5

6 7

12

13 14

15

16

17

18

19

20

23

8 IN proc

9 proc\_t structure (handle)
10 IN env
11 Environment array (array of strings)

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

# Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

#### Advice to PMIx server hosts —

Host environments are required to execute this operation prior to starting the client process.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via PMIx\_Spawn) plus other values (e.g., variables required to properly initialize the client's fabric library).

# 21 11.1.8 PMIx server dmodex request

# 22 Summary

Define a function by which the host server can request modex data from the local PMIx server.

#### **Format**

PMIx v1.0

C

IN proc

pmix proc t structure (handle)

IN cbfunc

Callback function pmix\_dmodex\_response\_fn\_t (function reference)

IN cbdata

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

Returns one of the following:

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

#### **Description**

Define a function by which the host server can request modex data from the local PMIx server. Traditional wireup procedures revolve around the per-process posting of data (e.g., location and endpoint information) via the PMIx\_Put and PMIx\_Commit functions followed by a PMIx\_Fence barrier that globally exchanges the posted information. However, the barrier operation represents a signficant time impact at large scale.

PMIx supports an alternative wireup method known as *Direct Modex* that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the PMIx\_server\_dmodex\_request function, receiving the response in the provided *cbfunc* once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

#### Advice to users -

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

| 1                | 11.1.9    | PMIx_server_setup_application   |
|------------------|-----------|---|
| 2<br>3<br>4      |           | Summary  Provide a function by which the resource manager can request application-specific setup data prior to launch of a job. |
| 5                |           | Format  |
|                  | PMIx v2.0 | <b>▼</b>  |
| 6<br>7<br>8<br>9 |           | <pre>pmix_status_t PMIx_server_setup_application(const pmix_nspace_t nspace,</pre>  |
| 10               |           | void *cbdata)   |
|                  |           | C   |
| 11               |           | IN nspace   |
| 12               |           | namespace (string)  |
| 13               |           | IN info   |
| 14               |           | Array of info structures (array of handles)  IN ninfo   |
| 15<br>16         |           | Number of elements in the <i>info</i> 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 <i>cbfunc</i> 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 <i>cbfunc</i> will not be called                          |
|                  |           | Required Attributes   |
| 26               |           | PMIx libraries that support this operation are required to support the following:   |
| 27               |           | <pre>PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)</pre>  |
| 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<br>3<br>4<br>5   | PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)  Array of pmix_info_t describing requested fabric resources. This must include at least:  PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and  PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.   |
| 6<br>7<br>8<br>9<br>10<br>11<br>12<br>13<br>14<br>15<br>16<br>17<br>18 | The key to be used when accessing this requested fabric 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 fabric 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_FABRIC_TYPE - the type of resources provided; PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH - the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if PMIX_INFO_REQD was not set in the request. |
| 20<br>21   | <pre>PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t) Fabric security key</pre>  |
| 22<br>23   | <pre>PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*) Type of desired transport (e.g., "tcp", "udp")</pre>  |
| 24<br>25   | PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)  ID string for the NIC (aka plane) to be used for this allocation (e.g., CIDR for Ethernet)  |
| 26<br>27   | <pre>PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)     Number of endpoints to allocate per process</pre>   |
| 28<br>29   | PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)  Number of endpoints to allocate per node  Optional Attributes   |
| 30   | PMIx libraries that support this operation may support the following:  |
| 31<br>32   | PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)  Mbits/sec.   |
| 33<br>34   | <pre>PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)    Quality of service level.</pre>  |
| 35   | <pre>PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)</pre>  |

| 1                   | Time in seconds.   |
|---------------------|--|
| 2<br>3<br>4<br>5    | The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the application. The PMIx server library may utilize this information to harvest/forward model-specific environmental variables, record the programming model associated with the application, etc.   |
| 6<br>7              | <ul> <li>PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*)</li> <li>Programming model being initialized (e.g., "MPI" or "OpenMP")</li> </ul>   |
| 8<br>9              | • PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*)  Programming model implementation ID (e.g., "OpenMPI" or "MPICH")  |
| 0                   | • PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)  Programming model version string (e.g., "2.1.1")  |
| 2<br>3<br>4         | <b>Description</b> 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  |
| 6<br>  7<br>  8     | Host environments are required to execute this operation prior to launching a job. In addition to supported directives, the <i>info</i> array must include a description of the <b>job</b> using the <b>PMIX_NODE_MAP</b> and <b>PMIX_PROC_MAP</b> attributes.   |
| 9<br>20<br>21<br>22 | 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   |
| 23                  | Support for har vesting of chynolinichtar variables and providing of focal configuration information   |

# 11.1.10 PMIx\_Register\_attributes

# Summary

26

27

Register host environment attribute support for a function.

Format 1 PMIx v4.0 2 pmix status t 3 PMIx Register attributes (char \*function, 4 pmix regattr t attrs[], size\_t nattrs) 5 6 IN function 7 String name of function (string) 8 IN attrs 9 Array of pmix\_regattr\_t describing the supported attributes (handle) 10 IN nattrs Number of elements in *attrs* (size\_t) 11 12 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. **Description** 13 The PMIx\_Register\_attributes function is used by the host environment to register with 14 its PMIx server library the attributes it supports for each pmix server module t function. 15 16 The function is the string name of the server module function (e.g., "register events", 17 "validate credential", or "allocate") whose attributes are being registered. See the **pmix regattr** t entry for a description of the *attrs* array elements. 18 19 Note that the host environment can also query the library (using the PMIx Query\_info\_nb 20 API) for its attribute support both at the server, client, and tool levels once the host has executed **PMIx\_server\_init** since the server will internally register those values. 21 Advice to PMIx server hosts — Host environments are strongly encouraged to register all supported attributes immediately after 22 23 initializing the library to ensure that user requests are correctly serviced.

#### Advice to PMIx library implementers —

PMIx implementations are *required* to register all internally supported attributes for each API during initialization of the library (i.e., when the process calls their respective PMIx init function). Specifically, the implementation *must not* register supported attributes upon first call to a given API as this would prevent users from discovering supported attributes prior to first use of an API.

It is the implementation's responsibility to associate registered attributes for a given **pmix\_server\_module\_t** function with their corresponding user-facing API. Supported attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by the level (see 3.4.33) at which the attribute is supported.

Note that attributes can/will be registered on an API for each level. It is *required* that the implementation support user queries for supported attributes on a per-level basis. Duplicate registrations at the *same* level for a function *shall* return an error - however, duplicate registrations at *different* levels *shall* be independently tracked.

# 11.1.11 PMIx\_server\_setup\_local\_support

#### Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

#### **Format**

```
PMIx v2.0
```

1

2

3

5

6 7

8

9

10

11

12

14 15

16

17

25

26 27

28

29

30

31

32

```
pmix_status_t

PMIx_server_setup_local_support(const pmix_nspace_t nspace,

pmix_info_t info[], size_t ninfo,

pmix_op_cbfunc_t cbfunc,

void *cbdata);
```

```
23 IN nspace
24 Namespace
```

Namespace (string)

IN info

Array of info structures (array of handles)

IN ninfo

Number of elements in the *info* array (size\_t)

IN cbfunc

Callback function pmix\_op\_cbfunc\_t (function reference)

IN cbdata

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

| Returns one | of the | following: |
|-------------|--------|------------|
|-------------|--------|------------|

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

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a fabric library might need to setup the local driver for "instant on" addressing. The data provided in the *info* array is the data returned to the host RM by the callback function executed as a result of a call to **PMIx server setup application**.

#### Advice to PMIx server hosts —

Host environments are required to execute this operation prior to starting any local application processes from the specified namespace.

# 17 11.1.12 PMIx\_server\_IOF\_deliver

#### Summary

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

#### **Format**

```
PMIx v3.0
```

| ı        | IIA SOULCE   |
|----------|--|
| 2        | Pointer to <pre>pmix_proc_t</pre> identifying source of the IO (handle)  |
| 3        | IN channel   |
| 4        | IO channel of the data ( pmix_iof_channel_t )  |
| 5        | IN bo  |
| 6        | Pointer to <pre>pmix_byte_object_t</pre> containing the payload to be delivered (handle)   |
| 7        | IN info  |
| 8        | Array of <pre>pmix_info_t</pre> metadata describing the data (array of handles)  |
| 9        | IN ninfo   |
| 10       | Number of elements in the <i>info</i> array (size_t)   |
| 11       | IN cbfunc  |
| 12       | Callback function <pre>pmix_op_cbfunc_t</pre> (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 - resul   |
| 17       | will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback  |
| 18       | function prior to returning from the API.  |
| 19       | • PMTY OPERATION CUCCEPPED indicating that the request was immediately processed and   |
| 19<br>20 | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and<br/>returned success - the cbfunc will not be called</li> </ul> |
| 20       | returned success - the cojune will not be cancu  |
| 21       | • a PMIx error constant indicating either an error in the input or that the request was immediately  |

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.

# 8 11.1.13 PMIx\_server\_collect\_inventory

processed and failed - the cbfunc will not be called

## Summary

22

23

24

25

26

27

29

30

Collect inventory of resources on a node

#### Format 1 PMIx v3.0 2 pmix status t 3 PMIx\_server\_collect\_inventory(const pmix\_info\_t directives[], 4 size t ndirs, 5 pmix\_info\_cbfunc\_t cbfunc, 6 void \*cbdata); 7 IN directives Array of **pmix\_info\_t** directing the request (array of handles) 8 9 IN ndirs Number of elements in the *directives* array (size\_t) 10 IN 11 12 Callback function to return collected data (pmix info cbfunc t function reference) IN cbdata 13 14 Data to be passed to the callback function (memory reference) 15 Returns **PMIX** SUCCESS or a negative value corresponding to a PMIx error constant. In the event 16 the function returns an error, the *cbfunc* will not be called. **Description** 17 Provide a function by which the host environment can request its PMIx server library collect an 18 inventory of local resources. Supported resources depends upon the PMIx implementation, but may 19 20 include the local node topology and fabric interfaces. Advice to PMIx server hosts — 21 This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested 22 information. Inventory collection is expected to be a rare event – at system startup and upon 23 command from a system administrator. Inventory updates are expected to initiate a smaller 24 operation involving only the changed information. For example, replacement of a node would 25 generate an event to notify the scheduler with an inventory update without invoking a global 26 inventory operation.

## 11.1.14 PMIx\_server\_deliver\_inventory

#### Summary

Pass collected inventory to the PMIx server library for storage

28

|                  | C   |
|------------------|---|
| <i>PMIx v3.0</i> |   |
| 2                | pmix_status_t   |
| 3                | <pre>PMIx_server_deliver_inventory(const pmix_info_t info[],</pre>  |
| 4                | size_t ninfo,   |
| 5                | <pre>const pmix_info_t directives[],</pre>  |
| 6                | size_t ndirs,   |
| 7                | <pre>pmix_op_cbfunc_t cbfunc,</pre>   |
| 8                | <pre>void *cbdata);</pre>   |
|                  | C   |
| 9                | IN info   |
| 10               | Array of pmix_info_t containing the inventory (array of handles)  |
| 11               | IN ninfo  |
| 12               | Number of elements in the <i>info</i> array (size_t)  |
| 13               | IN directives   |
| 14               | Array of pmix_info_t directing the request (array of handles)   |
| 15               | IN ndirs  |
| 16               | Number of elements in the <i>directives</i> array (size_t)  |
| 17               | IN cbfunc   |
| 18               | Callback function pmix_op_cbfunc_t (function reference)   |
| 19               | IN cbdata   |
| 20               | Data to be passed to the callback function (memory reference)   |
| 21               | 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 <i>cbfunc</i> . 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 <i>success</i> - the <i>cbfunc</i> 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 <i>cbfunc</i> will not be called   |
| 29               | Description   |
| 30               | Provide a function by which the host environment can pass inventory information obtained from a           |
| 31               | node to the PMIx server library for storage. Inventory data is subsequently used by the PMIx server       |
| 32               | library for allocations in response to <b>PMIx_server_setup_application</b> , and may be                  |
| 33               | available to the library's host via the <b>PMIx_Get</b> API (depending upon PMIx implementation).         |
| 34               | The <i>cbfunc</i> callback function will be called once the PMIx server library no longer requires access |
| 35               | to the provided data.   |
| 00               | to the provided data.   |

Format

## 1 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. PMIx implementations are required to return a **PMIX\_ERR\_NOT\_SUPPORTED** status to all calls to functions that require host environment support and are not backed by a corresponding server module entry.

The host RM will provide the function pointers in a <code>pmix\_server\_module\_t</code> structure passed to <code>PMIx\_server\_init</code>. 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

## 20 Summary

2

4

5 6

7

8

9

10

11 12

13

14

15

16 17

18

21

List of function pointers that a PMIx server passes to **PMIx\_server\_init** during startup.

#### 22 Format

```
1
            typedef struct pmix server module 3 0 0 t
2
                /* v1x interfaces */
3
                pmix_server_client_connected_fn_t
                                                      client_connected;
                pmix_server_client_finalized_fn_t
4
                                                      client finalized;
5
                pmix_server_abort_fn_t
                                                      abort;
6
                pmix_server_fencenb_fn_t
                                                      fence nb;
7
                pmix_server_dmodex_req_fn_t
                                                      direct modex;
8
                pmix_server_publish_fn_t
                                                      publish;
9
                pmix_server_lookup_fn_t
                                                      lookup;
                pmix_server_unpublish_fn t
10
                                                      unpublish;
                pmix_server_spawn_fn_t
11
                                                      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;
                /* v3x interfaces */
25
                pmix_server_get_cred_fn_t
26
                                                      get_credential;
27
                pmix_server_validate_cred_fn_t
                                                      validate_credential;
28
                pmix_server_iof_fn_t
                                                      iof pull;
29
                pmix_server_stdin_fn_t
                                                      push_stdin;
30
                /* v4x interfaces */
31
                pmix server grp fn t
                                                      group;
32
                pmix_server_fabric_fn_t
                                                      fabric;
```

## 11.2.2 pmix\_server\_client\_connected\_fn\_t

pmix\_server\_module\_t;

#### Summary

33

35

36

Notify the host server that a client connected to this server.

# Format

PMIx v1.0

C

IN proc

pmix\_proc\_t structure (handle)

IN server\_object

object reference (memory reference)

IN cbfunc

Callback function pmix\_op\_cbfunc\_t (function reference)

IN cbdata

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

#### Returns one of the following:

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

## Description

Notify the host environment that a client has called **PMIx\_Init**. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server\_object parameter will be the value of the server\_object parameter passed to **PMIx\_server\_register\_client** by the host server when registering the connecting client. If provided, an implementation of **pmix\_server\_client\_connected\_fn\_t** is only required to call the callback function designated. A host server can choose to not be notified when clients connect by setting **pmix\_server\_client\_connected\_fn\_t** to **NULL**.

It is possible that only a subset of the clients in a namespace call **PMIx\_Init**. The server's **pmix\_server\_client\_connected\_fn\_t** implementation should not depend on being called once per rank in a namespace or delay calling the callback function until all ranks have connected. However, if a rank makes any PMIx calls, it must first call **PMIx\_Init** and therefore the server's **pmix\_server\_client\_connected\_fn\_t** will be called before any other server functions specific to the rank.

| Adv    | ice to | PM        | Ix ser | ver   | host  | 9   |
|--------|--------|-----------|--------|-------|-------|-----|
| / \U v | וטט נט | 1 1 1 1 1 |        | v C i | 11031 | . • |

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

1

2

5

6

8

9

10

11

12

13

14

15

16

17 18

19

20

21

22

23

24

25 26

27

28

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

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
```

| 1        | IN proc  |
|----------|--|
| 2        | <pre>pmix_proc_t structure identifying the process requesting the abort (handle)</pre>   |
| 3        | <pre>IN server_object</pre>  |
| 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 <i>procs</i> array (integer)   |
| 14       | IN cbfunc  |
| 15       | Callback function <b>pmix_op_cbfunc_t</b> (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 - results and the success of the succes |
| 20<br>21 | will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback functio prior to returning from the API.  |
| 22<br>23 | • <b>PMIX_OPERATION_SUCCEEDED</b> , indicating that the request was immediately processed an returned <i>success</i> - the <i>cbfunc</i> will not be called  |
|          | returned success - the evjune will not be caned  |
| 24       | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the  |
| 25       | request, even though the function entry was provided in the server module - the cbfunc will not  |
| 26       | 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 <i>cbfunc</i> will not be called  |
| 29       | Description  |
| 30       | A local client called <b>PMIx_Abort</b> . Note that the client will be in a blocked state until the host   |
| 31       | server executes the callback function, thus allowing the PMIx server library to release the client.  |
| 32       | The array of <i>procs</i> indicates which processes are to be terminated. A <b>NULL</b> indicates that all   |

# 11.2.5 pmix\_server\_fencenb\_fn\_t

## Summary

33

35

36

At least one client called either  ${\tt PMIx\_Fence}$  or  ${\tt PMIx\_Fence\_nb}$ .

processes in the client's namespace are to be terminated.

| 1  | D141 10   | FOI  | ormat C. ————   |                     |
|----|-----------|------|---|---------------------|
| _  | PMIx v1.0 |      |   | •                   |
| 2  |           | tyr  | <pre>ypedef pmix_status_t (*pmix_server_fencenb_fn_t)(</pre>                            |                     |
| 3  |           |      | const_pmix_proc_t procs[]   | ,                   |
| 4  |           |      | size_t nprocs,  |                     |
| 5  |           |      | <pre>const pmix_info_t info[],</pre>  |                     |
| 6  |           |      | size_t ninfo,   |                     |
| 7  |           |      | char *data, size_t ndata,   |                     |
| 8  |           |      | pmix_modex_cbfunc_t cbfun   | ic,                 |
| 9  |           |      | void *cbdata)   |                     |
| 10 |           | IN   | l procs   |                     |
| 11 |           | 114  | Array of pmix_proc_t structures identifying operation participants(arra                 | ov of handles)      |
| 12 |           | IN   | • • •   | ly of fiditules)    |
| 13 |           |      | Number of elements in the <i>procs</i> array (integer)                                  |                     |
| 14 |           | IN   |   |                     |
| 15 |           |      | Array of info structures (array of handles)   |                     |
| 16 |           | IN   |   |                     |
| 17 |           |      | Number of elements in the <i>info</i> array (integer)                                   |                     |
| 8  |           | IN   |   |                     |
| 19 |           |      | (string)  |                     |
| 20 |           | IN   | · · · · · · · · · · · · · · · · · · ·   |                     |
| 21 |           |      | (integer)   |                     |
| 22 |           | IN   |   |                     |
| 23 |           |      | Callback function <pre>pmix_modex_cbfunc_t</pre> (function reference)                   |                     |
| 24 |           | IN   | cbdata  |                     |
| 25 |           |      | Data to be passed to the callback function (memory reference)                           |                     |
| 26 |           | Reti | eturns one of the following:  |                     |
| 27 |           | • P  | PMIX_SUCCESS, indicating that the request is being processed by the host en             | nvironment - result |
| 28 |           |      | will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the |                     |
| 29 |           |      | prior to returning from the API.  |                     |
| 30 |           | • P  | PMIX_OPERATION_SUCCEEDED, indicating that the request was immediat                      | ely processed and   |
| 31 |           |      | returned <i>success</i> - the <i>cbfunc</i> will not be called                          | • 1                 |
| 32 |           | • P  | PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not                  | ot support the      |
| 33 |           | re   | request, even though the function entry was provided in the server module - the         | e cbfunc will not   |
| 34 |           | b    | be called   |                     |
| 35 |           | • a  | a PMIx error constant indicating either an error in the input or that the request       | was immediately     |
| 36 |           | р    | processed and failed - the <i>cbfunc</i> will not be called                             |                     |

| The | following attributes are required to be supported by all host environments:   |
|-----|---|
| PMI | X_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:  |
| PMI | X_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 protection that process from ever exposing its data.   |
| PMI | X_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 encourage check their host environment for supported values. |
|     | <pre>X_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool) If true, indicates that the requested choice of algorithm is mandatory.</pre>   |
|     |   |

 All local clients in the provided array of *procs* called either **PMIx\_Fence** or **PMIx\_Fence\_nb**. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the **PMIx\_Fence/PMIx\_Fence\_nb** operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix server fencenb fn t** function.

## Advice to PMIx library implementers ————

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

### — Advice to PMIx server hosts —————

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

```
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.

#### 11.2.6 pmix\_server\_dmodex\_req\_fn\_t 2 Summary Used by the PMIx server to request its local host contact the PMIx server on the remote node that 3 hosts the specified proc to obtain and return a direct modex blob for that proc. 4 Format 5 PMIx v1.0 6 typedef pmix status t (\*pmix server dmodex req fn t) ( 7 const pmix\_proc\_t \*proc, const pmix info t info[], 8 9 size t ninfo, 10 pmix modex cbfunc t cbfunc, void \*cbdata) 11 IN 12 proc pmix\_proc\_t structure identifying the process whose data is being requested (handle) 13 IN 14 Array of info structures (array of handles) 15 16 IN ninfo Number of elements in the *info* array (integer) 17 18 IN cbfunc Callback function **pmix modex cbfunc t** (function reference) 19 IN 20 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. • PMIX\_ERR\_NOT\_SUPPORTED, indicating that the host environment does not support the 26 27 request, even though the function entry was provided in the server module - the cbfunc will not 28 be called 29 • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

30

31

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

**A**-----

#### Optional Attributes The following attributes are optional for host environments that support this operation: 1 2 PMIX TIMEOUT "pmix.timeout" (int) 3 Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 4 the target process from ever exposing its data. 5 6 Description 7 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 8 PMIx Put and PMIx Commit. 9 10 The array of *info* structs is used to pass user-requested options to the server. This can include a 11 timeout to preclude an indefinite wait for data that may never become available. The directives are optional unless the mandatory flag has been set - in such cases, the host RM is required to return an 12 error if the directive cannot be met. 13 11.2.7 pmix server publish fn t Summary 15 Publish data per the PMIx API specification. 16 **Format** 17 PMIx v1.0 typedef pmix\_status\_t (\*pmix\_server\_publish\_fn\_t)( 18 19 const pmix\_proc\_t \*proc, 20 const pmix\_info\_t info[], size\_t ninfo, 21 22 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata) 23 IN 24 proc 25 pmix\_proc\_t structure of the process publishing the data (handle) 26 IN 27 Array of info structures (array of handles) 28 IN ninfo

IN

cbfunc

29 30

31

Callback function **pmix\_op\_cbfunc\_t** (function reference)

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

| 1<br>2               | IN cbdata Data to be passed to the callback function (memory reference)   |
|----------------------|---|
| 3                    | Returns one of the following:   |
| 4<br>5<br>6          | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.                              |
| 7<br>8               | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called  |
| 9<br>10<br>11        | • PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called   |
| 12<br>13             | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called   |
|                      | ▼   |
| 14<br>15             | 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 <i>info</i> array:  |
| 16<br>17             | <pre>PMIX_USERID "pmix.euid" (uint32_t)     Effective user id.</pre>  |
| 18<br>19             | PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.  |
| 20                   |   |
| 21                   | Host environments that implement this entry point are required to support the following attributes:   |
| 22<br>23             | PMIX_RANGE "pmix.range" (pmix_data_range_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications.  |
| 24<br>25             | <pre>PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)     Value for calls to PMIx_Publish .</pre>   |
|                      | ▼ Optional Attributes   |
| 26                   | The following attributes are optional for host environments that support this operation:  |
| 27<br>28<br>29<br>30 | 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. |

Publish data per the PMIx\_Publish specification. The callback is to be executed upon completion of the operation. The default data range is left to the host environment, but expected to be PMIX\_RANGE\_SESSION, and the default persistence PMIX\_PERSIST\_SESSION or their equivalent. These values can be specified by including the respective attributed in the *info* array.

The persistence indicates how long the server should retain the data.

#### Advice to PMIx server hosts -

The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range. However, the server must return an error (a) if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of published info by the original publisher - it is left to the discretion of the host environment to allow info-key-based flags to modify this behavior.

The **PMIX\_USERID** and **PMIX\_GRPID** of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

## 5 11.2.8 pmix\_server\_lookup\_fn\_t

#### Summary

Lookup published data.

#### Format

```
PMIx v1.0
```

1

3

4 5

6

7

8

10

11

12

13 14

15

17

18

```
typedef pmix_status_t (*pmix_server_lookup_fn_t)(
const pmix_proc_t *proc,
char **keys,
const pmix_info_t info[],
size_t ninfo,
pmix_lookup_cbfunc_t cbfunc,
void *cbdata)
```

|    | _   |
|----|-----|
| •  |     |
|    |     |
| ١. | - 1 |
|    |     |

| 1              | IN proc  |
|----------------|--|
| 2              | <pre>pmix_proc_t structure of the process seeking the data (handle)</pre>  |
| 3              | IN keys  |
| 4              | (array of strings)  IN info  |
| 5              |  |
| 6<br>7         | Array of info structures (array of handles)  IN ninfo  |
| 8              | Number of elements in the <i>info</i> array (integer)  |
| 9              | IN cbfunc  |
| 0              | Callback function <pre>pmix_lookup_cbfunc_t</pre> (function reference)   |
| 1              | IN cbdata  |
| 2              | Data to be passed to the callback function (memory reference)  |
| 3              | Returns one of the following:  |
| 4<br>5<br>6    | • <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API. |
| 7<br>8         | • PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called  |
| 9<br>20<br>21  | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called   |
| 22<br>23       | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called  |
|                | Required Attributes  |
| 24<br>25       | 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 <i>info</i> array:   |
| 26<br>27       | <pre>PMIX_USERID "pmix.euid" (uint32_t)</pre>  |
| 28<br>29<br>30 | PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.   |
| s1             | Host environments that implement this entry point are required to support the following attributes:  |
| 12             | PMIX_RANGE "pmix.range" (pmix_data_range_t)  |
| 3              | Value for calls to publish/lookup/unpublish or for monitoring event notifications.   |
| 34             | PMIX_WAIT "pmix.wait" (int)  |
|                | Pmin. Hulo (1110)  |

Caller requests that the PMIx server wait until at least the specified number of values are 1 2 found (0 indicates all and is the default). Optional Attributes 3 The following attributes are optional for host environments that support this operation: 4 PMIX\_TIMEOUT "pmix.timeout" (int) 5 Time in seconds before the specified operation should time out (0 indicating infinite) in 6 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent 7 the target process from ever exposing its data. **Description** 8 9 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys 10 identifying the data being requested. 11 The array of *info* structs is used to pass user-requested options to the server. The default data range 12 is left to the host environment, but expected to be PMIX RANGE SESSION. This can include a 13 wait flag to indicate that the server should wait for all data to become available before executing the 14 callback function, or should immediately callback with whatever data is available. In addition, a 15 timeout can be specified on the wait to preclude an indefinite wait for data that may never be published. 16 Advice to PMIx server hosts — 17 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 18 guarantee support for any specific range - i.e., the environment does not need to return an error if 19 20 the data store doesn't support a specified range so long as it is covered by some internally defined

# 11.2.9 pmix\_server\_unpublish\_fn\_t

## Summary

range.

Delete data from the data store.

21

23

| 1              | Format   |
|----------------|--|
| PMIx v1.0      |  |
| 2              | <pre>typedef pmix_status_t (*pmix_server_unpublish_fn_t)(</pre>  |
| 3              | <pre>const pmix_proc_t *proc,</pre>  |
| 4              | char **keys,   |
| 5              | <pre>const pmix_info_t info[],</pre>   |
| 6              | size_t ninfo,  |
| 7              | <pre>pmix_op_cbfunc_t cbfunc,</pre>  |
| 8              | void *cbdata)  |
|                | C  |
| 9              | IN proc  |
| 10             | <pre>pmix_proc_t structure identifying the process making the request (handle)</pre>   |
| 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 <i>info</i> 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<br>23<br>24 | • <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API. |
|                |  |
| 25<br>26       | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called   |
|                |  |
| 27             | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the  |
| 28             | request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not   |
| 29             | be called  |
| 30             | • a PMIx error constant indicating either an error in the input or that the request was immediately  |
| 31             | processed and failed - the <i>cbfunc</i> 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 <i>info</i> array:   |
|                |  |
| 34<br>35       | PMIX_USERID "pmix.euid" (uint32_t)  Effective user id.   |
| 34<br>35       | PMIX_USERID "pmix.euid" (uint32_t)  Effective user id.   |

| <pre>PMIX_GRPID "pmix.egid" (uint32_t)     Effective group id.</pre>  |
|---|
|   |
| 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 preven the target process from ever exposing its data.  |
| Description  Delete data from the data store. The host server will be passed a NULL-terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The default data range is left to the host environment, but expected to be PMIX_RANGE_SESSION. The callback is to be executed upon completion of the delete procedure.       |
| Advice to PMIx server hosts   |
| The PMIX_USERID and PMIX_GRPID of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined |
|   |

# 2 11.2.10 pmix\_server\_spawn\_fn\_t

## Summary

Spawn a set of applications/processes as per the PMIx\_Spawn API.

| DIA      | romat   |
|----------|---|
| PMIx     |   |
| 2        | <pre>typedef pmix_status_t (*pmix_server_spawn_fn_t)(</pre>   |
| 3        | <pre>const pmix_proc_t *proc,</pre>   |
| 4        | <pre>const pmix_info_t job_info[],</pre>  |
| 5        | size_t ninfo,   |
| 6        | <pre>const pmix_app_t apps[],</pre>   |
| 7        | size_t napps,   |
| 8        | <pre>pmix_spawn_cbfunc_t cbfunc,</pre>  |
| 9        | <pre>void *cbdata)</pre>  |
|          | ^ C   |
| 0        | IN proc   |
| 1        | <pre>pmix_proc_t structure of the process making the request (handle)</pre>                               |
| 2        | IN job_info   |
| 3        | Array of info structures (array of handles)   |
| 4        | IN ninfo  |
| 5        | Number of elements in the <i>jobinfo</i> array (integer)  |
| 6        | IN apps   |
| 7        | Array of pmix_app_t structures (array of handles)   |
| 8        | IN napps  |
| 9        | Number of elements in the <i>apps</i> array (integer)   |
| 20       | IN cbfunc   |
| .0<br>?1 |   |
|          | Callback function <pre>pmix_spawn_cbfunc_t</pre> (function reference)  IN cbdata                          |
| 22       |   |
| 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 <i>cbfunc</i> . 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 <i>success</i> - the <i>cbfunc</i> will not be called  |
| 80       | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the                       |
| 31       | request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not    |
| 32       | be called   |
|          |   |
| 33       | • a PMIx error constant indicating either an error in the input or that the request was immediately       |
| 34       | processed and failed - the <i>cbfunc</i> will not be called   |

#### Required Attributes PMIx libraries are required to pass any provided attributes to the host environment for processing. 1 2 In addition, the following attributes are required to be included in the passed *info* array: 3 PMIX USERID "pmix.euid" (uint32 t) Effective user id. 4 PMIX\_GRPID "pmix.egid" (uint32\_t) 5 6 Effective group id. 7 8 Host environments that provide this module entry point are required to pass the PMIX SPAWNED 9 and PMIX\_PARENT\_ID attributes to all PMIx servers launching new child processes so those 10 values can be returned to clients upon connection to the PMIx server. In addition, they are required 11 to support the following attributes when present in either the job info or the info array of an element of the apps array: 12 PMIX WDIR "pmix.wdir" (char\*) 13 Working directory for spawned processes. 14 15 PMIX SET SESSION CWD "pmix.ssncwd" (bool) Set the application's current working directory to the session working directory assigned by 16 the RM - when accessed using PMIx\_Get , use the PMIX\_RANK\_WILDCARD value for 17 the rank to discover the session working directory assigned to the provided namespace 18 19 PMIX PREFIX "pmix.prefix" (char\*) 20 Prefix to use for starting spawned processes. 21 PMIX HOST "pmix.host" (char\*) 22 Comma-delimited list of hosts to use for spawned processes. 23 PMIX HOSTFILE "pmix.hostfile" (char\*) 24 Hostfile to use for spawned processes. \_\_\_\_\_ Optional Attributes 25 The following attributes are optional for host environments that support this operation: 26 PMIX\_ADD\_HOSTFILE "pmix.addhostfile" (char\*) Hostfile listing hosts to add to existing allocation. 27 28 PMIX ADD HOST "pmix.addhost" (char\*) Comma-delimited list of hosts to add to the allocation. 29 30 PMIX PRELOAD BIN "pmix.preloadbin" (bool) 31 Preload binaries onto nodes. 32 PMIX PRELOAD FILES "pmix.preloadfiles" (char\*)

| 1                    | Comma-delimited list of files to pre-position on nodes.   |
|----------------------|---|
| 2                    | <pre>PMIX_PERSONALITY "pmix.pers" (char*) Name of personality to use.</pre>   |
| 4<br>5<br>6<br>7     | <pre>PMIX_MAPPER "pmix.mapper" (char*)     Mapping mechanism to use for placing spawned processes - when accessed using     PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the mapping     mechanism used for the provided namespace.</pre> |
| 8<br>9               | PMIX_DISPLAY_MAP "pmix.dispmap" (bool)  Display process mapping upon spawn.   |
| 10<br>11             | <pre>PMIX_PPR "pmix.ppr" (char*) Number of processes to spawn on each identified resource.</pre>  |
| 12<br>13<br>14<br>15 | <pre>PMIX_MAPBY "pmix.mapby" (char*)     Process mapping policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the     provided namespace</pre>                                       |
| 16<br>17<br>18<br>19 | <pre>PMIX_RANKBY "pmix.rankby" (char*)     Process ranking policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the     provided namespace</pre>                                  |
| 20<br>21<br>22<br>23 | <pre>PMIX_BINDTO "pmix.bindto" (char*)     Process binding policy - when accessed using PMIx_Get , use the     PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the     provided namespace</pre>                                     |
| 24<br>25             | PMIX_NON_PMI "pmix.nonpmi" (bool)  Spawned processes will not call PMIx_Init.   |
| 26<br>27             | PMIX_STDIN_TGT "pmix.stdin" (uint32_t) Spawned process rank that is to receive stdin.   |
| 28<br>29             | <pre>PMIX_FWD_STDIN "pmix.fwd.stdin" (bool) Forward this process's stdin to the designated process.</pre>   |
| 30<br>31             | PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)  Forward stdout from spawned processes to this process.  |
| 32<br>33             | <pre>PMIX_FWD_STDERR "pmix.fwd.stderr" (bool) Forward stderr from spawned processes to this process.</pre>  |
| 34<br>35             | PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool) Spawned application consists of debugger daemons.  |
| 36                   | PMIX TAG OUTPUT "pmix.tagout" (bool)  |

| 1                    | Tag application output with the identity of the source process.   |
|----------------------|---|
| 2 3                  | PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool) Timestamp output from applications.   |
| 4<br>5               | PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)  Merge stdout and stderr streams from application processes.   |
| 6<br>7               | <pre>PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*) Output application output to the specified file.</pre>  |
| 8<br>9               | PMIX_INDEX_ARGV "pmix.indxargv" (bool)  Mark the argv with the rank of the process.   |
| 10<br>11<br>12<br>13 | PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)  Number of cpus to assign to each rank - when accessed using PMIx_Get , use the  PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the provided namespace                                  |
| 14<br>15             | PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)  Do not place processes on the head node.   |
| 16<br>17             | PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)  Do not oversubscribe the cpus.  |
| 18<br>19             | PMIX_REPORT_BINDINGS "pmix.repbind" (bool) Report bindings of the individual processes.   |
| 20<br>21<br>22<br>23 | <pre>PMIX_CPU_LIST "pmix.cpulist" (char*) List of cpus to use for this job - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided namespace</pre>  |
| 24<br>25             | PMIX_JOB_RECOVERABLE "pmix.recover" (bool) Application supports recoverable operations.   |
| 26<br>27             | PMIX_JOB_CONTINUOUS "pmix.continuous" (bool) Application is continuous, all failed processes should be immediately restarted.   |
| 28<br>29<br>30<br>31 | PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)  Maximum number of times to restart a job - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided namespace  |
| 32<br>33<br>34<br>35 | 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. |

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
```

```
14
15
16
17
```

PMIx v1.0

1

2

3

5

6

7

8

9

11 12

13

18

19

20

21

22

23

24

25

26

27

28 29

30 31

32

33

34

35 36

C

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.

| 1<br>2                | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called  |
|-----------------------|---|
| 3<br>4<br>5           | • PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called   |
| 6<br>7                | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called   |
|                       | ▼   |
| 8                     | PMIx libraries are required to pass any provided attributes to the host environment for processing.   |
|                       | ▼Optional Attributes  |
| 9                     | The following attributes are optional for host environments that support this operation:  |
| 0<br>1<br>2<br>3      | 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.   |
| 4<br>5<br>6<br>7<br>8 | 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. |
| 9<br>0                | <pre>PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)     If true, indicates that the requested choice of algorithm is mandatory.</pre>  |

2

3

4

5

6

7

8

9

10

13

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server's **pmix\_server\_connect\_fn\_t** function, and the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

## Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

#### Advice to PMIx server hosts –

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

# 11 11.2.12 pmix\_server\_disconnect\_fn\_t

### 12 Summary

Disconnect a previously connected set of processes.

| 1        | Format   |
|----------|--|
| PMIx v1. |  |
| 2        | <pre>typedef pmix_status_t (*pmix_server_disconnect_fn_t)(</pre>   |
| 3        | <pre>const pmix_proc_t procs[],</pre>  |
| 4        | size_t nprocs,   |
| 5        | <pre>const pmix_info_t info[],</pre>   |
| 6        | size_t ninfo,  |
| 7        | <pre>pmix_op_cbfunc_t cbfunc,</pre>  |
| 8        | void *cbdata)  |
|          |  |
| 9        | IN procs   |
| 10       | Array of pmix_proc_t structures identifying participants (array of handles)                                      |
| 11       | IN nprocs  |
| 12       | Number of elements in the <i>procs</i> array (integer)   |
| 13       | IN info  |
| 14       | Array of info structures (array of handles)  |
| 15       | IN ninfo   |
| 16       | Number of elements in the <i>info</i> 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 <i>cbfunc</i> . 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 <i>success</i> - the <i>cbfunc</i> will not be called   |
|          |  |
| 27       | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the                              |
| 28<br>29 | request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called |
| 29       | be called  |
| 30       | • a PMIx error constant indicating either an error in the input or that the request was immediately              |
| 31       | processed and failed - the <i>cbfunc</i> will not be called  |
|          | ▼ Required Attributes  |
| 32       | PMIx libraries are required to pass any provided attributes to the host environment for processing.              |
| -        | A state of the required to pass any provided uniform for inout on vironment for processing.                      |

|                  | → Optional Attributes  |
|------------------|--|
| 1                | The following attributes are optional for host environments that support this operation:   |
| 2<br>3<br>4<br>5 | 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.  |
| 6<br>7<br>8<br>9 | <b>Description</b> Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server's has called the <pre>pmix_server_disconnect_fn_t</pre> function, and the host environment has completed any required supporting operations. |
|                  | Advice to PMIx library implementers ————————————————————————————————————   |
| 11<br>12         | 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  |
| 13<br>14<br>15   | The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.                      |
| 16               | A <b>PMIX_ERR_INVALID_OPERATION</b> error must be returned if the specified set of <i>procs</i> was not previously <i>connected</i> via a call to the <b>pmix_server_connect_fn_t</b> function.  |

**Summary** 

Register to receive notifications for the specified events.

| 1              |           | Format   |
|----------------|-----------|--|
|                | PMIx v1.0 |  |
| 2              |           | <pre>typedef pmix_status_t (*pmix_server_register_events_fn_t)(</pre>  |
| 3              |           | <pre>pmix_status_t *codes,</pre>   |
| 4              |           | size_t ncodes,   |
| 5              |           | <pre>const pmix_info_t info[],</pre>   |
| 6              |           | size_t ninfo,  |
| 7              |           | <pre>pmix_op_cbfunc_t cbfunc,</pre>  |
| 8              |           | <pre>void *cbdata)</pre>   |
|                |           | C  |
| 9              |           | IN codes   |
| 10             |           | Array of pmix_status_t values (array of handles)   |
| 11             |           | IN ncodes  |
| 12             |           | Number of elements in the <i>codes</i> array (integer)   |
| 13             |           | IN info  |
| 14             |           | Array of info structures (array of handles)  |
| 15             |           | IN ninfo   |
| 16             |           | Number of elements in the <i>info</i> array (integer)  |
| 17             |           | IN cbfunc  |
| 18             |           | Callback function <b>pmix_op_cbfunc_t</b> (function reference)   |
| 19             |           | IN cbdata  |
| 20             |           | Data to be passed to the callback function (memory reference)  |
| 21             |           | Returns one of the following:  |
| 22<br>23<br>24 |           | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API. |
|                |           |  |
| 25<br>26       |           | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and<br/>returned success - the cbfunc will not be called</li> </ul>   |
| 27             |           | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the  |
| 28             |           | request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not   |
| 29             |           | be called  |
| 30             |           | • a PMIx error constant indicating either an error in the input or that the request was immediately  |
| 31             |           | processed and failed - the <i>cbfunc</i> 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 <i>info</i> array:   |
| 34<br>35       |           | <pre>PMIX_USERID "pmix.euid" (uint32_t)     Effective user id.</pre>   |

| 1 2                  | PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.  |
|----------------------|---|
| 3<br>4<br>5          | Description Register to receive notifications for the specified status codes. The <i>info</i> array included in this API is reserved for possible future directives to further steer notification.  Advice to PMIx library implementers   |
| 6<br>7               | The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:  |
| 8<br>9<br>10         | <ul> <li>the client has requested notification of an environmental code (i.e., a PMIx code in the range<br/>beyond PMIX_ERR_SYS_OTHER) or a code that lies outside the defined PMIx range of<br/>constants; and</li> </ul>  |
| 11<br>12             | • 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   |
| 13<br>14<br>15<br>16 | 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). |

# 17 11.2.14 pmix\_server\_deregister\_events\_fn\_t

# Summary

18

19

Deregister to receive notifications for the specified events.

| 1                                   |          | Format   |
|-------------------------------------|----------|--|
| 2<br>3<br>4<br>5<br>6               | MIx v1.0 | <pre>typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(</pre>  |
| 7<br>8<br>9<br>10<br>11<br>12<br>13 |          | IN codes Array of pmix_status_t values (array of handles)  IN ncodes Number of elements in the codes array (integer)  IN cbfunc Callback function pmix_op_cbfunc_t (function reference)  IN cbdata Data to be passed to the callback function (memory reference)   |
| 15<br>16<br>17<br>18                |          | Returns one of the following:  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.  |
| 19<br>20<br>21<br>22<br>23          |          | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called</li> <li>PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called</li> </ul> |
| 24<br>25                            |          | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called  |
| 26<br>27<br>28                      |          | Description Deregister to receive notifications for the specified events to which the PMIx server has previously registered.  Advice to PMIx library implementers  |
| 29<br>30<br>31<br>32<br>33          |          | The PMIx server library must track all client registrations. This module function shall only be called when:  • the library is deregistering environmental codes (i.e., a PMIx codes in the range between PMIX_ERR_SYS_BASE and PMIX_ERR_SYS_OTHER, inclusive) or codes that lies outside the defined PMIx range of constants; and   |
|                                     |          |  |

1 • no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it 2 remain. 3 11.2.15 pmix\_server\_notify\_event\_fn\_t 5 Summary Notify the specified processes of an event. 6 **Format** PMIx v2.0 8 typedef pmix\_status\_t (\*pmix\_server\_notify\_event\_fn\_t)(pmix\_status\_t code, const pmix\_proc\_t \*source, 9 10 pmix\_data\_range\_t range, 11 pmix\_info\_t info[], 12 size\_t ninfo, pmix\_op\_cbfunc\_t cbfunc, 13 void \*cbdata); 14 IN 15 code The **pmix\_status\_t** event code being referenced structure (handle) 16 IN 17 18 pmix proc t of process that generated the event (handle) 19 IN range 20 pmix data range t range over which the event is to be distributed (handle) info 21 IN Optional array of pmix\_info\_t structures containing additional information on the event 22 (array of handles) 23 IN ninfo 24 Number of elements in the *info* array (integer) 25 26 IN Callback function pmix\_op\_cbfunc\_t (function reference) 27 cbdata 28 IN Data to be passed to the callback function (memory reference) 29 30 Returns one of the following: • PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result 31 32 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function 33 prior to returning from the API.

34

35

• PMIX OPERATION SUCCEEDED, indicating that the request was immediately processed and

returned success - the cbfunc will not be called

• PMIX\_ERR\_NOT\_SUPPORTED, indicating that the host environment does not support the 1 2 request, even though the function entry was provided in the server module - the cbfunc will not be called 3 4 • 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 5 -----Required Attributes PMIx libraries are required to pass any provided attributes to the host environment for processing. 6 Host environments that provide this module entry point are required to support the following 8 attributes: 9 10 PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t) Value for calls to publish/lookup/unpublish or for monitoring event notifications. 11 12 Description 13 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. 14 The process generating the event is provided in the *source* parameter, and any further descriptive 15 16 information is included in the info array. Advice to PMIx server hosts ——— 17 The callback function is to be executed once the host environment no longer requires that the PMIx 18 server library maintain the provided data structures. It does not necessarily indicate that the event 19 has been delivered to any process, nor that the event has been distributed for delivery

## 20 11.2.16 pmix\_server\_listener\_fn\_t

## Summary

Register a socket the host server can monitor for connection requests.

21

```
Format
1
   PMIx v1.0
2
               typedef pmix_status_t (*pmix_server_listener_fn_t)(
 3
                                                     int listening sd,
                                                     pmix connection cbfunc t cbfunc,
 4
                                                     void *cbdata)
5
               IN
6
                    incoming sd
 7
                    (integer)
8
               IN
                    cbfunc
9
                    Callback function pmix connection cbfunc t (function reference)
               IN
10
                    cbdata
                    (memory reference)
11
12
               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.
13
               Description
14
15
               Register a socket the host environment can monitor for connection requests, harvest them, and then
16
               call the PMIx server library's internal callback function for further processing. A listener thread is
17
               essential to efficiently harvesting connection requests from large numbers of local clients such as
               occur when running on large SMPs. The host server listener is required to call accept on the
18
19
               incoming connection request, and then pass the resulting socket to the provided cbfunc. A NULL
20
               for this function will cause the internal PMIx server to spawn its own listener thread.
    11.2.17
                 pmix_server_query_fn_t
22
               Summary
               Query information from the resource manager.
23
               Format
24
   PMIx v2.0
25
               typedef pmix_status_t (*pmix_server_query_fn_t)(
26
                                                     pmix_proc_t *proct,
27
                                                     pmix query t *queries, size t nqueries,
28
                                                     pmix info cbfunc t cbfunc,
                                                     void *cbdata)
29
               IN
30
                    proct
31
                    pmix_proc_t structure of the requesting process (handle)
               IN
32
                    queries
```

Array of pmix query t structures (array of handles)

33

| 1<br>2<br>3<br>4<br>5<br>6 | <ul> <li>IN nqueries         Number of elements in the queries array (integer)     </li> <li>IN cbfunc         Callback function pmix_info_cbfunc_t (function reference)     </li> <li>IN cbdata         Data to be passed to the callback function (memory reference)     </li> </ul> |  |
|----------------------------|--|--|
| 7                          | Returns one of the following:  |  |
| 8<br>9<br>10               | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API.   |  |
| 11<br>12                   | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and<br/>returned success - the cbfunc will not be called</li> </ul>   |  |
| 13<br>14<br>15             | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called   |  |
| 16<br>17                   | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called  |  |
|                            | ▼  |  |
| 18<br>19                   | 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 <i>info</i> array:   |  |
| 20<br>21                   | PMIX_USERID "pmix.euid" (uint32_t) Effective user id.  |  |
| 22<br>23                   | PMIX_GRPID "pmix.egid" (uint32_t)  Effective group id.   |  |
|                            | ▼ Optional Attributes  |  |
| 24                         | The following attributes are optional for host environments that support this operation:   |  |
| 25<br>26                   | PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*) Request a comma-delimited list of active namespaces.   |  |
| 27<br>28                   | <pre>PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t) Status of a specified, currently executing job.</pre>  |  |
| 29<br>30                   | <pre>PMIX_QUERY_QUEUE_LIST "pmix.qry.qlst" (char*) Request a comma-delimited list of scheduler queues.</pre>   |  |
| 31<br>32                   | <pre>PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD) Status of a specified scheduler queue.</pre>   |  |

| 1<br>2<br>3      | PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*) Input namespace of the job whose information is being requested returns ( pmix_data_array_t) an array of pmix_proc_info_t.  |
|------------------|---|
| 4<br>5<br>6<br>7 | 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. |
| 8<br>9           | PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool) Return a comma-delimited list of supported spawn attributes.   |
| 0<br>1           | PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)  Return a comma-delimited list of supported debug attributes.  |
| 2<br>3           | PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)  Return information on memory usage for the processes indicated in the qualifiers.  |
| 4<br>5           | PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)  Constrain the query to local information only.   |
| 6<br>7           | PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)  Report only average values for sampled information.  |
| 8<br>9           | PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool) Report minimum and maximum values.  |
| 20<br>21         | <pre>PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*) String identifier of the allocation whose status is being requested.</pre>  |
| 22<br>23<br>24   | <pre>PMIX_TIME_REMAINING "pmix.time.remaining" (char*)</pre>  |
|                  | <b>AA</b>   |
| 25               | Description   |
| 26               | Query information from the host environment. The query will include the namespace/rank of the   |
| 27               | process that is requesting the info, an array of <b>pmix_query_t</b> describing the request, and a  |
| 28               | callback function/data for the return.  Advice to PMIx library implementers   |
| 29               | The PMIx server library should not block in this function as the host environment may, depending  |
| 30               | upon the information being requested, require significant time to respond.  |

#### Summarv Register that a tool has connected to the server. 3 Format PMIx v2.05 typedef void (\*pmix\_server\_tool\_connection\_fn\_t)( 6 pmix info t info[], size t ninfo, 7 pmix tool connection obfunc t obfunc, void \*cbdata) 8 IN 9 info Array of pmix info t structures (array of handles) 10 11 IN 12 Number of elements in the *info* array (integer) cbfunc 13 Callback function pmix tool connection cbfunc t (function reference) 14 IN cbdata 15 Data to be passed to the callback function (memory reference) 16 Required Attributes 17 PMIx libraries are required to pass the following attributes in the *info* array: PMIX\_USERID "pmix.euid" (uint32\_t) 18 Effective user id. 19 20 PMIX\_GRPID "pmix.egid" (uint32\_t) 21 Effective group id. Optional Attributes \_\_\_\_\_ 22 The following attributes are optional for host environments that support this operation: 23 PMIX FWD STDOUT "pmix.fwd.stdout" (bool) Forward **stdout** from spawned processes to this process. 24 25 PMIX FWD STDERR "pmix.fwd.stderr" (bool) Forward **stderr** from spawned processes to this process. 26 27 PMIX FWD STDIN "pmix.fwd.stdin" (bool) Forward this process's **stdin** to the designated process. 28

11.2.18 pmix\_server\_tool\_connection\_fn\_t

# Description Register that a too namespace/rank ic qualifiers for the c

Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The <code>pmix\_info\_t</code> array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

#### Advice to PMIx server hosts -

The host environment is solely responsible for authenticating and authorizing the connection, and for authorizing all subsequent tool requests. The host must not execute the callback function prior to returning from the API.

# 11.2.19 pmix\_server\_log\_fn\_t

#### Summary

Log data on behalf of a client.

#### **Format**

PMIx v2.0

5

6

7

8

10 11

12

18 19

20 21

22

23 24

25

26

27

28

29

30

31

. . . . . .

```
typedef void (*pmix_server_log_fn_t)(
const pmix_proc_t *client,
const pmix_info_t data[], size_t ndata,
const pmix_info_t directives[], size_t ndirs,
pmix_op_cbfunc_t cbfunc, void *cbdata)
```

IN client

pmix\_proc\_t structure (handle)

IN data

Array of info structures (array of handles)

IN ndata

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

IN directives

Array of info structures (array of handles)

IN ndirs

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

IN cbfunc

Callback function **pmix** op **cbfunc t** (function reference)

IN cbdata

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

#### Required Attributes ------PMIx libraries are required to pass any provided attributes to the host environment for processing. 1 In addition, the following attributes are required to be included in the passed *info* array: 2 PMIX USERID "pmix.euid" (uint32\_t) 3 Effective user id. 4 PMIX\_GRPID "pmix.egid" (uint32\_t) 5 6 Effective group id. 7 Host environments that provide this module entry point are required to support the following 8 9 attributes: PMIX\_LOG\_STDERR "pmix.log.stderr" (char\*) 10 11 Log string to **stderr**. PMIX\_LOG\_STDOUT "pmix.log.stdout" (char\*) 12 13 Log string to **stdout**. 14 PMIX LOG SYSLOG "pmix.log.syslog" (char\*) Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, 15 otherwise to local syslog 16 **▲**-----**-**Optional Attributes -----17 The following attributes are optional for host environments that support this operation: PMIX\_LOG\_MSG "pmix.log.msg" (pmix\_byte\_object\_t) 18 19 Message blob to be sent somewhere. PMIX LOG EMAIL "pmix.log.email" (pmix data array t) 20 Log via email based on **pmix\_info\_t** containing directives. 21 22 PMIX LOG EMAIL ADDR "pmix.log.emaddr" (char\*) 23 Comma-delimited list of email addresses that are to receive the message. PMIX LOG EMAIL\_SUBJECT "pmix.log.emsub" (char\*) 24 Subject line for email. 25 26 PMIX\_LOG\_EMAIL\_MSG "pmix.log.emmsg" (char\*) 27 Message to be included in email.

**A** ------

# 1 **Description**2 Log data on bel

Log data on behalf of a client. This function is not intended for output of computational results, but rather for reporting status and error messages. The host must not execute the callback function prior to returning from the API.

# 11.2.20 pmix\_server\_alloc\_fn\_t

#### Summary

Request allocation operations on behalf of a client.

#### **Format**

PMIx v2.0

3

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

36

IN client

pmix\_proc\_t structure of process making request (handle)

IN directive

Specific action being requested ( pmix\_alloc\_directive\_t )

IN data

Array of info structures (array of handles)

IN ndata

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

IN cbfunc

Callback function **pmix info cbfunc t** (function reference)

IN cbdata

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

#### Returns one of the following:

- PMIX\_SUCCESS, indicating that the request is being processed by the host environment result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and returned *success* the *cbfunc* will not be called
- PMIX\_ERR\_NOT\_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed the *cbfunc* will not be called

```
Required Attributes -----
              PMIx libraries are required to pass any provided attributes to the host environment for processing.
 1
 2
              In addition, the following attributes are required to be included in the passed info array:
 3
              PMIX USERID "pmix.euid" (uint32 t)
                   Effective user id.
 4
              PMIX_GRPID "pmix.egid" (uint32_t)
 5
6
                   Effective group id.
7
              Host environments that provide this module entry point are required to support the following
8
9
              attributes:
              PMIX_ALLOC_ID "pmix.alloc.id" (char*)
10
11
                   A string identifier (provided by the host environment) for the resulting allocation which can
                   later be used to reference the allocated resources in, for example, a call to PMIx Spawn.
12
              PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
13
                   The number of nodes.
14
15
              PMIX ALLOC NUM CPUS "pmix.alloc.ncpus" (uint64 t)
16
                   Number of cpus.
17
              PMIX ALLOC TIME "pmix.alloc.time" (uint32 t)
                   Time in seconds.
18
              ▲------
                                           Optional Attributes
19
              The following attributes are optional for host environments that support this operation:
20
              PMIX ALLOC NODE LIST "pmix.alloc.nlist" (char*)
21
                   Regular expression of the specific nodes.
22
              PMIX ALLOC NUM CPU LIST "pmix.alloc.ncpulist" (char*)
                   Regular expression of the number of cpus for each node.
23
24
              PMIX ALLOC CPU LIST "pmix.alloc.cpulist" (char*)
25
                   Regular expression of the specific cpus indicating the cpus involved.
26
              PMIX ALLOC MEM SIZE "pmix.alloc.msize" (float)
                   Number of Megabytes.
27
28
              PMIX ALLOC FABRIC "pmix.alloc.net" (array)
                   Array of pmix_info_t describing requested fabric resources. This must include at least:
29
                   PMIX ALLOC FABRIC ID, PMIX ALLOC FABRIC TYPE, and
30
                   PMIX ALLOC FABRIC ENDPTS, plus whatever other descriptors are desired.
31
32
              PMIX ALLOC FABRIC ID "pmix.alloc.netid" (char*)
```

The key to be used when accessing this requested fabric 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 fabric 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 FABRIC TYPE** - the type of resources provided; PMIX ALLOC FABRIC PLANE - if applicable, what plane the resources were assigned from; PMIX\_ALLOC\_FABRIC\_QOS - the assigned QoS; PMIX\_ALLOC\_BANDWIDTH the allocated bandwidth; PMIX\_ALLOC\_FABRIC\_SEC\_KEY - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX INFO REOD** was not set in the request. PMIX ALLOC BANDWIDTH "pmix.alloc.bw" (float) Mbits/sec. PMIX\_ALLOC\_FABRIC\_QOS "pmix.alloc.netqos" (char\*)

### Description

Ouality of service level.

1

2

3

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

35

36

Request new allocation or modifications to an existing allocation on behalf of a client. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute for an existing allocation. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities.
- Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to the scheduler with a promise to return them upon subsequent request.

The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of **pmix\_info\_t** structures.

# 11.2.21 pmix\_server\_job\_control\_fn\_t

#### Summary

Execute a job control action on behalf of a client.

| 1 PMIx v2.0           | Format C   |  |  |
|-----------------------|--|--|--|
| 2<br>3<br>4<br>5<br>6 | <pre>typedef pmix_status_t (*pmix_server_job_control_fn_t)(</pre>  |  |  |
| 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 <i>targets</i> array (integer)   |  |  |
| 13                    | IN directives  |  |  |
| 14                    | Array of info structures (array of handles)  |  |  |
| 15                    | IN ndirs   |  |  |
| 16<br>17              | Number of elements in the <i>info</i> array (integer)  |  |  |
| 18                    | IN cbfunc  Callback function pmix_op_cbfunc_t (function reference)   |  |  |
| 19                    | IN cbdata (function reference)   |  |  |
| 20                    | Data to be passed to the callback function (memory reference)  |  |  |
| 21                    | Returns one of the following:  |  |  |
| 22<br>23<br>24        | • <b>PMIX_SUCCESS</b> , indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the host must not invoke the callback function prior to returning from the API. |  |  |
| 25<br>26              | <ul> <li>PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and<br/>returned success - the cbfunc will not be called</li> </ul>   |  |  |
| 27<br>28<br>29        | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called   |  |  |
| 30<br>31              | <ul> <li>a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called</li> </ul>  |  |  |
|                       | ▼  |  |  |
| 32<br>33              | PMIx libraries are required to pass any attributes provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the passed <i>info</i> array:                               |  |  |
| 34<br>35              | PMIX_USERID "pmix.euid" (uint32_t)  Effective user id.   |  |  |

```
PMIX_GRPID "pmix.egid" (uint32_t)
 1
 2
                     Effective group id.
 3
               Host environments that provide this module entry point are required to support the following
 4
 5
               attributes:
6
               PMIX JOB CTRL ID "pmix.jctrl.id" (char*)
 7
                     Provide a string identifier for this request. The user can provide an identifier for the
8
                     requested operation, thus allowing them to later request status of the operation or to
9
                     terminate it. The host, therefore, shall track it with the request for future reference.
10
               PMIX JOB CTRL PAUSE "pmix.jctrl.pause" (bool)
                     Pause the specified processes.
11
12
               PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
                     Resume ("un-pause") the specified processes.
13
               PMIX JOB CTRL KILL "pmix.jctrl.kill" (bool)
14
                     Forcibly terminate the specified processes and cleanup.
15
16
               PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
17
                     Send given signal to specified processes.
18
               PMIX JOB CTRL TERMINATE "pmix.jctrl.term" (bool)
19
                     Politely terminate the specified processes.
                                               Optional Attributes
20
               The following attributes are optional for host environments that support this operation:
               PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
21
22
                     Cancel the specified request - the provided request ID must match the
                     PMIX JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
23
24
                     NULL implies cancel all requests from this requestor.
               PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
25
26
                     Restart the specified processes using the given checkpoint ID.
27
               PMIX JOB CTRL CHECKPOINT "pmix.jctrl.ckpt" (char*)
                     Checkpoint the specified processes and assign the given ID to it.
28
29
               PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
                     Use event notification to trigger a process checkpoint.
30
               PMIX JOB_CTRL_CHECKPOINT_SIGNAL
31
                                                         "pmix.jctrl.ckptsig" (int)
                     Use the given signal to trigger a process checkpoint.
32
33
               PMIX JOB CTRL CHECKPOINT TIMEOUT "pmix.jctrl.ckptsig" (int)
```

```
Time in seconds to wait for a checkpoint to complete.
 1
               PMIX JOB CTRL CHECKPOINT METHOD
 2
               "pmix.jctrl.ckmethod" (pmix_data_array_t)
 3
                    Array of pmix_info_t declaring each method and value supported by this application.
 4
 5
               PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
 6
                    Regular expression identifying nodes that are to be provisioned.
 7
               PMIX JOB CTRL PROVISION IMAGE "pmix.jctrl.pvnimg" (char*)
 8
                    Name of the image that is to be provisioned.
               PMIX JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
 9
10
                    Indicate that the job can be pre-empted.
11
               Description
12
               Execute a job control action on behalf of a client. The targets array identifies the processes to
13
               which the requested job control action is to be applied. A NULL value can be used to indicate all
               processes in the caller's namespace. The use of PMIX RANK WILDCARD can also be used to
14
               indicate that all processes in the given namespace are to be included.
15
16
               The directives are provided as pmix_info_t structures in the directives array. The callback
17
               function provides a status to indicate whether or not the request was granted, and to provide some
18
               information as to the reason for any denial in the pmix_info_cbfunc_t array of
19
               pmix_info_t structures.
    11.2.22 pmix server monitor fn t
20
21
               Summarv
22
               Request that a client be monitored for activity.
               Format
23
   PMIx v2.0
24
               typedef pmix_status_t (*pmix_server_monitor_fn_t)(
25
                                                    const pmix_proc_t *requestor,
                                                    const pmix_info_t *monitor, pmix_status_t error
26
27
                                                    const pmix_info_t directives[], size_t ndirs,
28
                                                    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

| 1  | IN requestor  |
|----|---|
| 2  | <pre>pmix_proc_t structure of requesting process (handle)</pre>   |
| 3  | IN monitor  |
| 4  | <pre>pmix_info_t identifying the type of monitor being requested (handle)</pre>                           |
| 5  | IN error  |
| 6  | Status code to use in generating event if alarm triggers (integer)  |
| 7  | IN directives   |
| 8  | Array of info structures (array of handles)   |
| 9  | IN ndirs  |
| 10 | Number of elements in the <i>info</i> array (integer)   |
| 11 | IN cbfunc   |
| 12 | Callback function <b>pmix_op_cbfunc_t</b> (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 <i>cbfunc</i> . 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 <i>success</i> - the <i>cbfunc</i> will not be called  |
|    | ř   |
| 21 | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the                       |
| 22 | request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not    |
| 23 | be called   |
| 24 | • a PMIx error constant indicating either an error in the input or that the request was immediately       |
| 25 | processed and failed - the <i>cbfunc</i> will not be called   |
| 26 | This entry point is only called for monitoring requests that are not directly supported by the PMIx       |
|    | This chary point is only cancallor monitoring requests that are not uncerty supported by the I with       |

## Required Attributes

If supported by the PMIx server library, then the library must not pass any supported attributes to the host environment. Any attributes provided by the client that are not directly supported by the server library must be passed to the host environment if it provides this module entry. In addition, the following attributes are required to be included in the passed *info* array:

server library itself.

| l l      |   |  |
|----------|---|--|
| 2        | Host environments are not required to support any specific monitoring attributes.   |  |
|          | ▼ Optional Attributes   |  |
| 3        | The following attributes may be implemented by a host environment.  |  |
| 4<br>5   | <pre>PMIX_MONITOR_ID "pmix.monitor.id" (char*) Provide a string identifier for this request.</pre>  |  |
| 6<br>7   | <pre>PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*) Identifier to be canceled (NULL means cancel all monitoring for this process).</pre>                 |  |
| 8<br>9   | <pre>PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool) The application desires to control the response to a monitoring event.</pre>                    |  |
| 10<br>11 | <pre>PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void) Register to have the PMIx server monitor the requestor for heartbeats.</pre>                        |  |
| 12<br>13 | PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t) Time in seconds before declaring heartbeat missed.  |  |
| 14<br>15 | PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)  Number of heartbeats that can be missed before generating the event.                          |  |
| 16<br>17 | <pre>PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*) Register to monitor file for signs of life.</pre>  |  |
| 18<br>19 | PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)  Monitor size of given file is growing to determine if the application is running.                       |  |
| 20<br>21 | <pre>PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*) Monitor time since last access of given file to determine if the application is running.</pre> |  |
| 22<br>23 | <pre>PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)  Monitor time since last modified of given file to determine if the application is running.</pre> |  |
| 24<br>25 | PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t) Time in seconds between checking the file.   |  |
| 26<br>27 | PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)  Number of file checks that can be missed before generating the event.                              |  |
| 28<br>29 | <b>Description</b> Request that a client be monitored for activity.   |  |

Request that a client be monitored for activity.

#### 11.2.23 pmix\_server\_get\_cred\_fn\_t Summary Request a credential from the host environment 3 Format PMIx v3.05 typedef pmix status t (\*pmix server get cred fn t)( 6 const pmix proc t \*proc, 7 const pmix info t directives[], 8 size t ndirs, 9 pmix credential cbfunc t cbfunc, 10 void \*cbdata); C 11 IN proc 12 pmix proc t structure of requesting process (handle) IN 13 directives Array of info structures (array of handles) 14 IN ndirs 15 16 Number of elements in the *info* array (integer) 17 IN cbfunc 18 Callback function to return the credential (pmix\_credential\_cbfunc\_t function reference) 19 IN 20 cbdata 21 Data to be passed to the callback function (memory reference) 22 Returns PMIX\_SUCCESS, PMIX\_ERR\_NOT\_SUPPORTED indicating that the host environment 23 does not support the request (even though the function entry was provided in the server module), or a negative value corresponding to a PMIx error constant. In the event the function returns an error, 24 25 the cbfunc will not be called. Required Attributes **\_**\_\_\_\_\_ 26 If the PMIx library does not itself provide the requested credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include 27 the following attributes in the passed *info* array: 28 PMIX\_USERID "pmix.euid" (uint32\_t) 29 Effective user id.

PMIX\_GRPID "pmix.egid" (uint32\_t)

Effective group id.

30 31

| → Optional Attributes  |  |
|--|--|
| The following attributes are optional for host environments that support this operation:   |  |
| <pre>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.</pre>  |  |
| 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

16

17

19

20

Request a credential from the host environment

# 18 11.2.24 pmix\_server\_validate\_cred\_fn\_t

## Summary

Request validation of a credential

| 1  | D144 2.0 | Format   |  |  |
|--|----------|--|--|--|
| PMIx v3.0 typedef pmix_status_t (*pmix_server_validate_cred_fn_t)( |          |  |  |  |
| 3  |          | const pmix_proc_t *proc,   |  |  |
| 4  |          | <pre>const pmix_byte_object_t *cred,</pre>   |  |  |
| 5  |          | <pre>const pmix_info_t directives[],</pre>   |  |  |
| 6  |          | size t ndirs,  |  |  |
| 7  |          | pmix_validation_cbfunc_t cbfunc,   |  |  |
| 8  |          | void *cbdata);   |  |  |
|  |          | C  |  |  |
| 9  |          | IN proc  |  |  |
| 0  |          | <pre>pmix_proc_t structure of requesting process (handle)</pre>  |  |  |
| 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 <i>info</i> array (integer)  IN cbfunc   |  |  |
| 17<br>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<br>24   |          | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i>       |  |  |
| 25<br>26   |          | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called |  |  |
| 27   |          | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the  |  |  |
| 28   |          | request, even though the function entry was provided in the server module - the cbfunc will not  |  |  |
| 29   |          | be called  |  |  |
| 30   |          | • a PMIx error constant indicating either an error in the input or that the request was immediately  |  |  |
|  |          | processed and failed - the <i>cbfunc</i> will not be called  |  |  |
|  |          |  |  |  |
|  |          | Required Attributes  |  |  |
| 32   |          | If the PMIx library does not itself validate the credential, then it is required to pass any attributes  |  |  |
| 33   |          | provided by the client to the host environment for processing. In addition, it must include the  |  |  |
| 34   |          | following attributes in the passed <i>info</i> array:  |  |  |
| 35   |          | PMIX_USERID "pmix.euid" (uint32_t)   |  |  |

| 1        | Effective user id.  |
|----------|---|
| 2        | <pre>PMIX_GRPID "pmix.egid" (uint32_t)</pre>  |
| 3        | Effective group id.   |
| 4        |   |
| 5        | Host environments are not required to support any specific attributes.  |
|          | <b>A</b>  |
|          | → Optional Attributes   |
| 6        | The following attributes are optional for host environments that support this operation:  |
| 7        | PMIX_TIMEOUT "pmix.timeout" (int)   |
| 8        | Time in seconds before the specified operation should time out $(0 \text{ indicating infinite})$ in   |
| 9        | error. The timeout parameter can help avoid "hangs" due to programming errors that prevent  |
| 10       | the target process from ever exposing its data.   |
|          |   |
|          | Advice to PMIx library implementers —   |
| 11       | We recommend that implementation of the PMIX_TIMEOUT attribute be left to the host  |
| 12       | environment due to race condition considerations between completion of the operation versus   |
| 13       | internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT   |
| 14<br>15 | directly in the PMIx server library must take care to resolve the race condition and should avoid passing <b>PMIX_TIMEOUT</b> to the host environment so that multiple competing timeouts are not |
| 16       | created.  |
|          | ^   |
| 17       | Description   |
| 18       | Request validation of a credential obtained from the host environment via a prior call to the   |
| 9        | <pre>pmix_server_get_cred_fn_t module entry.</pre>  |
| . 4      | 1.0.05  |
| 20       | 1.2.25 pmix_server_iof_fn_t   |
| 01       | Summary   |

Request the specified IO channels be forwarded from the given array of processes.

```
Format
 1
   PMIx v3.0
 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,
                                               pmix_iof_channel_t channels,
 5
 6
                                               pmix op cbfunc t cbfunc, void *cbdata);
 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
               IN
                    directives
11
12
                    Array of pmix info t structures further defining the request (array of handles)
13
               IN
                    ndirs
14
                    Number of elements in the info array (integer)
                    channels
15
               IN
                    Bitmask identifying the channels to be forwarded ( pmix_iof_channel_t )
16
17
               IN
                    cbfunc
18
                    Callback function pmix_op_cbfunc_t (function reference)
19
               IN
                    cbdata
                    Data to be passed to the callback function (memory reference)
20
21
               Returns one of the following:
               • PMIX SUCCESS, indicating that the request is being processed by the host environment - result
22
23
                 will be returned in the provided cbfunc. Note that the library must not invoke the callback
                 function prior to returning from the API.
24
25

    PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and

                 returned success - the cbfunc will not be called
26
27
               • PMIX ERR NOT SUPPORTED, indicating that the host environment does not support the
28
                 request, even though the function entry was provided in the server module - the cbfunc will not
                 be called
29
30
               • a PMIx error constant indicating either an error in the input or that the request was immediately
                 processed and failed - the cbfunc will not be called
31
                                                Required Attributes
               The following attributes are required to be included in the passed info array:
32
               PMIX USERID "pmix.euid" (uint32 t)
33
                     Effective user id.
34
35
               PMIX GRPID "pmix.eqid" (uint32 t)
```

| 1                    | Effective group id.  |  |
|----------------------|--|--|
| 2                    |  |  |
| 3<br>4               | Host environments that provide this module entry point are required to support the following attributes:   |  |
| 5<br>6<br>7          | 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.  |  |
| 8<br>9               | PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)  In an overflow situation, drop the oldest bytes to make room in the cache.   |  |
| 10<br>11<br>12       | 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  |  |
| 13                   | The following attributes may be supported by a host environment.   |  |
| 14<br>15<br>16<br>17 | 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. |  |
| 19<br>20<br>21<br>22 | 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.  |  |
| 23<br>24<br>25<br>26 | <b>Description</b> 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 — — — — — — — — — — — — — — — — — — —  |  |
| 27<br>28<br>29       | The forwarding of stdin is a <i>push</i> process - processes cannot request that it be <i>pulled</i> from some other source. Requests including the <b>PMIX_FWD_STDIN_CHANNEL</b> channel will return a <b>PMIX_ERR_NOT_SUPPORTED</b> error.   |  |

# 11.2.26 pmix\_server\_stdin\_fn\_t

#### Summarv 3 Pass standard input data to the host environment for transmission to specified recipients. Format PMIx v3.0 5 typedef pmix status t (\*pmix server stdin fn t) ( 6 const pmix proc t \*source, const pmix\_proc\_t targets[], 7 size t ntargets, 8 9 const pmix info t directives[], 10 size t ndirs, const pmix\_byte\_object\_t \*bo, 11 pmix\_op\_cbfunc\_t cbfunc, void \*cbdata); 12 IN 13 source 14 pmix proc t structure of source process (handle) targets 15 IN Array of **pmix\_proc\_t** target identifiers (handle) 16 17 IN ntargets 18 Number of elements in the *targets* array (integer) 19 IN directives Array of info structures (array of handles) 20 IN ndirs 21 Number of elements in the *info* array (integer) 22 23 IN Pointer to pmix\_byte\_object\_t containing the payload (handle) 24 25 IN cbfunc Callback function pmix\_op\_cbfunc\_t (function reference) 26 27 IN Data to be passed to the callback function (memory reference) 28 29 Returns one of the following: • PMIX\_SUCCESS, indicating that the request is being processed by the host environment - result 30 will be returned in the provided cbfunc. Note that the library must not invoke the callback 31 function prior to returning from the API. 32 • PMIX\_OPERATION\_SUCCEEDED, indicating that the request was immediately processed and 33 returned success - the cbfunc will not be called 34 35 • PMIX ERR NOT SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not 36 37 be called

```
• a PMIx error constant indicating either an error in the input or that the request was immediately
 1
2
                processed and failed - the cbfunc will not be called
                  -----
                                            Required Attributes
3
              The following attributes are required to be included in the passed info array:
              PMIX USERID "pmix.euid" (uint32 t)
 4
 5
                   Effective user id.
6
              PMIX_GRPID "pmix.egid" (uint32_t)
 7
                    Effective group id.
              ▲-----
8
              Description
9
              Passes stdin to the host environment for transmission to specified recipients. The host environment
10
              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.
11
    11.2.27
                pmix server grp fn t
              Summary
13
14
              Request group operations (construct, destruct, etc.) on behalf of a set of processes.
              Format
15
   PMIx v4.0
16
              typedef pmix_status_t (*pmix_server_grp_fn_t)(
                                               pmix_group_operation_t op, char grp[],
17
18
                                               const pmix_proc_t procs[], size_t nprocs,
                                               const pmix_info_t directives[],
19
20
                                               size t ndirs,
                                               pmix_info_cbfunc_t cbfunc, void *cbdata);
21
22
              IN
                   op
23
                  pmix_group_operation_t value indicating operation the host is requested to perform
24
                  (integer)
              IN
25
                   grp
                  Character string identifying the group (string)
26
27
              IN
                  Array of pmix proc t identifiers of participants (handle)
28
29
              IN nprocs
                  Number of elements in the procs array (integer)
30
31
              IN
                   directives
32
                  Array of info structures (array of handles)
```

| 1<br>2<br>3<br>4<br>5<br>6             | <ul> <li>IN ndirs         Number of elements in the <i>info</i> array (integer)     </li> <li>IN cbfunc         Callback function pmix_info_cbfunc_t (function reference)     </li> <li>IN cbdata         Data to be passed to the callback function (memory reference)     </li> </ul>   |
|--|---|
| 7                                      | Returns one of the following:   |
| 8<br>9<br>0                            | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback function prior to returning from the API.   |
| 1<br>2                                 | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will not be called  |
| 3<br>4<br>5                            | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called  |
| 6<br>7                                 | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will not be called   |
|  | ▼ Optional Attributes   |
| 8                                      | The following attributes may be supported by a host environment.  |
| 9<br>20<br>21<br>22<br>23              | PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)  Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.   |
| 24<br>25<br>26<br>27<br>28<br>29<br>30 | PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)  Group operation only involves local processes. PMIx implementations are <i>required</i> to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false |
| 3<br>3                                 | <pre>PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)</pre>  |
| 34<br>35<br>36                         | PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)  Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false   |

```
PMIX_RANGE "pmix.range" (pmix_data_range_t)
 1
 2
                    Value for calls to publish/lookup/unpublish or for monitoring event notifications.
              The following attributes may be included in the host's response:
 3
              PMIX GROUP ID "pmix.grp.id" (char*)
 4
 5
                    User-provided group identifier
              PMIX GROUP_MEMBERSHIP "pmix.grp.mbrs" (pmix_data_array_t*)
6
 7
                    Array of group member ID's
              PMIX_GROUP_CONTEXT_ID "pmix.grp.ctxid" (size_t)
8
9
                    Context identifier assigned to the group by the host RM.
10
              PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)
                    Data collected to be shared during group construction
11
              Description
12
13
              Perform the specified operation across the identified processes, plus any special actions included in
              the directives. Return the result of any special action requests in the callback function when the
14
              operation is completed. Actions may include a request ( PMIX GROUP ASSIGN CONTEXT ID
15
              ) that the host assign a unique numerical (size t) ID to this group - if given, the PMIX RANGE
16
17
              attribute will specify the range across which the ID must be unique (default to
18
              PMIX RANGE SESSION).
    11.2.28 pmix_server_fabric_fn_t
              Summary
20
              Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.
21
22
              Format
   PMIx v4.0
23
              typedef pmix status t (*pmix server fabric fn t)(
                                                 const pmix proc t *requestor,
24
25
                                                 pmix_fabric_operation_t op,
                                                 const pmix_info_t directives[],
26
27
                                                 size_t ndirs,
```

pmix\_info\_cbfunc\_t cbfunc, void \*cbdata);

| 1             | IN requestor   |  |
|---------------|--|--|
| 2             | <pre>pmix_proc_t identifying the requestor (handle)</pre>  |  |
| 3             | IN op  |  |
| 4             | <pre>pmix_fabric_operation_t value indicating operation the host is requested to perform</pre>                   |  |
| 5             | (integer)  |  |
| 6             | IN directives  |  |
| 7             | Array of info structures (array of handles)  |  |
| 8             | IN ndirs   |  |
| 9             | Number of elements in the <i>info</i> array (integer)  |  |
| 0             | IN cbfunc  |  |
| 1             | Callback function <pre>pmix_info_cbfunc_t</pre> (function reference)   |  |
| 2             | IN cbdata  |  |
| 3             | Data to be passed to the callback function (memory reference)  |  |
| 4             | Returns one of the following:  |  |
| 5             | • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result                  |  |
| 6             | will be returned in the provided <i>cbfunc</i> . Note that the library must not invoke the callback              |  |
| 7             | function prior to returning from the API.  |  |
| 8             | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and                            |  |
| 9             | returned <i>success</i> - the <i>cbfunc</i> will not be called   |  |
| 20            | •  |  |
| 20            | • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the                              |  |
| ?1<br>?2      | request, even though the function entry was provided in the server module - the <i>cbfunc</i> will not be called |  |
| .2            | be caned   |  |
| 23            | • a PMIx error constant indicating either an error in the input or that the request was immediately              |  |
| 24            | processed and failed - the <i>cbfunc</i> will not be called  |  |
|               | Required Attributes  |  |
| 25            | The following directives are required to be supported by all hosts to aid users in identifying the               |  |
| 26            | fabric and (if applicable) the device to whom the operation references:  |  |
| . <del></del> |  |  |
| 27            | PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string)  |  |
| 28            | Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)  |  |
| .9            | <pre>PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)</pre>   |  |
| 30            | An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)                                      |  |
| <b>31</b>     | <pre>PMIX_FABRIC_PLANE "pmix.fab.plane" (char*)</pre>  |  |
| 32            | ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request                      |  |
| 3             | for information, specifies the plane whose information is to be returned. When used directly                     |  |
| 34            | in a request, returns a <b>pmix_data_array_t</b> of string identifiers for all fabric planes in                  |  |
| 5             | the system.  |  |
|               |  |  |
|               |  |  |

## Description

Perform the specified operation. Return the result of any requests in the callback function when the operation is completed. Operations may, for example, include a request for fabric information. See <code>pmix\_fabric\_t</code> for a list of expected information to be included in the response. Note that requests for device index are to be returned in the callback function's array of <code>pmix\_info\_t</code> using the <code>PMIX\_FABRIC\_DEVICE\_INDEX</code> attribute.

#### **CHAPTER 12**

# **Fabric Support Definitions**

As the drive for performance continues, interest has grown in both scheduling algorithms that take into account network locality of the allocated resources, and in optimizing collective communication patterns by structuring them to follow fabric topology. Several interfaces have been defined that are specifically intended to support WLMs (also known as *schedulers*) by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric.

In contrast, hierarchical collective operations require each process have global information about both its peers and the fabric. For example, one might aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches. Creating such optimized patterns relies on detailed knowledge of the fabric location of each participant.

PMIx supports these efforts by defining datatypes and attributes by which fabric coordinates for processes and devices can be obtained from the host SMS. When used in conjunction with the PMIx *instant on* methods, this results in the ability of a process to obtain the fabric coordinate of all other processes without incurring additional overhead associated with the publish/exchange of that information.

# 16 12.1 Fabric Support Constants

The following constants are defined for use in fabric-related events.

**PMIX\_FABRIC\_UPDATE\_PENDING** The PMIx server library has been alerted to a change in the fabric that requires updating of one or more registered **pmix\_fabric\_t** objects.

PMIX\_FABRIC\_UPDATED The PMIx server library has completed updating the entries of all
 affected pmix\_fabric\_t objects registered with the library. Access to the entries of those
 objects may now resume.

**PMIX\_FABRIC\_COORDS\_UPDATED** Fabric coordinates have been updated - the affected fabrics/planes are identified in the notification. Coordinates of processes and devices on those affected components should be refreshed prior to next use.

# 12.2 Fabric Support Datatypes

Several datatype definitions have been created to support fabric-related operations and information.

## 12.2.1 Fabric Coordinate Structure

} pmix coord t;

The pmix\_coord\_t structure describes the fabric coordinates of a specified process in a given view PMIx v4.0 typedef struct pmix\_coord { char \*fabric; char \*plane; pmix coord view t view; uint32 t \*coord; size t dims;

All coordinate values shall be expressed as unsigned integers due to their units being defined in fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

The fabric and plane fields are assigned by the fabric provider to help the user identify the fabric to which the coordinates refer. Note that providers are not required to assign any particular value to the fields and may choose to leave the fields blank. Example entries include {"Ethernet", "mgmt"} or {"infiniband", "data1"}.

## Advice to PMIx library implementers ————

Note that the <code>pmix\_coord\_t</code> structure does not imply nor mandate any requirement on how the coordinate data is to be stored within the PMIx library. Implementers are free to store the coordinate in whatever format they choose.

A fabric coordinate is usually associated with a given fabric device - e.g., a particular NIC on a node. Thus, while the fabric coordinate of a device must be unique in a given view, the coordinate may be shared by multiple processes on a node. If the node contains multiple fabric devices, then either the device closest to the binding location of a process shall be used as its coordinate, or (if the process is unbound or its binding is not known) all devices on the node shall be reported as a pmix data array t of pmix coord t structures.

Nodes with multiple fabric devices can also have those devices configured as multiple **fabric planes**. In such cases, a given process (even if bound to a specific location) may be associated with a coordinate on each plane. The resulting set of fabric coordinates shall be reported as a **pmix\_data\_array\_t** of **pmix\_coord\_t** structures. The caller may request a coordinate from a specific fabric plane by passing the **PMIX\_FABRIC\_PLANE** attribute as a directive/qualifier to the **PMIx Get** or **PMIx Query info** nb call.

#### 12.2.2 Fabric Coordinate Support Macros The following macros are provided to support the **pmix\_coord\_t** structure. 2 12.2.2.1 Initialize the pmix\_coord\_t structure 3 Initialize the **pmix** coord t fields 4 PMIx v4.0 PMIX\_COORD\_CONSTRUCT (m) 5 IN 6 m 7 Pointer to the structure to be initialized (pointer to pmix\_coord\_t) 12.2.2.2 Destruct the pmix\_coord\_t structure Destruct the **pmix** coord t fields 9 PMIx v4.0PMIX COORD DESTRUCT (m) 10 11 IN Pointer to the structure to be destructed (pointer to pmix coord t) 12 12.2.2.3 Create a pmix\_coord\_t array 13 14 Allocate and initialize a pmix\_coord\_t array PMIx v4.0 15 PMIX COORD CREATE (m, n) INOUT m 16 Address where the pointer to the array of **pmix\_coord\_t** structures shall be stored (handle) 17 IN 18 Number of structures to be allocated (size\_t) 19 12.2.2.4 20 Release a pmix\_coord\_t array Release an array of pmix\_coord\_t structures 21 PMIx v4.0 22 PMIX COORD FREE (m, n) 23 IN 24 Pointer to the array of **pmix\_coord\_t** structures (handle) IN 25 26 Number of structures in the array (size t)

# 1 12.2.3 Fabric Coordinate Views

| PMIx v4.0        | C  |
|------------------|--|
| 2                | <pre>typedef uint8_t pmix_coord_view_t;</pre>  |
| 3                | #define PMIX_COORD_VIEW_UNDEF 0x00   |
| 4                | #define PMIX_COORD_LOGICAL_VIEW 0x01   |
| 5                | #define PMIX_COORD_PHYSICAL_VIEW 0x02  |
|                  | C  |
| 6<br>7           | Fabric coordinates can be reported based on different <i>views</i> according to user preference at the time of request. The following views have been defined: |
| 8                | PMIX_COORD_VIEW_UNDEF The coordinate view has not been defined.  |
| 9                | PMIX_COORD_LOGICAL_VIEW The coordinates are provided in a <i>logical</i> view, typically   |
| 10               | given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by  |
| 11               | the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains,   |
| 12               | and other similar factors employed by that fabric.   |
| 13               | PMIX_COORD_PHYSICAL_VIEW The coordinates are provided in a <i>physical</i> view based on   |
| 14               | the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative  |
| 15               | position of that interface on the specific fabric cabling.   |
|                  | Advice to PMIx library implementers —  |
| 16               | PMIx library implementers are advised to avoid declaring the above constants as actual <b>enum</b>   |
| 17               | values in order to allow host environments to add support for possibly proprietary coordinate views.   |
| 18               |  |
|                  |  |
| 19               | If the requester does not specify a view, coordinates shall default to the logical view.   |
| 20 <b>12.2.4</b> | Fabric Link State  |
| 20 121211        |  |
| 21               | The pmix_link_state_t is a uint32_t type for fabric link states.   |
| PMIx v4.0        | C  |
| 22               | typedef uint8 t pmix link state t;   |
|                  | C  |
| 23               | The following constants can be used to set a variable of the type <b>pmix_link_state_t</b> . All   |
| 24               | definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state   |
| 25               | values start at zero.  |
|                  |  |
| 26               | PMIX_LINK_STATE_UNKNOWN The port state is unknown or not applicable.   |
| 27               | PMIX_LINK_DOWN The port is inactive.   |
| 28               | PMIX_LINK_UP The port is active.   |

# 12.2.5 Fabric Operation Constants

```
The pmix_fabric_operation_t structure is an enumerated type for specifying fabric
   PMIx v4.0
 3
                operations used in the PMIx server module's pmix_server_fabric_fn_t API. All values
 4
                were originally defined in version 4 of the standard unless otherwise marked.
 5
                PMIX FABRIC REQUEST INFO
                                                     Request information on a specific fabric - if the fabric isn't
 6
                     specified as per PMIx_Fabric_register, then return information on the system default
 7
                     fabric. Information to be returned is described in pmix fabric t.
                                                   Update information on a specific fabric - the index of the
 8
                PMIX FABRIC UPDATE INFO
 9
                     fabric ( PMIX FABRIC INDEX ) to be updated must be provided.
10
                PMIX_FABRIC_GET_VERTEX_INFO
                                                         Request information on a specific NIC within the
11
                     identified fabric - the index of the device ( PMIX FABRIC DEVICE INDEX ) and of the
12
                     fabric ( PMIX FABRIC INDEX ) must be provided. If the NIC identifier is not specified,
                     then return vertex info on all NICs in the fabric. Information to be included on each vertex is
13
14
                     described in pmix fabric t.
                                                 Advice to users -
15
                     Requesting information on every NIC in the fabric may be an expensive operation in terms of
16
                     both memory footprint and time.
17
                PMIX FABRIC GET DEVICE INDEX
                                                          Request the fabric-wide index (returned as
                     PMIX FABRIC DEVICE INDEX) for a specific NIC within the identified fabric based on
18
19
                     the provided vertex information. The index of the fabric must be provided.
```

# 20 12.2.6 Fabric registration structure

21

22

The **pmix\_fabric\_t** structure is used by a WLM to interact with fabric-related PMIx interfaces, and to provide information about the fabric for use in scheduling algorithms or other purposes.

```
PMIx v4.0

23 typedef struct pmix_fabric_s {
24 char *name;
25 size_t index;
26 pmix_info_t *info;
27 size_t ninfo;
28 void *module;
29 } pmix fabric t;
```

| 1                          | Note that in this structure:   |
|----------------------------|--|
| 2<br>3<br>4                | • <i>name</i> is an optional user-supplied string name identifying the fabric being referenced by this struct. If provided, the field must be a <b>NULL</b> -terminated string composed of standard alphanumeric values supported by common utilities such as <i>strcmp</i> .;   |
| 5                          | • <i>index</i> is a PMIx-provided number identifying this object;  |
| 6<br>7                     | • <i>info</i> is an array of <b>pmix_info_t</b> containing information (provided by the PMIx library) about the fabric;  |
| 8                          | • <i>ninfo</i> is the number of elements in the <i>info</i> array  |
| 9                          | • module points to an opaque object reserved for use by the PMIx server library.   |
| 10<br>11<br>12<br>13       | Note that only the <i>name</i> field is provided by the user - all other fields are provided by the PMIx library and must not be modified by the user. The <i>info</i> array contains a varying amount of information depending upon both the PMIx implementation and information available from the fabric vendor. At a minimum, it must contain (ordering is arbitrary): |
|                            | Required Attributes  |
| 14<br>15                   | PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string) Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)  |
| 16<br>17                   | PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string) An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)  |
| 18<br>19<br>20             | PMIX_FABRIC_NUM_VERTICES "pmix.fab.nverts" (size_t)  Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and columns) in the cost matrix  |
| 21                         | and may optionally contain one or more of the following:   |
|                            | ▼ Optional Attributes  |
| 22<br>23<br>24             | PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer)  Pointer to a two-dimensional array of point-to-point relative communication costs expressed as uint16_t values  |
| 25<br>26<br>27<br>28<br>29 | PMIX_FABRIC_GROUPS "pmix.fab.grps" (string)  A string delineating the group membership of nodes in the system, where each fabric group consists of the group number followed by a colon and a comma-delimited list of nodes in that group, with the groups delimited by semi-colons (e.g., 0:node000,node002,node004,node006;1:node001,node003,node005,node007)            |
| 30                         | PMIX_FABRIC_DIMS "pmix.fab.dims" (uint32_t)  |

| 1<br>2<br>3<br>4                | Number of dimensions in the specified fabric plane/view. If no plane is specified in a request, then the dimensions of all planes in the system will be returned as a <pre>pmix_data_array_t</pre> containing an array of <pre>uint32_t</pre> values. Default is to provide dimensions in <pre>logical</pre> view.  |
|---------------------------------|---|
| 5<br>6<br>7<br>8<br>9           | PMIX_FABRIC_PLANE "pmix.fab.plane" (char*)  ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly in a request, returns a pmix_data_array_t of string identifiers for all fabric planes in the system.   |
| 0<br>1<br>2<br>3<br>4<br>5<br>6 | PMIX_FABRIC_SHAPE "pmix.fab.shape" (pmix_data_array_t*)  The size of each dimension in the specified fabric plane/view, returned in a  pmix_data_array_t containing an array of uint32_t values. The size is defined as  the number of elements present in that dimension - e.g., the number of NICs in one  dimension of a physical view of a fabric plane. If no plane is specified, then the shape of  each plane in the system will be returned in an array of fabric shapes. Default is to provide  the shape in logical view. |
| 7<br>8                          | <pre>PMIX_FABRIC_SHAPE_STRING "pmix.fab.shapestr" (string) Network shape expressed as a string (e.g., "10x12x2").</pre>   |
| 9<br>20<br>21<br>22             | While unusual due to scaling issues, implementations may include an array of  PMIX_FABRIC_DEVICE elements describing the vertex information for each NIC in the system.  Each element shall contain a pmix_data_array_t of pmix_info_t values describing the device. Each array may contain one or more of the following (ordering is arbitrary):   |
| 23<br>24<br>25                  | <pre>PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string) The operating system name associated with the device. This may be a logical fabric interface name (e.g. eth0 or eno1) or an absolute filename.</pre>   |
| 26<br>27                        | <pre>PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string) Indicates the name of the vendor that distributes the NIC.</pre>   |
| 28<br>29                        | <pre>PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid" (string) This is a vendor-provided identifier for the device or product.</pre>   |
| 30<br>31                        | <pre>PMIX_HOSTNAME "pmix.hname" (char*) Name of the host (e.g., where a specified process is running, or a given device is located).</pre>  |
| 32<br>33                        | <pre>PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string) The name of the driver associated with the device</pre>  |
| 34<br>35                        | <pre>PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string) The device's firmware version</pre>  |
| 86<br>87<br>88                  | PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)  The primary link-level address associated with the NIC, such as a Media Access  Control (MAC) address. If multiple addresses are available, only one will be reported.  |

```
PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
 1
2
                     The maximum transfer unit of link level frames or packets, in bytes.
               PMIX FABRIC DEVICE SPEED "pmix.fabdev.speed" (size t)
 3
 4
                     The active link data rate, given in bits per second.
               PMIX FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )
5
6
                     The last available physical port state. Possible values are PMIX LINK STATE UNKNOWN,
7
                     PMIX LINK DOWN, and PMIX LINK UP, to indicate if the port state is unknown or not
8
                     applicable (unknown), inactive (down), or active (up).
9
               PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
10
                     Specifies the type of fabric interface currently active on the device, such as Ethernet or
                     InfiniBand.
11
12
               PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
                     The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
13
14
               PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
                     A node-level unique identifier for a Peripheral Component Interconnect (PCI) device.
15
                     Provided only if the device is located on a PCI bus. The identifier is constructed as a
16
17
                     four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit
18
                     device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus, an
19
                     example identifier might be "abc1:0f:23:01". The combination of node identifier (
20
                     PMIX HOSTNAME OF PMIX NODEID ) and PMIX FABRIC DEVICE PCI DEVID
                     shall be unique within the system.
21
```

# 22 12.3 Fabric Support Attributes

The following attributes are used by the library supporting the system's WLM to either access or return fabric-related information (e.g., as part of the **pmix fabric t** structure).

```
PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
```

Server requests access to WLM-supporting features - passed solely to the **PMIx\_server\_init** API to indicate that the library is to be initialized for scheduler support.

```
PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer)
```

Pointer to a two-dimensional array of point-to-point relative communication costs expressed as uint16\_t values

```
PMIX FABRIC GROUPS "pmix.fab.grps" (string)
```

A string delineating the group membership of nodes in the system, where each fabric group consists of the group number followed by a colon and a comma-delimited list of nodes in that group, with the groups delimited by semi-colons (e.g.,

0:node000,node002,node004,node006;1:node001,node003,node005,node007)

23

24 25

26

27

28

29

30 31

32

33

34

The following attributes may be returned by calls to the scheduler-related APIs or in response to 1 2 queries (e.g., PMIx Get or PMIx Query info) made by processes or tools. PMIX\_FABRIC\_VENDOR "pmix.fab.vndr" (string) 3 Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel) 4 PMIX\_FABRIC\_IDENTIFIER "pmix.fab.id" (string) 5 6 An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1) PMIX\_FABRIC\_INDEX "pmix.fab.idx" (size\_t) 7 The index of the fabric as returned in pmix fabric t 8 PMIX FABRIC NUM VERTICES "pmix.fab.nverts" (size t) 9 Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and 10 columns) in the cost matrix 11 PMIX\_FABRIC\_COORDINATE "pmix.fab.coord" (pmix\_data\_array\_t) 12 Fabric coordinate(s) of the specified process in the view and/or plane provided by the 13 requester. If only one NIC has been assigned to the specified process, then the array will 14 contain only one address. Otherwise, the array will contain the coordinates of all NICs 15 16 available to the process in order of least to greatest distance from the process (NICs equally distant from the process will be listed in arbitrary order). 17 PMIX FABRIC VIEW "pmix.fab.view" (pmix coord view t) 18 Fabric coordinate view to be used for the requested coordinate - see 19 20 pmix\_coord\_view\_t for the list of accepted values. PMIX\_FABRIC\_DIMS "pmix.fab.dims" (uint32\_t) 21 Number of dimensions in the specified fabric plane/view. If no plane is specified in a 22 request, then the dimensions of all planes in the system will be returned as a 23 pmix\_data\_array\_t containing an array of uint32\_t values. Default is to provide 24 25 dimensions in logical view. PMIX\_FABRIC\_PLANE "pmix.fab.plane" (char\*) 26 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request 27 for information, specifies the plane whose information is to be returned. When used directly 28 in a request, returns a pmix\_data\_array\_t of string identifiers for all fabric planes in 29 30 the system. PMIX FABRIC\_ENDPT "pmix.fab.endpt" (pmix\_data\_array\_t) 31 32 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given process (e.g., in the case where multiple NICs are associated with a package to which the 33 process is bound), the returned values will be provided in a pmix\_data\_array\_t - the 34 returned data type of the individual values in the array varies by fabric provider. 35 PMIX\_FABRIC\_SHAPE "pmix.fab.shape" (pmix\_data\_array\_t\*) 36 The size of each dimension in the specified fabric plane/view, returned in a 37 pmix\_data\_array\_t containing an array of uint32\_t values. The size is defined as 38 the number of elements present in that dimension - e.g., the number of NICs in one 39 40 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of each plane in the system will be returned in an array of fabric shapes. Default is to provide 41 the shape in logical view. 42 PMIX FABRIC SHAPE\_STRING "pmix.fab.shapestr" (string) 43

```
Network shape expressed as a string (e.g., "10x12x2").
 1
               The following attributes are used to describe devices (a.k.a., NICs) attached to the fabric.
2
               PMIX_FABRIC_DEVICE "pmix.fabdev" ( pmix_data_array_t )
 3
                     An array of pmix_info_t describing a particular fabric device (NIC).
 4
               PMIX_FABRIC_DEVICE_INDEX "pmix.fabdev.idx" (uint32_t)
5
                     System-unique index of a particular fabric device (NIC).
6
7
               PMIX FABRIC DEVICE NAME "pmix.fabdev.nm" (string)
                     The operating system name associated with the device. This may be a logical fabric interface
8
                     name (e.g. eth0 or eno1) or an absolute filename.
9
               PMIX FABRIC DEVICE VENDOR "pmix.fabdev.vndr" (string)
10
                     Indicates the name of the vendor that distributes the NIC.
11
               PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
12
                     The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
13
               PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid" (string)
14
15
                     This is a vendor-provided identifier for the device or product.
               PMIX FABRIC DEVICE DRIVER "pmix.fabdev.driver" (string)
16
                     The name of the driver associated with the device
17
18
               PMIX FABRIC DEVICE FIRMWARE "pmix.fabdev.fmwr" (string)
19
                     The device's firmware version
               PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)
20
                     The primary link-level address associated with the NIC, such as a MAC address. If multiple
21
                     addresses are available, only one will be reported.
22
               PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
23
24
                     The maximum transfer unit of link level frames or packets, in bytes.
25
               PMIX FABRIC DEVICE SPEED "pmix.fabdev.speed" (size t)
26
                     The active link data rate, given in bits per second.
               PMIX FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )
27
28
29
                     The last available physical port state. Possible values are PMIX_LINK_STATE_UNKNOWN,
                     PMIX LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not
30
                     applicable (unknown), inactive (down), or active (up).
31
               PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
32
33
                     Specifies the type of fabric interface currently active on the device, such as Ethernet or
                     InfiniBand.
34
               PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
35
                     A node-level unique identifier for a PCI device. Provided only if the device is located on a
36
                     PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
37
38
                     the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
                     zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
39
                     combination of node identifier ( PMIX_HOSTNAME or PMIX_NODEID ) and
40
                     PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the system.
41
```

# 12.4 Fabric Support Functions

The following APIs allow the WLM to request specific services from the fabric subsystem via the PMIx library.

#### Advice to PMIx server hosts -

Due to their high cost in terms of execution, memory consumption, and interactions with other SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation of these APIs be restricted to a single PMIx server in a system that is supporting the SMS component responsible for the scheduling of allocations (i.e., the system <code>scheduler</code>). The <code>PMIX\_SERVER\_SCHEDULER</code> attribute can be used for this purpose to control the execution path. Clients, tools, and other servers utilizing these functions are advised to have their requests forwarded to the server supporting the scheduler using the <code>pmix\_server\_fabric\_fn\_t</code> server module function, as needed.

## 12.4.1 PMIx\_Fabric\_register

#### Summary

Register for access to fabric-related information.

#### **Format**

PMIx v4.0

4

5

6

7

8

9

10

11

13

14

15

16

17

18

19

20

21

22 23

24

25

26 27

28

pmix\_status\_t

PMIx\_Fabric\_register(pmix\_fabric\_t \*fabric,

const pmix\_info\_t directives[],

size\_t ndirs)

#### IN fabric

address of a **pmix\_fabric\_t** (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

#### IN directives

an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**, then the highest priority available fabric will be used (array of handles)

#### IN ndirs

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

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

## Required Attributes

The following directives are required to be supported by all PMIx libraries to aid users in identifying the fabric whose data is being sought:

### PMIX\_FABRIC\_PLANE "pmix.fab.plane" (char\*)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly in a request, returns a <code>pmix\_data\_array\_t</code> of string identifiers for all fabric planes in the system.

```
PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)
```

An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)

### PMIX\_FABRIC\_VENDOR "pmix.fab.vndr" (string)

Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)

### **Description**

Register for access to fabric-related information, including the communication cost matrix. This call must be made prior to requesting information from a fabric. The caller may request access to a particular fabric using the vendor, type, or identifier, or to a specific **fabric plane** via the **PMIX\_FABRIC\_PLANE** attribute - otherwise, the default fabric will be returned.

For performance reasons, the PMIx library does not provide thread protection for accessing the information in the <code>pmix\_fabric\_t</code> structure. Instead, the PMIx implementation shall provide two methods for coordinating updates to the provided fabric information:

- Users may periodically poll for updates using the PMIx Fabric update API
- Users may register for PMIX\_FABRIC\_UPDATE\_PENDING events indicating that an update to
  the cost matrix is pending. When received, users are required to terminate or pause any actions
  involving access to the cost matrix before returning from the event. Completion of the
  PMIX\_FABRIC\_UPDATE\_PENDING event handler indicates to the PMIx library that the
  fabric object's entries are available for updating. This may include releasing and re-allocating
  memory as the number of vertices may have changed (e.g., due to addition or removal of one or
  more NICs). When the update has been completed, the PMIx library will generate a
  PMIX\_FABRIC\_UPDATED event indicating that it is safe to begin using the updated fabric
  object(s).

There is no requirement that the caller exclusively use either one of these options. For example, the user may choose to both register for fabric update events, but poll for an update prior to some critical operation.

#### 12.4.2 PMIx\_Fabric\_update Summary 2 3 Update fabric-related information. **Format** PMIx v4.05 pmix\_status\_t PMIx\_Fabric\_update(pmix\_fabric\_t \*fabric) 6 IN 7 fabric address of a pmix\_fabric\_t (backed by storage) (handle) 8 9 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant. 10 **Description** Update fabric-related information. This call can be made at any time to request an update of the 11 fabric information contained in the provided **pmix\_fabric\_t** object. The caller is not allowed 12 to access the provided **pmix\_fabric\_t** until the call has returned. 13 12.4.3 PMIx Fabric deregister Summary 15 16 Deregister a fabric object. Format 17 PMIx v4.0pmix\_status\_t PMIx\_Fabric\_deregister(pmix\_fabric\_t \*fabric) 18 IN 19 input 20 address of a pmix fabric t (handle) 21 Returns **PMIX SUCCESS** or a negative value corresponding to a PMIx error constant. Description 22 23 Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information (e.g., cost matrix) associated with it. Contents of the provided pmix fabric t will be 24 25 invalidated upon function return. 12.4.4 PMIx\_Fabric\_get\_vertex\_info 26 27 Summary 28 Given a communication cost matrix index for a specified fabric, return the corresponding vertex 29 info.

| 1                | Format   |
|------------------|--|
| <i>PMIx v4.0</i> |  |
| 2                | pmix_status_t  |
| 3                | <pre>PMIx_Fabric_get_vertex_info(pmix_fabric_t *fabric, uint32_t index,</pre>                  |
| 4                | <pre>pmix_info_t **info, size_t *ninfo)</pre>  |
|                  | G —  |
| 5                | IN fabric  |
| 6                | address of a pmix_fabric_t (handle)  |
| 7                | IN index   |
| 8                | vertex index (i.e., communication cost matrix row or column number) (integer)                  |
| 9                | INOUT info   |
| 0                | Address where a pointer to an array of <b>pmix_info_t</b> containing the results of the query  |
| 1                | can be returned (memory reference)   |
| 2<br>3           | INOUT ninfo Address where the number of elements in <i>info</i> can be returned (handle)       |
| 3                |  |
| 4                | Returns one of the following:  |
| 5                | • PMIX_SUCCESS, indicating return of a valid value.  |
| 6                | • PMIX_ERR_BAD_PARAM, indicating that the provided index is out of bounds.                     |
| 7                | • a PMIx error constant indicating either an error in the input or that the request failed.    |
| 8                | Description  |
| 9                | Query information about a specified vertex (fabric device, or NIC) in the system. The returned |
| 20               | status indicates if requested data was found or not. The returned array of pmix_info_t will    |
| 21               | contain information on the specified vertex - the exact contents will depend on the PMIx       |
| 22               | implementation and the fabric vendor. At a minimum, it must contain sufficient information to  |
| 23               | uniquely identify the device within the system (ordering is arbitrary):                        |
|                  | Required Attributes  |
| 24               | PMIX_HOSTNAME "pmix.hname" (char*)   |
| 25               | Name of the host (e.g., where a specified process is running, or a given device is located).   |
| 26               | The <b>PMIX_NODEID</b> may be returned in its place, or in addition to the hostname.           |
| 27               | PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string)  |
| 28               | The operating system name associated with the device. This may be a logical fabric interface   |
| <b>.</b> 9       | name (e.g. eth0 or eno1) or an absolute filename.  |
| 30               | PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string)  |
| 11               | Indicates the name of the vendor that distributes the NIC.                                     |
| 32               | PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)  |
| 33               | The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").                        |
| 34               | PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)                                   |
|                  | (  |

| 1<br>2<br>3    | A node-level unique identifier for a PCI device. Provided only if the device is located on a PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in |
|----------------|---|
| 4              | zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The   |
| 5              | combination of node identifier ( PMIX_HOSTNAME or PMIX_NODEID ) and   |
| 6              | PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the system. This item   |
| 7              | should be included if the device bus type is PCI - the equivalent should be provided for any  |
| 8              | other bus type.   |
| 9              | The returned array may optionally contain one or more of the following:   |
|                | ▼ Optional Attributes   |
| 10<br>11       | <pre>PMIX_FABRIC_DEVICE_ID "pmix.fabdev.devid" (string) This is a vendor-provided identifier for the device or product.</pre>   |
| 12<br>13       | <pre>PMIX_FABRIC_DEVICE_DRIVER "pmix.fabdev.driver" (string) The name of the driver associated with the device</pre>  |
| 14<br>15       | <pre>PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string) The device's firmware version</pre>  |
| 16<br>17<br>18 | <pre>PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string) The primary link-level address associated with the NIC, such as a MAC address. If multiple addresses are available, only one will be reported.</pre>  |
| 19<br>20       | <pre>PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t) The maximum transfer unit of link level frames or packets, in bytes.</pre>   |
| 21<br>22       | <pre>PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t) The active link data rate, given in bits per second.</pre>   |
| 23             | <pre>PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )</pre>   |
| 24             | The last available physical port state. Possible values are PMIX_LINK_STATE_UNKNOWN,  |
| 25             | PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not   |
| 26             | applicable (unknown), inactive (down), or active (up).  |
| 27             | <pre>PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)</pre>  |
| 28             | Specifies the type of fabric interface currently active on the device, such as Ethernet or  |
| 29             | InfiniBand.   |

The caller is responsible for releasing the returned array.

# 12.4.5 PMIx\_Fabric\_get\_device\_index

#### Summary 2 3 Given vertex info, return the corresponding communication cost matrix index. Format PMIx v4.0 5 pmix status t 6 PMIx Fabric get device index(pmix fabric t \*fabric, 7 const pmix info t vertex[], size t ninfo, 8 uint32 t \*index) C 9 IN fabric 10 address of a pmix\_fabric\_t (handle) IN 11 array of pmix\_info\_t containing info describing the vertex whose index is being queried 12 (handle) 13 IN ninfo 14 number of elements in vertex 15 OUT index 16 pointer to the location where the index is to be returned (memory reference (handle)) 17 18 Returns one of the following: 19 • PMIX\_SUCCESS, indicating return of a valid value. 20 • a PMIx error constant indicating either an error in the input or that the request failed. Description 21 22 Query the index number of a vertex corresponding to the provided description. The description must provide adequate information to uniquely identify the target vertex. At a minimum, this must 23 24 include identification of the node hosting the device using either the PMIX HOSTNAME or 25 **PMIX NODEID**, plus a node-level unique identifier for the device (e.g., the 26 PMIX FABRIC DEVICE PCI DEVID for a PCI device).

### **CHAPTER 13**

# **Process Sets and Groups**

PMIx supports two slightly related, but functionally different concepts known as *process sets* and *process groups*. This chapter these two concepts and describes how they are utilized, along with their corresponding APIs.

# 13.1 Process Sets

A PMIx *Process Set* is a user-provided label associated with a given set of application processes. Definition of a PMIx process set typically occurs at time of application execution - e.g., on a PRRTE command line:

\$ 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.

- Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are involved in an operation are translated by their PMIx server into their *native* identifier prior to the operation being passed to the host environment. For example, an application can define a process group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the group id of *foo*. If the application subsequently calls the PMIx\_Fence API with a process identifier of {foo, PMIX\_RANK\_WILDCARD}, the PMIx server will replace that identifier with an array consisting of {210456, 0} and {210456, 1} the host-assigned identifiers of the participating processes prior to passing the request up to the host environment
- Process groups can request that the host environment assign a unique size\_t PGCID to the
  group at time of group construction. An MPI library may, for example, use the PGCID as the
  MPI communicator identifier for the group.

The two concepts do, however, overlap in one specific area. Process *groups* are included in the process *set* information returned by calls to **PMIx\_Query\_info\_nb**. Thus, a *process group* can effectively be considered an extended version of a *process set* that adds dynamic definition and operational context to the *process set* concept.

Advice to PMIx library implementers

PMIx implementations are required to include all active *group* identifiers in the returned list of process *set* names provided in response to the appropriate PMIx\_Query\_info\_nb call.

# 13.2 Process Groups

PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for purposes such as passing events or participating in PMIx fence operations. As with processes that assemble via PMIx\_Connect, each member of the group is provided with both the job-level information of any other namespace represented in the group, and the contact information for all group members. However, *groups* differ from PMIx\_Connect assemblages in the following key areas:

- Relation to the host environment
  - Calls to PMIx\_Connect are relayed to the host environment. This means that the host RM should treat the failure of any process in the specified assemblage as a reportable event and take appropriate action. However, the environment is not required to define a new identifier for the connected assemblage or any of its member processes, nor does it define a new rank for each process within that assemblage. In addition, the PMIx server does not provide any tracking support for the assemblage. Thus, the caller is responsible for addressing members of the connected assemblage using their RM-provided identifiers.

Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed, the server creates a tracker that associates the specified processes with the user-provided group identifier, and assigns a new group rank based on their relative position in the array of processes provided in the call to PMIx\_Group\_construct. Members of the group can subsequently utilize the group identifier in PMIx function calls to address the group's members, using either PMIX\_RANK\_WILDCARD to refer to all of them or the group-level rank of specific members. The PMIx server will translate the specified processes into their RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment has no visibility into the group's existence or membership.

### Advice to users

User-provided group identifiers must be distinct from anything provided by the RM so as to avoid collisions between group identifiers and RM-assigned namespaces. This can usually be accomplished through the use of an application-specific prefix - e.g., "myapp-foo"

### • Construction procedure

- PMIx\_Connect calls require that every process call the API before completing i.e., it is modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a given application thread can only be involved in one connect/accept operation at a time, and is blocked in that operation until all specified processes participate. In addition, there is no provision for replacing processes in the assemblage due to failure to participate, nor a mechanism by which a process might decline participation.
- PMIx Groups are designed to be more flexible in their construction procedure by relaxing these constraints. While a standard blocking form of constructing groups is provided, the event notification system is utilized to provide a designated *group leader* with the ability to replace participants that fail to participate within a given timeout period. This provides a mechanism by which the application can, if desired, replace members on-the-fly or allow the group to proceed with partial membership. In such cases, the final group membership is returned to all participants upon completion of the operation.

Additionally, PMIx supports dynamic definition of group membership based on an invite/join model. A process can asynchronously initiate construction of a group of any processes via the <code>PMIx\_Group\_invite</code> function call. Invitations are delivered via a PMIx event (using the <code>PMIX\_GROUP\_INVITED</code> event) to the invited processes which can then either accept or decline the invitation using the <code>PMIx\_Group\_join</code> API. The initiating process tracks responses by registering for the events generated by the call to <code>PMIx\_Group\_join</code>, timeouts, or process terminations, optionally replacing processes that decline the invitation, fail to respond in time, or terminate without responding. Upon completion of the operation, the final list of participants is communicated to each member of the new group.

### Destruct procedure

- Processes that assemble via PMIx\_Connect must all depart the assemblage together i.e., no member can depart the assemblage while leaving the remaining members in it. Even the non-blocking form of PMIx\_Disconnect retains this requirement in that members remain a part of the assemblage until all members have called PMIx\_Disconnect\_nb
- Members of a PMIx Group may depart the group at any time via the PMIx\_Group\_leave API. Other members are notified of the departure via the PMIX\_GROUP\_LEFT event to distinguish such events from those reporting process termination. This leaves the remaining members free to continue group operations. The PMIx\_Group\_destruct operation offers a collective method akin to PMIx\_Disconnect for deconstructing the entire group.

Note that applications supporting dynamic group behaviors such as asynchronous departure take responsibility for ensuring global consistency in the group definition prior to executing group collective operations - i.e., it is the application's responsibility to either ensure that knowledge of the current group membership is globally consistent across the participants, or to register for appropriate events to deal with the lack of consistency during the operation.

In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected* when constructed via **PMIx\_Connect**. The relevant APIs are explained below.

#### Advice to users

The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs must register for the corresponding events. Failure to do so will likely lead to operational failures. Users are recommended to utilize the PMIX\_TIMEOUT directive (or retain an internal timer) on calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have not registered for required events will never respond.

# 13.2.1 Group Operation Constants

The **pmix\_group\_operation\_t** structure is an enumerated type for specifying group operations. All values were originally defined in version 4 of the standard unless otherwise marked.

**PMIX\_GROUP\_DECLINE** Decline an invitation to join a PMIx group - provided for readability of user code

**PMIX\_GROUP\_ACCEPT** Accept an invitation to join a PMIx group - provided for readability of user code

**PMIX\_GROUP\_CONSTRUCT** Construct a group composed of the specified processes - used by a PMIx server library to direct host operation

**PMIX\_GROUP\_DESTRUCT** Destruct the specified group - used by a PMIx server library to direct host operation

#### 13.2.2 PMIx\_Group\_construct Summary Construct a PMIx process group 3 Format PMIx v4.0 5 pmix status t 6 PMIx Group construct(const char grp[], const pmix\_proc\_t procs[], size\_t nprocs, 7 const pmix info t directives[], size t ndirs, 8 pmix info t \*\*results, size t \*nresults) 9 C IN 10 grp NULL-terminated character array of maximum size PMIX MAX NSLEN containing the 11 group identifier (string) 12 IN procs 13 Array of pmix proc t structures containing the PMIx identifiers of the member processes 14 (array of handles) 15 IN nprocs 16 Number of elements in the *procs* array (size\_t) 17 18 IN directives 19 Array of pmix info t structures (array of handles) IN ndirs 20 Number of elements in the *directives* array (size\_t) 21 **INOUT** results 22 23 Pointer to a location where the array of **pmix\_info\_t** describing the results of the operation is to be returned (pointer to handle) 24 **INOUT** nresults 25 Pointer to a size\_t location where the number of elements in results is to be returned 26 27 (memory reference) 28 Returns one of the following: 29 • PMIX\_SUCCESS, indicating that the request has been successfully completed 30 • PMIX\_ERR\_NOT\_SUPPORTED The PMIx library and/or the host RM does not support this

• a PMIx error constant indicating either an error in the input or that the request failed to be

31

32

33

operation

completed

CHAPTER 13. PROCESS SETS AND GROUPS

|                                      | Required Attributes  |
|--------------------------------------|--|
| 1<br>2                               | The following attributes are <i>required</i> to be supported by all PMIx libraries that support this operation:  |
| 3<br>4                               | PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)  This process is the leader of the group   |
| 5<br>6<br>7                          | PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)  Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false  |
| 8<br>9<br>10<br>11<br>12<br>13<br>14 | PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)  Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false |
| 16                                   | Host environments that support this operation are required to provide the following attributes:  |
| 17<br>18<br>19<br>20<br>21           | PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)  Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.  |
| 22<br>23<br>24                       | PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)  Notify remaining members when another member terminates without first leaving the group.  The default is false  |
|                                      | ▼ Optional Attributes  |
| 25                                   | The following attributes are optional for host environments that support this operation:   |
| 26<br>27<br>28<br>29                 | PMIX_TIMEOUT "pmix.timeout" (int)  Time in seconds before the specified operation should time out (0 indicating infinite) in error. The timeout parameter can help avoid "hangs" due to programming errors that prevent the target process from ever exposing its data.  |

## Advice to PMIx library implementers -

We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

### **Description**

Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less than or equal to **PMIX\_MAX\_NSLEN**. Only characters accepted by standard string comparison functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

If the PMIX\_GROUP\_NOTIFY\_TERMINATION attribute is provided and has a value of true, then either the construct leader (if PMIX\_GROUP\_LEADER is provided) or all participants who register for the PMIX\_GROUP\_MEMBER\_FAILED event will receive events whenever a process fails or terminates prior to calling PMIX\_Group\_construct – i.e. if a group leader is declared, only that process will receive the event. In the absence of a declared leader, all specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the PMIX\_GROUP\_MEMBERSHIP attribute) as part of the *results* array returned by this API.

Failure of the declared leader at any time will cause a PMIX\_GROUP\_LEADER\_FAILED event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the PMIX\_GROUP\_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX\_GROUP\_LEADER\_SELECTED event identifying the new leader. If no leader was selected, then the pmix\_info\_t provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns **PMIX\_GROUP\_CONSTRUCT\_ABORT** from either the **PMIX\_GROUP\_MEMBER\_FAILED** or the **PMIX\_GROUP\_LEADER\_FAILED** event handler will

cause the construct process to abort, returning from the call with a **PMIX\_GROUP\_CONSTRUCT\_ABORT** status.

If the PMIX\_GROUP\_NOTIFY\_TERMINATION attribute is not provided or has a value of false, then the PMIx\_Group\_construct operation will simply return an error whenever a proposed group member fails or terminates prior to calling PMIx\_Group\_construct.

Providing the PMIX\_GROUP\_OPTIONAL attribute with a value of true directs the PMIx library to consider participation by any specified group member as non-required - thus, the operation will return PMIX\_SUCCESS if all members participate, or PMIX\_ERR\_PARTIAL\_SUCCESS if some members fail to participate. The results array will contain the final group membership in the latter case. Note that this use-case can cause the operation to hang if the PMIX\_TIMEOUT attribute is not specified and one or more group members fail to call PMIx\_Group\_construct while continuing to execute. Also, note that no leader or member failed events will be generated during the operation.

Processes in a group under construction are not allowed to leave the group until group construction is complete. Upon completion of the construct procedure, each group member will have access to the job-level information of all namespaces represented in the group plus any information posted via <code>PMIx\_Put</code> (subject to the usual scoping directives) for every group member.

# ——— Advice to PMIx library implementers —

At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related information from each participating namespace plus any information posted by group members via <code>PMIx\_Put</code> (subject to scoping directives) is available to each member via calls to <code>PMIx\_Get</code>.

### Advice to PMIx server hosts —

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a PMIx\_Group\_construct and a PMIx\_Fence operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

# 13.2.3 PMIx\_Group\_construct\_nb

## Summary

Non-blocking form of PMIx\_Group\_construct

| 1        | Format   |
|----------|--|
| PMIx v4. |  |
| 2        | pmix_status_t  |
| 3        | <pre>PMIx_Group_construct_nb(const char grp[],</pre>   |
| 4        | <pre>const pmix_proc_t procs[], size_t nprocs,</pre>   |
| 5        | <pre>const pmix_info_t directives[], size_t ndire</pre>  |
| 6        | <pre>pmix_info_cbfunc_t cbfunc, void *cbdata)</pre>  |
|          | 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 <i>procs</i> 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 <i>directives</i> array (size_t)                                       |
| 19       | IN cbfunc  |
| 20       | Callback function <b>pmix_info_cbfunc_t</b> (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 <i>cbfunc</i>    |
| 30       | will <i>not</i> 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<br>2                                 | <ul> <li>PMIX_ERR_PARTIAL_SUCCESS The operation succeeded but not all specified members<br/>participated - the final group membership is included in the callback function</li> </ul>  |
|--|--|
| 3<br>4                                 | • PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM does not.  |
| 5                                      | • a non-zero PMIx error constant indicating a reason for the request's failure   |
|  | ▼ Required Attributes  |
| 6<br>7                                 | PMIx libraries that choose not to support this operation <i>must</i> return <b>PMIX_ERR_NOT_SUPPORTED</b> when the function is called.   |
| 8<br>9                                 | The following attributes are <i>required</i> to be supported by all PMIx libraries that support this operation:  |
| 10<br>11                               | PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)  This process is the leader of the group   |
| 12<br>13<br>14                         | <pre>PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool) Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false</pre>  |
| 15<br>16<br>17<br>18<br>19<br>20<br>21 | PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)  Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false |
| 23                                     | Host environments that support this operation are required to provide the following attributes:  |
| 24<br>25<br>26<br>27<br>28             | PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)  Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.  |
| 29<br>30<br>31                         | <pre>PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)    Notify remaining members when another member terminates without first leaving the group.    The default is false</pre>   |

|    |        | The strict of th |
|----|--------|--|
| 1  |        | The following attributes are optional for host environments that support this operation:   |
| 2  |        | PMIX_TIMEOUT "pmix.timeout" (int)  |
| 3  |        | Time in seconds before the specified operation should time out (0 indicating infinite) in  |
| 4  |        | error. The timeout parameter can help avoid "hangs" due to programming errors that prevent   |
| 5  |        | the target process from ever exposing its data.  |
|    |        | <b>^^</b>  |
|    |        | Advice to PMIx library implementers  |
| 6  |        | We recommend that implementation of the <b>PMIX_TIMEOUT</b> attribute be left to the host  |
| 7  |        | environment due to race condition considerations between completion of the operation versus  |
| 8  |        | internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT  |
| 9  |        | directly in the PMIx server library must take care to resolve the race condition and should avoid  |
| 10 |        | passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not   |
| 11 |        | created.   |
|    |        |  |
| 12 |        | Description  |
| 13 |        | Non-blocking version of the <b>PMIx_Group_construct</b> operation. The callback function will  |
| 14 |        | be called once all group members have called either PMIx_Group_construct or  |
| 15 |        | PMIx_Group_construct_nb.   |
| 16 | 13.2.4 | PMIx_Group_destruct  |
| 17 |        | Summary  |
| 18 |        | Destruct a PMIx process group  |

|           | Format   |
|-----------|--|
| PMIx v4.0 | <b>▼</b>   |
|           | pmix_status_t  |
|           | <pre>PMIx_Group_destruct(const char grp[],</pre>   |
|           | <pre>const pmix_info_t directives[], size_t ndirs)</pre>   |
|           | <u> </u>   |
|           | IN grp   |
|           | NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the                              |
|           | identifier of the group to be destructed (string)  |
|           | IN directives  |
|           | Array of pmix_info_t structures (array of handles)  IN ndirs   |
|           | Number of elements in the <i>directives</i> array (size_t)   |
|           | Returns one of the following:  |
|           | • PMIX_SUCCESS, indicating that the request has been successfully completed                                |
|           | • PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this                         |
|           | operation  |
|           | • a PMIx error constant indicating either an error in the input or that the request failed to be completed |
|           | Required Attributes  |
|           | For implementations and host environments that support the operation, there are no identified              |
|           | required attributes for this API.  |
|           | <b>A</b>   |
|           | ▼ 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 ( $\theta$ 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 v4.0  |

## Advice to PMIx library implementers -

We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

### **Description**

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling PMIx\_Group\_destruct or its non-blocking version unless the PMIX\_GROUP\_NOTIFY\_TERMINATION attribute was provided (with a value of false) at time of group construction. If notification was requested, then the PMIX\_GROUP\_MEMBER\_FAILED event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The PMIX\_Group\_destruct operation will subsequently return PMIX\_SUCCESS when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

## Advice to PMIx server hosts -

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a PMIx\_Group\_destruct and a PMIx\_Fence operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

# 13.2.5 PMIx\_Group\_destruct\_nb

#### Summary

Non-blocking form of PMIx\_Group\_destruct

| 1              |           | Format   |
|----------------|-----------|--|
|                | PMIx v4.0 | <u> </u>   |
| 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)   |
|                |           |  |
| 6              |           | IN grp   |
| 7              |           | <b>NULL</b> -terminated character array of maximum size <b>PMIX_MAX_NSLEN</b> 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<br>12       |           | Number of elements in the <i>directives</i> 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<br>19<br>20 |           | • <b>PMIX_SUCCESS</b> , indicating that the request is being processed - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API. |
| 21<br>22       |           | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called  |
| 23<br>24       |           | • PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called  |
| 25<br>26       |           | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called   |
| 27<br>28       |           | If executed, the status returned in the provided callback function will be one of the following constants:   |
| 29             |           | • PMIX_SUCCESS The operation was successfully completed  |
| 30<br>31       |           | • PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM does not.  |
| 32             |           | • a non-zero PMIx error constant indicating a reason for the request's failure   |
|                |           |  |

|                                 | Required Attributes  |
|---------------------------------|--|
| 1<br>2<br>3                     | PMIx libraries that choose not to support this operation <i>must</i> 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  |
| 4                               | The following attributes are optional for host environments that support this operation:   |
| 5<br>6<br>7<br>8                | 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  |
| 9<br> 0<br> 1<br> 2<br> 3<br> 4 | 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. |
| 5<br>  6<br>  7                 | <b>Description</b> Non-blocking version of the <b>PMIx_Group_destruct</b> operation. The callback function will be called once all members of the group have executed either <b>PMIx_Group_destruct</b> or <b>PMIx_Group_destruct_nb</b> .   |
| ı9 <b>13.2</b>                  | .6 PMIx_Group_invite   |
| 20<br>21                        | Summary Asynchronously construct a PMIx process group  |

| 1         |                  | Format   |
|-----------|------------------|--|
| 1         | <i>PMIx v4.0</i> |  |
| 2         |                  | pmix_status_t  |
| 3         |                  | <pre>PMIx_Group_invite(const char grp[],</pre>   |
| 4         |                  | <pre>const pmix_proc_t procs[], size_t nprocs,</pre>   |
| 5         |                  | <pre>const pmix_info_t directives[], size_t ndirs,</pre>   |
| 6         |                  | <pre>pmix_info_t **results, size_t *nresult)</pre>   |
|           |                  | 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 <i>procs</i> array (size_t)  |
| 15        |                  | IN directives  |
| 16        |                  | Array of pmix_info_t structures (array of handles)  IN ndirs   |
| 17<br>18  |                  | Number of elements in the <i>directives</i> array (size_t)   |
| 19        |                  | INOUT results  |
| 20        |                  | Pointer to a location where the array of pmix_info_t describing the results of the                   |
| - 0<br>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  |
|           |                  | Trequired Attributes   |
| 31        |                  | The following attributes are <i>required</i> 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  | Requests that the RM assign a new context identifier to the newly created group. The              |
| 4  | identifier is an unsigned, size_t value that the RM guarantees to be unique across the range      |
| 5  | specified in the request. Thus, the value serves as a means of identifying the group within       |
| 6  | that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.            |
| 7  | PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)   |
| 8  | Notify remaining members when another member terminates without first leaving the group.          |
| 9  | The default is false  |
|    | ▼ Optional Attributes   |
|    | The optional Attributes   |
| 10 | The following attributes are optional for host environments that support this operation:          |
| 11 | <pre>PMIX_TIMEOUT "pmix.timeout" (int)</pre>  |
| 12 | Time in seconds before the specified operation should time out ( $\theta$ indicating infinite) in |
| 13 | error. The timeout parameter can help avoid "hangs" due to programming errors that prevent        |
| 14 | the target process from ever exposing its data.   |
|    | <b>^</b>  |
|    | Advice to PMIx library implementers   |
| 15 | We recommend that implementation of the <b>PMIX_TIMEOUT</b> attribute be left to the host         |
| 16 | environment due to race condition considerations between completion of the operation versus       |
| 17 | internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT     |
| 18 | directly in the PMIx server library must take care to resolve the race condition and should avoid |
| 19 | passing PMIX_TIMEOUT to the host environment so that multiple competing timeouts are not          |
| 20 | 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 <code>PMIx\_Group\_join</code>. This will notify the inviting process that the invitation was either accepted (via the <code>PMIX\_GROUP\_INVITE\_ACCEPTED</code> event) or declined (via the <code>PMIX\_GROUP\_INVITE\_DECLINED</code> event). The <code>PMIX\_GROUP\_INVITE\_ACCEPTED</code> 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 <code>PMIx\_Group\_invite</code> or calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the PMIX\_GROUP\_INVITE\_DECLINED if the application allows invited processes to decline the invitation. This provides an opportunity for the application to either invite a replacement, declare "abort", or choose to remove the declining process from the final group. The inviting process should also register to receive PMIX\_GROUP\_INVITE\_FAILED events whenever a process fails or terminates prior to responding to the invitation. Actions taken by the inviting process in response to these events must be communicated at the end of the event handler by returning the corresponding result so that the PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level information of each other's namespaces plus any information posted via **PMIx\_Put** by the other members.

The inviting process is automatically considered the leader of the asynchronous group construction procedure and will receive all failure or termination events for invited members prior to completion. The inviting process is required to provide a **PMIX\_GROUP\_CONSTRUCT\_COMPLETE** event once the group has been fully assembled – this event is used by the PMIx library as a trigger to release participants from their call to **PMIx\_Group\_join** and provides information (e.g., the final group membership) to be returned in the *results* array.

### Advice to users

Applications are not allowed to use the group in any operations until group construction is complete. This is required in order to ensure consistent knowledge of group membership across all participants.

Failure of the inviting process at any time will cause a PMIX\_GROUP\_LEADER\_FAILED event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the PMIX\_GROUP\_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX\_GROUP\_LEADER\_SELECTED event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

# 13.2.7 PMIx\_Group\_invite\_nb

Returns one of the following:

1

2

4

5

6 7

8

34

#### 10 Summary Non-blocking form of PMIx\_Group\_invite 11 **Format** 12 PMIx v4.013 pmix status t PMIx\_Group\_invite\_nb(const char grp[], 14 const pmix\_proc\_t procs[], size\_t nprocs, 15 const pmix\_info\_t directives[], size\_t ndirs, 16 pmix info cbfunc t cbfunc, void \*cbdata) 17 C IN 18 grp 19 NULL-terminated character array of maximum size PMIX MAX NSLEN containing the group identifier (string) 20 IN procs 21 Array of pmix proc t structures containing the PMIx identifiers of the processes to be 22 invited (array of handles) 23 24 IN nprocs Number of elements in the *procs* array (size\_t) 25 26 IN directives 27 Array of pmix\_info\_t structures (array of handles) IN ndirs 28 29 Number of elements in the *directives* array (size\_t) IN cbfunc 30 31 Callback function pmix\_info\_cbfunc\_t (function reference) 32 IN cbdata 33 Data to be passed to the callback function (memory reference)

| 1<br>2<br>3                | • PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API.  |
|----------------------------|---|
| 4<br>5                     | • <b>PMIX_OPERATION_SUCCEEDED</b> , indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called   |
| 6<br>7                     | • PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called   |
| 8<br>9                     | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called  |
| 10<br>11                   | If executed, the status returned in the provided callback function will be one of the following constants:  |
| 12                         | • PMIX_SUCCESS The operation succeeded and all specified members participated.  |
| 13<br>14                   | • PMIX_ERR_PARTIAL_SUCCESS The operation succeeded but not all specified members participated - the final group membership is included in the callback function   |
| 15<br>16                   | • PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM does not.   |
| 17                         | • a non-zero PMIx error constant indicating a reason for the request's failure  |
|                            | ▼ Required Attributes   |
| 18<br>19                   | The following attributes are <i>required</i> to be supported by all PMIx libraries that support this operation:   |
| 20<br>21<br>22             | PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)  Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false   |
| 23                         | Host environments that support this operation are <i>required</i> to provide the following attributes:  |
| 24<br>25<br>26<br>27<br>28 | PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)  Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION. |
| 29<br>30<br>31             | PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)  Notify remaining members when another member terminates without first leaving the group.  The default is false   |

|   |        | The strict of th |  |  |  |  |
|---|--------|--|--|--|--|--|
| 1   |        | The following attributes are optional for host environments that support this operation:   |  |  |  |  |
| 2   |        | PMIX_TIMEOUT "pmix.timeout" (int)  |  |  |  |  |
| 3   |        | Time in seconds before the specified operation should time out (0 indicating infinite) in  |  |  |  |  |
| 4   |        | error. The timeout parameter can help avoid "hangs" due to programming errors that pro   |  |  |  |  |
| 5   |        |  |  |  |  |  |
|   |        | <b>^</b>   |  |  |  |  |
|   |        | Advice to PMIx library implementers  |  |  |  |  |
| 6   |        | We recommend that implementation of the <b>PMIX_TIMEOUT</b> attribute be left to the host  |  |  |  |  |
| 7   |        | environment due to race condition considerations between completion of the operation versus  |  |  |  |  |
| 8   |        | internal timeout in the PMIx server library. Implementers that choose to support PMIX_TIMEOUT  |  |  |  |  |
| 9   |        | directly in the PMIx server library must take care to resolve the race condition and should avoid  |  |  |  |  |
| 10  |        |  |  |  |  |  |
| 11  |        | created.   |  |  |  |  |
|   |        | Description  |  |  |  |  |
|   |        | <b>Description</b>   |  |  |  |  |
| Non-blocking version of the <b>PMIx_Group_invite</b> operation. The callback func |        |  |  |  |  |  |
| 14<br>15  |        | called once all invited members of the group (or their substitutes) have executed either   |  |  |  |  |
| 13  |        | PMIx_Group_join or PMIx_Group_join_nb.   |  |  |  |  |
| 16  | 13.2.8 | PMIx_Group_join  |  |  |  |  |
| 17  |        | Summary  |  |  |  |  |
| 18  |        | Accept an invitation to join a PMIx process group  |  |  |  |  |

| 1  | Format   |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|
| PMIx :                                   | v4.0   |  |  |  |  |  |  |
| 2  | pmix_status_t  |  |  |  |  |  |  |
| 3  | <pre>PMIx_Group_join(const char grp[],</pre>   |  |  |  |  |  |  |
| 4  | <pre>const pmix_proc_t *leader,</pre>  |  |  |  |  |  |  |
| <pre>5 pmix_group_operation_t opt,</pre> |  |  |  |  |  |  |  |
| 6  | <pre>const pmix_info_t directives[], size_t ndirs,</pre>   |  |  |  |  |  |  |
| 7  | <pre>pmix_info_t **results, size_t *nresult)</pre>   |  |  |  |  |  |  |
|  | C  |  |  |  |  |  |  |
| 8  | IN grp   |  |  |  |  |  |  |
| 9  | NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the                        |  |  |  |  |  |  |
| 10                                       | group identifier (string)  |  |  |  |  |  |  |
| 11                                       | IN leader  |  |  |  |  |  |  |
| 12                                       | Process that generated the invitation (handle)   |  |  |  |  |  |  |
| 13                                       | IN opt   |  |  |  |  |  |  |
| 14                                       | Accept or decline flag ( pmix_group_operation_t )  |  |  |  |  |  |  |
| 15                                       | IN directives  |  |  |  |  |  |  |
| 16                                       | Array of pmix_info_t structures (array of handles)   |  |  |  |  |  |  |
| 17                                       | IN ndirs   |  |  |  |  |  |  |
| 18                                       | Number of elements in the <i>directives</i> array (size_t)   |  |  |  |  |  |  |
| 19                                       | INOUT results  |  |  |  |  |  |  |
| 20                                       | Pointer to a location where the array of <b>pmix_info_t</b> describing the results of the            |  |  |  |  |  |  |
| 21<br>22                                 | operation is to be returned (pointer to handle)  INOUT nresults                                      |  |  |  |  |  |  |
| 22                                       | Pointer to a <b>size_t</b> location where the number of elements in <i>results</i> is to be returned |  |  |  |  |  |  |
| 23<br>24                                 | (memory reference)   |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 25                                       | Returns one of the following:  |  |  |  |  |  |  |
| 26                                       | <ul> <li>PMIX_SUCCESS, indicating that the request has been successfully completed</li> </ul>        |  |  |  |  |  |  |
| 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                                       | There are no identified required attributes for implementers.  |  |  |  |  |  |  |
|  | <b>A</b>   |  |  |  |  |  |  |

# 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 ( $\theta$ 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

1 2

3

4

5

6 7

8

9 10

11

12 13

14

15

16 17

18

19

20 21 Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the blocking form of this call unless it first "thread shifts" out of the handler and into its own thread context. Likewise, while it is safe to call the non-blocking form of the API from the event handler, the process *must not* block in the handler while waiting for the callback function to be called.

Calling this function causes the inviting process (aka the *group leader*) to be notified that the process has either accepted or declined the request. The blocking form of the API will return once the group has been completely constructed or the group's construction has failed (as described below) – likewise, the callback function of the non-blocking form will be executed upon the same conditions.

Failure of the leader during the call to PMIx\_Group\_join will cause a

PMIX\_GROUP\_LEADER\_FAILED event to be delivered to all invited participants so they can optionally declare a new leader. A new leader is identified by providing the

PMIX\_GROUP\_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX\_GROUP\_LEADER\_SELECTED event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

Any participant that returns PMIX\_GROUP\_CONSTRUCT\_ABORT from the leader failed event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning PMIX\_GROUP\_CONSTRUCT\_ABORT from the handler assigned to the PMIX\_GROUP\_INVITE\_ACCEPTED or PMIX\_GROUP\_INVITE\_DECLINED codes, or by generating an event for the abort code. Abort events will be sent to all invited participants.

# 13.2.9 PMIx\_Group\_join\_nb

pmix\_status\_t

### Summary

Non-blocking form of PMIx Group join

### Format

PMIx v4.0

IN grp

**NULL**-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group identifier (string)

IN leader

Process that generated the invitation (handle)

| 1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9 | IN opt Accept or decline flag (pmix_group_operation_t)  IN directives Array of pmix_info_t structures (array of handles)  IN ndirs Number of elements in the directives array (size_t)  IN cbfunc Callback function pmix_info_cbfunc_t (function reference)  IN cbdata Data to be passed to the callback function (memory reference) |
|---|--|
| 11  | Returns one of the following:  |
| 12<br>13<br>14                            | • PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided <i>cbfunc</i> . Note that the library <i>must not</i> invoke the callback function prior to returning from the API.   |
| 15<br>16                                  | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called  |
| 17<br>18                                  | • PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called  |
| 19<br>20                                  | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called   |
| 21<br>22                                  | If executed, the status returned in the provided callback function will be one of the following constants:   |
| 23<br>24                                  | • PMIX_SUCCESS The operation succeeded and group membership is in the callback function parameters   |
| 25<br>26                                  | • PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM 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<br>31<br>32<br>33                      | 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 1 2 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 3 directly in the PMIx server library must take care to resolve the race condition and should avoid 4 passing PMIX TIMEOUT to the host environment so that multiple competing timeouts are not 5 created. 6 **Description** 7 8 Non-blocking version of the PMIx\_Group\_join operation. The callback function will be called 9 once all invited members of the group (or their substitutes) have executed either PMIx\_Group\_join or PMIx\_Group\_join\_nb. 10 13.2.10 PMIx Group leave 11 12 Summary 13 Leave a PMIx process group Format 14 PMIx v4.0 15 pmix status t PMIx Group leave(const char grp[], 16 const pmix\_info\_t directives[], size\_t ndirs) 17 18 IN grp NULL-terminated character array of maximum size PMIX MAX NSLEN containing the 19 group identifier (string) 20 IN directives 21 22 Array of pmix info t structures (array of handles) 23 IN Number of elements in the *directives* array (size t) 24 25 Returns one of the following: 26 • 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 27 operation 28 29 • 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. 30

### Description

Leave a PMIx Group. Calls to PMIx\_Group\_leave (or its non-blocking form) will cause a PMIX\_GROUP\_LEFT event to be generated notifying all members of the group of the caller's departure. The function will return (or the non-blocking function will execute the specified callback function) once the event has been locally generated and is not indicative of remote receipt.

### Advice to users

The PMIx\_Group\_leave API is intended solely for asynchronous departures of individual processes from a group as it is not a scalable operation – i.e., when a process determines it should no longer be a part of a defined group, but the remainder of the group retains a valid reason to continue in existence. Developers are advised to use PMIx\_Group\_destruct (or its non-blocking form) for all other scenarios as it represents a more scalable operation.

# 13.2.11 PMIx\_Group\_leave\_nb

### Summary

Non-blocking form of PMIx\_Group\_leave

#### Format

PMIx v4.0

1

3

5

6 7

8

9

10

12

13

14

15

16

17

18

19

20

21

22

23 24

25

26

27 28

29

30

31

32

33

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)

IN grp

**NULL**-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the group identifier (string)

C -

IN directives

Array of **pmix\_info\_t** structures (array of handles)

IN ndirs

Number of elements in the *directives* array (size t)

IN cbfunc

Callback function **pmix\_op\_cbfunc\_t** (function reference)

IN cbdata

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

Returns one of the following:

• PMIX\_SUCCESS, indicating that the request is being processed - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.

| 1<br>2 | • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned <i>success</i> - the <i>cbfunc</i> will <i>not</i> be called            |  |  |  |
|--------|--|--|--|--|
| 3<br>4 | • PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the <i>cbfunc</i> will <i>not</i> be called  |  |  |  |
| 5<br>6 | • a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the <i>cbfunc</i> will <i>not</i> be called |  |  |  |
| 7<br>8 | If executed, the status returned in the provided callback function will be one of the following constants:   |  |  |  |
| 9<br>0 | <ul> <li>PMIX_SUCCESS The operation succeeded - i.e., the PMIX_GROUP_LEFT event was<br/>generated</li> </ul>   |  |  |  |
| 1<br>2 | • PMIX_ERR_NOT_SUPPORTED While the PMIx library supports this operation, the host RM does not.   |  |  |  |
| 3      | • a non-zero PMIx error constant indicating a reason for the request's failure   |  |  |  |
|        | ▼ Required Attributes  |  |  |  |
| 4      | There are no identified required attributes for implementers.  |  |  |  |
| 5      | Description  |  |  |  |
| 6      | Non-blocking version of the <b>PMIx_Group_leave</b> operation. The callback function will be   |  |  |  |
| 7      | called once the event has been locally generated and is not indicative of remote receipt.  |  |  |  |

### **APPENDIX A**

# **Python Bindings**

While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of PMIx to that specific language. Support for other languages is captured in the Standard by describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes. Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.

Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx "class" (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and therefore those have been included in the "client" class. Servers can utilize any of those functions plus a set focused on operations not commonly executed by an application process. Finally, tools can also act as servers but have their own initialization function.

# A.1 Design Considerations

12 Several issues arose during design of the Python bindings:

# A.1.1 Error Codes vs Python Exceptions

The C programming language reports errors through the return of the corresponding integer status codes. PMIx has defined a range of negative values for this purpose. However, Python has the option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and return PMIx status codes in lieu of raising exceptions as this method was considered more consistent for those working in both domains.

# A.1.2 Representation of Structured Data

PMIx utilizes a number of C-language structures to efficiently bundle related information. For example, the PMIx process identifier is represented as a struct containing a character array for the namespace and a 32-bit unsigned integer for the process rank. There are several options for translating such objects to Python – e.g., the PMIx process identifier could be represented as a two-element tuple (nspace, rank) or as a dictionary 'nspace': name, 'rank': 0. Exploration found no discernible benefit to either representation, nor was any clearly identifiable rationale developed that would lead a user to expect one versus the other for a given PMIx data type. Consistency in the translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion. Hence, the decision was made to express all complex datatypes as Python dictionaries.

# A.2 Datatype Definitions

 PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range integers (e.g., uint32\_t), and structures. Each datatype is represented by a named unsigned 16-bit integer (uint16\_t) constant. Users are advised to use the named PMIx constants for indicating datatypes instead of integer values to ensure compatibility with future PMIx versions.

With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 20 directly translate to Python. However, Python lacks the size-specific value definitions of C (e.g., uint8\_t) and thus some care must be taken to protect against overflow/underflow situations when moving between the languages. Python bindings that accept values including PMIx datatypes shall therefore have the datatype and associated value checked for compatibility with their PMIx-defined equivalents, returning an error if:

- datatypes not defined by PMIx are encountered
- provided values fall outside the range of the C-equivalent definition e.g., if a value identified as **PMIX UINT8** lies outside the **uint8** trange

Note that explicit labeling of PMIx datatype, even when Python itself doesn't care, is often required for the Python bindings to know how to properly interpret and label the provided value when passing it to the PMIx library.

Table A.1 lists the correspondence between datatypes in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                     | PMIx Name               | Python Definition         | Notes  |
|----------------------------------|-------------------------|---------------------------|--|
| bool                             | PMIX_BOOL               | boolean                   |  |
| byte                             | PMIX_BYTE               | A single element byte     |  |
|                                  |                         | array (i.e., a byte array |  |
|                                  |                         | of length one)            |  |
| char*                            | PMIX_STRING             | string                    |  |
| size_t                           | PMIX_SIZE               | integer                   |  |
| pid_t                            | PMIX_PID                | integer                   | value shall be limited to the uint32_t   |
|                                  |                         |                           | range  |
| <pre>int, int8_t, int16_t,</pre> | PMIX_INT, PMIX_INT8,    | integer                   | value shall be limited to its corresponding  |
| int32_t, int64_t                 | PMIX_INT16, PMIX_INT32, |                           | range  |
|                                  | PMIX_INT64              |                           |  |
| uint, uint8_t,                   | PMIX_UINT, PMIX_UINT8,  | integer                   | value shall be limited to its corresponding  |
| uint16_t, uint32_t,              | PMIX_UINT16,            |                           | range  |
| uint64_t                         | PMIX_UINT32,            |                           |  |
|                                  | PMIX_UINT64             |                           |  |
| float, double                    | PMIX_FLOAT,             | float                     | value shall be limited to its corresponding  |
|                                  | PMIX_DOUBLE             |                           | range  |
| struct timeval                   | PMIX_TIMEVAL            | {'sec': sec, 'usec':      | each field is an integer value   |
|                                  |                         | microsec}                 |  |
| time_t                           | PMIX_TIME               | integer                   | limited to positive values   |
| pmix_data_type_t                 | PMIX_DATA_TYPE          | integer                   | value shall be limited to the uint16_t range   |
| pmix_status_t                    | PMIX_STATUS             | integer                   |  |
| pmix_key_t                       | N/A                     | string                    | The string's length shall be limited to one less than the size of the <code>pmix_key_t</code> array (to reserve space for the terminating <code>NULL</code> )    |
| pmix_nspace_t                    | N/A                     | string                    | The string's length shall be limited to one less than the size of the <code>pmix_nspace_t</code> array (to reserve space for the terminating <code>NULL</code> ) |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition       | PMIx Name        | Python Definition  | Notes   |
|--------------------|------------------|--|---|
| pmix_rank_t        | PMIX_PROC_RANK   | integer  | value shall be limited to the uint32_t range excepting the reserved values near UINT32_MAX  |
| pmix_proc_t        | PMIX_PROC        | {'nspace': nspace, 'rank': rank}   | nspace is a Python string and rank is an integer value. The nspace string's length shall be limited to one less than the size of the pmix_nspace_t array (to reserve space for the terminating NULL), and the rank value shall conform to the constraints associated with pmix_rank_t |
| pmix_byte_object_t | PMIX_BYTE_OBJECT | {'bytes': bytes, 'size': size}   | bytes is a Python byte array and size is the integer number of bytes in that array.   |
| pmix_persistence_t | PMIX_PERSISTENCE | integer  | value shall be limited to the uint8_t range   |
| pmix_scope_t       | PMIX_SCOPE       | integer  | value shall be limited to the uint8_t range   |
| pmix_data_range_t  | PMIX_RANGE       | integer  | value shall be limited to the uint8_t range   |
| pmix_proc_state_t  | PMIX_PROC_STATE  | integer  | value shall be limited to the uint8_t range   |
| pmix_proc_info_t   | PMIX_PROC_INFO   | {'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state} | proc is a Python proc dictionary; hostname and executable are Python strings; and pid, exitcode, and state are Python integers  |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                      | PMIx Name            | Python Definition   | Notes   |
|-----------------------------------|----------------------|---|---|
| pmix_data_array_t                 | PMIX_DATA_ARRAY      | {'type': type, 'array':<br>array}   | type is the PMIx type of object in the array and array is a Python list containing the individual array elements. Note that array can consist of any PMIx types, including (for example) a Python info object that itself contains an array value |
| <pre>pmix_info_directives_t</pre> | PMIX_INFO_DIRECTIVES | integer   | value shall be limited to the uint32_t range  |
| pmix_alloc_directive_t            | PMIX_ALLOC_DIRECTIVE | integer   | value shall be limited to the uint8_t range   |
| pmix_iof_channel_t                | PMIX_IOF_CHANNEL     | integer   | value shall be limited to the uint16_t range  |
| pmix_envar_t                      | PMIX_ENVAR           | {'envar': envar, 'value': value, 'separator': separator}  | envar and value are Python strings, and separator a single-character Python string  |
| pmix_value_t                      | PMIX_VALUE           | {'value': value, 'val_type': type}  | type is the PMIx datatype of value, and value is the associated value expressed in the appropriate Python form for the specified datatype   |
| pmix_info_t                       | PMIX_INFO            | {'key': key, 'flags':<br>flags, value': value,<br>'val_type': type}                               | key is a Python string key, flags is a bitmask of info directives, type is the PMIx datatype of value, and value is the associated value expressed in the appropriate Python form for the specified datatype                                      |
| pmix_pdata_t                      | PMIX_PDATA           | {'proc': {'nspace':<br>nspace, 'rank': rank},<br>'key': key, 'value':<br>value, 'val_type': type} | proc is a Python proc dictionary;<br>key is a Python string key; type is the<br>PMIx datatype of value; and value is<br>the associated value expressed in the<br>appropriate Python form for the specified<br>datatype                            |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition      | PMIx Name       | Python Definition          | Notes  |
|-------------------|-----------------|----------------------------|--|
| pmix_app_t        | PMIX_APP        | {'cmd': cmd, 'argv':       | cmd is a Python string; argv and env are                   |
|                   |                 | [argv], 'env': [env],      | Python <i>lists</i> containing Python strings;             |
|                   |                 | 'maxprocs': maxprocs,      | maxprocs is an integer; and info is a                      |
|                   |                 | 'info': [info]}            | Python list of info values                                 |
| pmix_query_t      | PMIX_QUERY      | {'keys': [keys],           | keys is a Python list of Python strings, and               |
|                   |                 | 'qualifiers': [info]}      | qualifiers is a Python list of info values                 |
| pmix_regattr_t    | PMIX_REGATTR    | {'name': name, 'key':      | name and string are Python strings; type is                |
|                   |                 | key, 'type': type, 'info': | the PMIx datatype for the attribute's value;               |
|                   |                 | [info], 'description':     | <i>info</i> is a Python <i>list</i> of <b>info</b> values; |
|                   |                 | [desc]}                    | and description is a list of Python strings                |
|                   |                 |                            | describing the attribute                                   |
| pmix_link_state_t | PMIX_LINK_STATE | integer                    | value shall be limited to the uint8_t                      |
|                   |                 |                            | range  |

# A.2.1 Example

datatypes as well as use of the appropriate API form. An example small program may help illustrate the changes. Consider the following C-based program snippet: #include <pmix.h> . . . pmix info t info[2]; PMIX\_INFO\_LOAD(&info[0], PMIX\_PROGRAMMING\_MODEL, "TEST", PMIX\_STRING) PMIX\_INFO\_LOAD(&info[1], PMIX\_MODEL\_LIBRARY\_NAME, "PMIX", PMIX\_STRING) rc = PMIx\_Init(&myproc, info, 2); PMIX\_INFO\_DESTRUCT(&info[0]); // free the copied string PMIX\_INFO\_DESTRUCT(&info[1]); // free the copied string 

Converting a C-based program to its Python equivalent requires translation of the relevant

Moving to the Python version requires that the **pmix\_info\_t** be translated to the Python **info** equivalent, and that the returned information be captured in the return parameters as opposed to a pointer parameter in the function call, as shown below:

Note the use of the **PMIX\_STRING** identifier to ensure the Python bindings interpret the provided string value as a PMIx "string" and not an array of bytes.

# A.3 Callback Function Definitions

# A.3.1 IOF Delivery Function

## Summary

Callback function for delivering forwarded IO to a process

```
Format
1
                                                     Python –
   PMIx v4.0
2
              def iofcbfunc(iofhdlr:integer, channel:integer,
                                source:dict, payload:dict, info:list)
 3
                                                     Python ——
4
              IN
                  iofhdlr
                   Registration number of the handler being invoked (integer)
 5
6
              IN channel
7
                   Python channel bitmask identifying the channel the data arrived on (integer)
8
              IN
                   source
9
                   Python proc identifying the namespace/rank of the process that generated the data (dict)
              IN payload
10
                   Python byteobject containing the data (dict)
11
              IN
                  info
12
13
                   List of Python info provided by the source containing metadata about the payload. This
                   could include PMIX IOF COMPLETE (list)
14
              Returns: nothing
15
16
              See pmix iof cbfunc t for details
   A.3.2 Event Handler
18
              Summary
              Callback function for event handlers
19
              Format
20
                                                     Pvthon
   PMIx v4.0
21
              def evhandler(evhdlr:integer, status:integer,
                                source:dict, info:list, results:list)
22
                                                     Python -
              IN
23
                  iofhdlr
24
                   Registration number of the handler being invoked (integer)
25
              IN
26
                   Status associated with the operation (integer)
              IN source
27
                   Python proc identifying the namespace/rank of the process that generated the event (dict)
28
29
              IN info
                   List of Python info provided by the source containing metadata about the event (list)
30
31
              IN
                   results
32
                   List of Python info containing the aggregated results of all prior evhandlers (list)
33
              Returns:
```

1 • rc - Status returned by the event handler's operation (integer) • results - List of Python info containing results from this event handler's operation on the event 2 3 (list) See pmix\_notification\_fn\_t for details 4 A.3.3 **Server Module Functions** The following definitions represent functions that may be provided to the PMIx server library at 6 7 time of initialization for servicing of client requests. Module functions that are not provided default to returning "not supported" to the caller. 8 9 A.3.3.1 **Client Connected** 10 Summary 11 Notify the host server that a client connected to this server. **Format** 12 Python PMIx v4.013 def clientconnected(proc:dict is not None) Python 14 IN proc Python **proc** identifying the namespace/rank of the process that connected (dict) 15 16 Returns: 17 • rc - PMIX\_SUCCESS or a PMIx error code indicating the connection should be rejected 18 (integer) 19 See pmix\_server\_client\_connected\_fn\_t for details A.3.3.2 Client Finalized 20 21 Summary 22 Notify the host environment that a client called **PMIx\_Finalize**. Format 23 Python PMIx v4.0 def clientfinalized(proc:dict is not None): 24 **Python** 25 IN proc Python proc identifying the namespace/rank of the process that finalized (dict) 26 27 Returns: nothing 28 See pmix server client finalized fn t for details

| 1 <b>A.3.3</b>       | 3 Client Aborted  |
|----------------------|---|
| 2 3                  | <b>Summary</b> Notify the host environment that a local client called <b>PMIx_Abort</b> .                                 |
| 4 PMIx v4            | Format Python   |
| 5                    | <pre>def clientaborted(args:dict is not None)</pre>   |
| 6<br>7               | IN args Python dictionary containing:   |
| 8                    | • 'caller': Python <b>proc</b> identifying the namespace/rank of the process calling abort (dict)                         |
| 9                    | • 'status': PMIx status to be returned on exit (integer)  |
| 10                   | • 'msg': Optional string message to be printed (string)   |
| 11<br>12             | • 'targets': Optional list of Python <b>proc</b> identifying the namespace/rank of the processes to be aborted (list)     |
| 13                   | Returns:  |
| 14                   | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  |
| 15                   | See pmix_server_abort_fn_t for details  |
| 16 <b>A.3.3</b>      | 4 Fence   |
| 17<br>18             | Summary At least one client called either PMIx_Fence or PMIx_Fence_nb   |
| 19<br><i>PMIx v4</i> | Format Python   |
| 20                   | <pre>def fence(args:dict is not None)</pre>   |
| 21<br>22             | IN args Python dictionary containing:   |
| 23<br>24             | <ul> <li>'procs': List of Python proc identifying the namespace/rank of the participating<br/>processes (list)</li> </ul> |
| 25<br>26             | • 'directives': Optional list of Python <b>info</b> containing directives controlling the operation (list)                |
| 27                   | • 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)                           |
| 28                   | Returns:  |

| 1           |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  |
|-------------|-----------|---|
| 2           |           | • data - Python bytearray containing the aggregated data from all participants (bytearray)  |
| 3<br>4      | A.3.3.5   | See pmix_server_fencenb_fn_t for details Direct Modex   |
| 5<br>6<br>7 |           | <b>Summary</b> Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc. |
| 8           |           | Format Python   |
| ^           | PMIx v4.0 |   |
| 9           |           | <pre>def dmodex(args:dict is not None)</pre>  |
| 10<br>11    |           | IN args Python dictionary containing:   |
| 12          |           | • 'proc': Python <b>proc</b> of process whose data is being requested (dict)  |
| 13<br>14    |           | <ul> <li>'directives': Optional list of Python info containing directives controlling the operation<br/>(list)</li> </ul>   |
| 15          |           | Returns:  |
| 16          |           | • <i>rc</i> - <b>PMIX_SUCCESS</b> or a PMIx error code indicating the operation failed (integer)  |
| 17          |           | • data - Python bytearray containing the data for the specified process (bytearray)   |
| 18<br>19    | A.3.3.6   | See pmix_server_dmodex_req_fn_t for details Publish   |
| 20<br>21    |           | Summary Publish data per the PMIx API specification.  |
| 22          | PMIx v4.0 | Format Python —   |
| 23          | Time vi.o | def publish(args:dict is not None)  Python  |
| 24<br>25    |           | IN args Python dictionary containing:   |
| 26          |           | • 'proc': Python proc dictionary of process publishing the data (dict)  |
| 27          |           | • 'directives': List of Python info containing data and directives (list)   |
| 28          |           | Returns:  |
| 29          |           | • <i>rc</i> - <b>PMIX_SUCCESS</b> or a PMIx error code indicating the operation failed (integer)  |
| 30          |           | See pmix_server_publish_fn_t for details  |

```
A.3.3.7 Lookup
 1
 2
               Summary
 3
               Lookup published data.
               Format
                                                        Python
   PMIx v4.0
 5
               def lookup(args:dict is not None)
                                                        Python
               IN
 6
                    args
 7
                    Python dictionary containing:
                    • 'proc': Python proc of process seeking the data (dict)
 8
 9
                    • 'keys': List of Python strings (list)
10
                    • 'directives': Optional list of Python info containing directives (list)
               Returns:
11
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
12
               • pdata - List of pdata containing the returned results (list)
13
14
               See pmix server lookup fn t for details
    A.3.3.8
               Unpublish
15
               Summary
16
               Delete data from the data store.
17
               Format
18
                                                        Python
   PMIx v4.0
19
               def unpublish(args:dict is not None)
                                                        Python
               IN
20
                   args
                    Python dictionary containing:
21
                    • 'proc': Python proc of process unpublishing data (dict)
22
23
                    • 'keys': List of Python strings (list)
24
                    • 'directives': Optional list of Python info containing directives (list)
               Returns:
25
26
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
27
               See pmix server unpublish fn t for details
```

| 1        | A.3.3.9   | Spawn   |
|----------|-----------|---|
| 2        |           | Summary Spawn a set of applications/processes as per the PMIx_Spawn API.  |
| 4        |           | Format  |
| •        | PMIx v4.0 | Python —  |
| 5        |           | def spawn(args:dict is not None)  |
|          |           | Python —  |
| 6<br>7   |           | IN args Python dictionary containing:   |
| 8        |           | • 'proc': Python <b>proc</b> of process making the request (dict)   |
| 9        |           | • 'jobinfo': Optional list of Python info job-level directives and information (list)                                     |
| 0        |           | • 'apps': List of Python app describing applications to be spawned (list)   |
| 1        |           | Returns:  |
| 2        |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  |
| 3        |           | • nspace - Python string containing namespace of the spawned job (str)  |
| 4        |           | See pmix_server_spawn_fn_t for details  |
| 5        | A.3.3.10  | Connect   |
| 6<br>7   |           | <b>Summary</b> Record the specified processes as <i>connected</i> .   |
| 8        |           | Format  |
|          | PMIx v4.0 | Python —  |
| 9        |           | def connect(args:dict is not None)  Python  |
| 20<br>21 |           | IN args Python dictionary containing:   |
| 22       |           | <ul> <li>'procs': List of Python proc identifying the namespace/rank of the participating<br/>processes (list)</li> </ul> |
| 24<br>25 |           | • 'directives': Optional list of Python <b>info</b> containing directives controlling the operation (list)                |
| 26       |           | Returns:  |
| 27       |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  |
| 28       |           | See pmix_server_connect_fn_t for details  |

#### A.3.3.11 Disconnect 1 2 Summary 3 Disconnect a previously connected set of processes. **Format** Python PMIx v4.0 5 def disconnect(args:dict is not None) Python IN 6 args Python dictionary containing: 7 • 'procs': List of Python proc identifying the namespace/rank of the participating 8 9 processes (list) • 'directives': Optional list of Python **info** containing directives controlling the operation 10 11 (list) 12 Returns: 13 • rc - PMIX\_SUCCESS or a PMIx error code indicating the operation failed (integer) 14 See pmix server disconnect fn t for details A.3.3.12 Register Events 15 Summary 16 17 Register to receive notifications for the specified events. **Format** 18 Python PMIx v4.0 def register\_events(args:dict is not None) 19 Python IN 20 args 21 Python dictionary containing: 22 • 'codes': List of Python integers (list) • 'directives': Optional list of Python **info** containing directives controlling the operation 23 (list) 24 25 Returns: 26 • rc - PMIX\_SUCCESS or a PMIx error code indicating the operation failed (integer) See pmix\_server\_register\_events\_fn\_t for details 27

### A.3.3.13 Deregister Events 1 2 Summary Deregister to receive notifications for the specified events. 3 Format Python -PMIx v4.0 5 def deregister\_events(args:dict is not None) Python IN 6 args 7 Python dictionary containing: • 'codes': List of Python integers (list) Returns: 9 10 • rc - PMIX\_SUCCESS or a PMIx error code indicating the operation failed (integer) See pmix server deregister events fn t for details 11 A.3.3.14 Notify Event 12 Summary 13 14 Notify the specified range of processes of an event. 15 Format Python PMIx v4.0def notify\_event(args:dict is not None) 16 Python 17 IN args Python dictionary containing: 18 19 • 'code': Python integer pmix\_status\_t (integer) 20 • 'source': Python proc of process that generated the event (dict) • 'range': Python range in which the event is to be reported (integer) 21 22 • 'directives': Optional list of Python info directives (list) 23 Returns: 24 • rc - PMIX\_SUCCESS or a PMIx error code indicating the operation failed (integer) 25 See pmix\_server\_notify\_event\_fn\_t for details A.3.3.15 Query 26 Summary 27 28 Query information from the resource manager.

```
Format
 1
                                                       Python
   PMIx v4.0
 2
               def query(args:dict is not None)
                                                       Python
               IN
 3
                    args
                    Python dictionary containing:
                    • 'source': Python proc of requesting process (dict)
 5
                    • 'queries': List of Python query directives (list)
 6
               Returns:
 7
 8
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
 9
               • info - List of Python info containing the returned results (list)
               See pmix_server_query_fn_t for details
10
    A.3.3.16 Tool Connected
11
12
               Summary
13
               Register that a tool has connected to the server.
               Format
14
                                                       Python
   PMIx v4.0
15
               def tool connected(args:dict is not None)
                                                       Python
16
               IN
                    args
                    Python dictionary containing:
17
18
                    • 'directives': Optional list of Python info info on the connecting tool (list)
               Returns:
19
20
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
               • proc - Python proc containing the assigned namespace:rank for the tool (dict)
21
22
               See pmix server tool connection fn t for details
    A.3.3.17 Log
23
               Summary
24
25
               Log data on behalf of a client.
```

| 1        |           | Format  |
|----------|-----------|---|
| _        | PMIx v4.0 | Python —  |
| 2        |           | <pre>def log(args:dict is not None)</pre>   |
| 3        |           | IN args   |
| 4        |           | Python dictionary containing:   |
| 5        |           | • 'source': Python proc of requesting process (dict)                                |
| 6        |           | • 'data': Optional list of Python info containing data to be logged (list)          |
| 7        |           | • 'directives': Optional list of Python info containing directives (list)           |
| 8        |           | Returns:  |
| 9        |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  |
| 10       |           | See pmix_server_log_fn_t for details  |
| 11       | A.3.3.18  | Allocate Resources  |
| 12<br>13 |           | <b>Summary</b> Request allocation operations on behalf of a client.                 |
| 14       |           | Format Python   |
| 15       | PMIx v4.0 | def allocate(args:dict is not None) Python  |
| 16<br>17 |           | IN args Python dictionary containing:   |
| 18       |           | • 'source': Python proc of requesting process (dict)                                |
| 19       |           | • 'action': Python allocdir specifying requested action (integer)                   |
| 20       |           | • 'directives': Optional list of Python info containing directives (list)           |
| 21       |           | Returns:  |
| 22       |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  |
| 23       |           | • refarginfo - List of Python info containing results of requested operation (list) |
| 24       |           | See <pre>pmix_server_alloc_fn_t</pre> for details                                   |
| 25       | A.3.3.19  | Job Control   |
| 26<br>27 |           | Summary Execute a job control action on behalf of a client.                         |

```
Format
 1
                                                         Python
   PMIx v4.0
 2
               def job control(args:dict is not None)
                                                         Python
               IN
 3
                     args
                    Python dictionary containing:
                    • 'source': Python proc of requesting process (dict)
 5
                    • 'targets': List of Python proc specifying target processes (list)
 6
                    • 'directives': Optional list of Python info containing directives (list)
 8
               Returns:
 9
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
10
               See pmix_server_job_control_fn_t for details
    A.3.3.20 Monitor
11
               Summary
12
13
               Request that a client be monitored for activity.
               Format
14
                                                         Python
   PMIx v4.0
15
               def monitor(args:dict is not None)
                                                         Python
16
               IN
                     args
                    Python dictionary containing:
17
18
                     • 'source': Python proc of requesting process (dict)
19
                     • 'monitor': Python info attribute indicating the type of monitor being requested (dict)
                    • 'error': Status code to be used when generating an event notification (integer) alerting that
20
21
                       the monitor has been triggered.
22
                    • 'directives': Optional list of Python info containing directives (list)
23
               Returns:
24
               • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
25
               See pmix server monitor fn t for details
    A.3.3.21 Get Credential
26
27
               Summary
28
               Request a credential from the host environment
```

| 1  | PMIx v4.0           | Format Python  |
|--|---------------------|--|
| 2  | PMIX V4.0           | def get_credential(args:dict is not None)  |
|  |                     | Python —   |
| 3  |                     | IN args  |
| 4  |                     | Python dictionary containing:  |
| 5  |                     | • 'source': Python proc of requesting process (dict)   |
| 6  |                     | • 'directives': Optional list of Python info containing directives (list)  |
| 7  |                     | Returns:   |
| 8  |                     | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)   |
| 9  |                     | • cred - Python byteobject containing returned credential (dict)   |
| 10   |                     | • <i>info</i> - List of Python <b>info</b> containing any additional info about the credential (list)  |
| 11   |                     | See pmix_server_get_cred_fn_t for details  |
| 12   | A.3.3.22            | Validate Credential  |
| 13<br>14   |                     | Summary Request validation of a credential   |
|  |                     |  |
| 15   | PMIx v4.0           | Format Python —  |
| 15<br>16   | PMIx v4.0           |  |
|  | PMIx v4.0           | Python  def validate_credential(args:dict is not None)   |
| 16<br>17   | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args  |
| 16<br>17<br>18                                     | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  |
| 16<br>17<br>18<br>19                               | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  • 'source': Python proc of requesting process (dict)  |
| 16<br>17<br>18<br>19<br>20                         | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  • 'source': Python proc of requesting process (dict)  • 'credential': Python byteobject containing credential (dict)  |
| 16<br>17<br>18<br>19<br>20<br>21                   | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  • 'source': Python proc of requesting process (dict)  • 'credential': Python byteobject containing credential (dict)  • 'directives': Optional list of Python info containing directives (list)   |
| 16<br>17<br>18<br>19<br>20<br>21<br>22             | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  • 'source': Python proc of requesting process (dict)  • 'credential': Python byteobject containing credential (dict)  • 'directives': Optional list of Python info containing directives (list)  Returns:   |
| 16<br>17<br>18<br>19<br>20<br>21<br>22<br>23       | PMIx v4.0           | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  • 'source': Python proc of requesting process (dict)  • 'credential': Python byteobject containing credential (dict)  • 'directives': Optional list of Python info containing directives (list)  Returns:  • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)   |
| 16<br>17<br>18<br>19<br>20<br>21<br>22<br>23<br>24 | PMIx v4.0  A.3.3.23 | Python  def validate_credential(args:dict is not None)  Python  IN args Python dictionary containing:  • 'source': Python proc of requesting process (dict)  • 'credential': Python byteobject containing credential (dict)  • 'directives': Optional list of Python info containing directives (list)  Returns:  • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  • info - List of Python info containing any additional info from the credential (list)  See pmix_server_validate_cred_fn_t for details |

```
Format
 1
                                                        Python
   PMIx v4.0
 2
               def iof pull(args:dict is not None)
                                                         Python
               IN
 3
                     args
                    Python dictionary containing:
                    • 'sources': List of Python proc of processes whose IO is being requested (list)
 5
                    • 'channels': Bitmask of Python channel identifying IO channels to be forwarded (integer)
 6
                    • 'directives': Optional list of Python info containing directives (list)
 7
 8
               Returns:
 9
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
               See pmix server iof fn t for details
10
    A.3.3.24 IO Push
11
               Summary
12
               Pass standard input data to the host environment for transmission to specified recipients.
13
               Format
14
                                                         Python
   PMIx v4.0
15
               def iof push(args:dict is not None)
                                                         Python
16
               IN
                     args
                    Python dictionary containing:
17
18
                     • 'source': Python proc of process whose input is being forwarded (dict)
                    • 'payload': Python byteobject containing input bytes (dict)
19
                    • 'targets': List of proc of processes that are to receive the payload (list)
20
                    • 'directives': Optional list of Python info containing directives (list)
21
22
               Returns:
23
               • rc - PMIX SUCCESS or a PMIx error code indicating the operation failed (integer)
24
               See pmix_server_stdin_fn_t for details
    A.3.3.25 Group Operations
25
               Summary
26
27
               Request group operations (construct, destruct, etc.) on behalf of a set of processes.
```

| 1        |           | Format Python  |
|----------|-----------|--|
| _        | PMIx v4.0 |  |
| 2        |           | def group(args:dict is not None)  Python   |
| 3        |           | IN args  |
| 4        |           | Python dictionary containing:  |
| 5        |           | • 'op': Operation host is to perform on the specified group (integer)  |
| 6        |           | • 'group': String identifier of target group (str)   |
| 7        |           | • 'procs': List of Python <b>proc</b> of participating processes (dict)  |
| 8        |           | • 'directives': Optional list of Python info containing directives (list)  |
| 9        |           | Returns:   |
| 10       |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)                                     |
| 11       |           | • refarginfo - List of Python info containing results of requested operation (list)                                    |
| 12       |           | See pmix_server_grp_fn_t for details   |
| 13       | A.3.3.26  | Fabric Operations  |
| 14<br>15 |           | <b>Summary</b> Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process. |
| 16       | PMIx v4.0 | Format Python  |
| 17       |           | <pre>def fabric(args:dict is not None)</pre>   |
| 18<br>19 |           | IN args Python dictionary containing:  |
| 20       |           | • 'source': Python <b>proc</b> of requesting process (dict)  |
| 21       |           | • 'op': Operation host is to perform on the specified fabric (integer)   |
| 22       |           | • 'directives': Optional list of Python info containing directives (list)  |
| 23       |           | Returns:   |
| 24       |           | • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)                                     |
| 25       |           | • refarginfo - List of Python info containing results of requested operation (list)                                    |
| 26       |           | See pmix_server_fabric_fn_t for details  |

## A.4 PMIxClient

The client Python class is by far the richest in terms of APIs as it houses all the APIs that an application might utilize. Due to the datatype translation requirements of the C-Python interface, only the blocking form of each API is supported – providing a Python callback function directly to the C interface underlying the bindings was not a supportable option.

## 6 A.4.1 Client.init

## 7 Summary

8

11

12

13

14

15

16

20

21

22

23

Initialize the PMIx client library after obtaining a new PMIxClient object

# PMIx v4.0 Python rc, proc = myclient.init(info:list) Python

#### IN info

**Format** 

List of Python **info** dictionaries (list)

Returns:

- rc PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- proc a Python proc dictionary (dict)

See PMIx\_Init for description of all relevant attributes and behaviors

# 17 A.4.2 Client.initialized

PMIx v4.0

rc = myclient.initialized()

Python

Python

Returns:

• rc - a value of 1 (true) will be returned if the PMIx library has been initialized, and 0 (false) otherwise (integer)

See PMIx Initialized for description of all relevant attributes and behaviors

# A.4.3 Client.get version

```
Format
                                                    Python —
   PMIx v4.0
3
              vers = myclient.get_version()
                                                    Python
              Returns:
4
              • vers - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)
 5
6
              See PMIx_Get_version for description of all relevant attributes and behaviors
    A.4.4 Client.finalize
              Summary
8
              Finalize the PMIx client library.
9
              Format
10
                                                    Python -
   PMIx v4.0
              rc = myclient.finalize(info:list)
11
                                                    Python -
12
              IN
                   info
                   List of Python info dictionaries (list)
13
14
              Returns:
15
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
16
              See PMIx_Finalize for description of all relevant attributes and behaviors
              Client.abort
    A.4.5
              Summary
18
19
              Request that the provided list of procs be aborted
```

```
Format
1
                                                      Python -
   PMIx v4.0
2
               rc = myclient.abort(status:integer, msg:str, targets:list)
                                                      Python
               IN
3
                    status
4
                   PMIx status to be returned on exit (integer)
5
               IN
                   String message to be printed (string)
6
               IN
                   targets
7
                   List of Python proc dictionaries (list)
8
9
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
10
               See PMIx_Abort for description of all relevant attributes and behaviors
11
              Client.store internal
    A.4.6
               Summary
13
14
               Store some data locally for retrieval by other areas of the process
               Format
15
                                                      Python -
   PMIx v4.0
               rc = myclient.store_internal(proc:dict, key:str, value:dict)
16
                                                      Python
17
               IN
                    proc
                   Python proc dictionary of the process being referenced (dict)
18
19
               IN
                   String key of the data (string)
20
               IN
                    value
21
22
                   Python value dictionary (dict)
23
               Returns:
24
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
25
               See PMIx_Store_internal for details
    A.4.7
              Client.put
               Summary
27
28
               Push a key/value pair into the client's namespace.
```

```
Format
1
                                                    Python —
   PMIx v4.0
2
              rc = myclient.put(scope:integer, key:str, value:dict)
                                                    Python
              IN
3
                   scope
4
                   Scope of the data being posted (integer)
5
              IN
                   String key of the data (string)
6
              IN
                  value
7
                   Python value dictionary (dict)
8
9
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_Put for description of all relevant attributes and behaviors
11
    A.4.8
             Client.commit
              Summary
13
14
              Push all previously PMIxClient.put values to the local PMIx server.
              Format
15
                                   Python
   PMIx v4.0
16
              rc = myclient.commit()
                                                  Python
17
              Returns:
18
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
19
              See PMIx_Commit for description of all relevant attributes and behaviors
             Client.fence
    A.4.9
21
              Summary
22
              Execute a blocking barrier across the processes identified in the specified list
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = myclient.fence(peers:list, directives:list)
                                                      Python
               IN
3
                    peers
 4
                    List of Python proc dictionaries (list)
5
               IN
                  directives
                    List of Python info dictionaries (list)
6
               Returns:
 7
 8
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               See PMIx Fence for description of all relevant attributes and behaviors
    A.4.10
                Client.get
               Summary
11
               Retrieve a key/value pair
12
13
               Format
                                                      Python
   PMIx v4.0
14
               rc, val = myclient.get(proc:dict, key:str, directives:list)
                                                      Python
               IN
15
                   proc
                    Python proc whose data is being requested (dict)
16
17
               IN
                    Python string key of the data to be returned (str)
18
19
               IN
                    directives
20
                    List of Python info dictionaries (list)
               Returns:
21
22
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
               • val - Python value containing the returned data (dict)
24
               See PMIx_Get for description of all relevant attributes and behaviors
    A.4.11
                Client.publish
25
               Summary
26
27
               Publish data for later access via PMIx Lookup.
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = myclient.publish(directives:list)
                                                       Python
               IN
3
                    directives
                    List of Python info dictionaries containing data to be published and directives (list)
               Returns:
 5
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 6
 7
               See PMIx_Publish for description of all relevant attributes and behaviors
    A.4.12
                Client.lookup
               Summary
9
               Lookup information published by this or another process with PMIx Publish.
10
               Format
11
                                                      Python
   PMIx v4.0
12
               rc,info = myclient.lookup(pdata:list, directives:list)
                                                       Python
13
               IN
                   pdata
                    List of Python pdata dictionaries identifying data to be retrieved (list)
14
                    directives
15
                    List of Python info dictionaries (list)
16
17
               Returns:
18
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
               • info - Python list of info containing the returned data (list)
19
20
               See PMIx_Lookup for description of all relevant attributes and behaviors
    A.4.13
                Client.unpublish
22
               Summary
               Delete data published by this process with PMIx Publish.
23
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = myclient.unpublish(keys:list, directives:list)
                                                      Python
               IN
3
                    keys
4
                   List of Python string keys identifying data to be deleted (list)
5
               IN
                   directives
                   List of Python info dictionaries (list)
6
               Returns:
 7
 8
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               See PMIx Unpublish for description of all relevant attributes and behaviors
    A.4.14
                Client.spawn
               Summary
11
               Spawn a new job.
12
13
               Format
                                                      Python -
   PMIx v4.0
               rc,nspace = myclient.spawn(jobinfo:list, apps:list)
14
                                                      Python
               IN
15
                    jobinfo
                   List of Python info dictionaries (list)
16
17
               IN
                    apps
                   List of Python app dictionaries (list)
18
19
               Returns:
20
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
21
               • nspace - Python nspace of the new job (dict)
22
               See PMIx_Spawn for description of all relevant attributes and behaviors
              Client.connect
    A.4.15
               Summary
24
25
               Connect namespaces.
```

```
Format
1
                                                    Python
   PMIx v4.0
2
              rc = myclient.connect(peers:list, directives:list)
                                                    Python
              IN
3
                   peers
 4
                   List of Python proc dictionaries (list)
 5
              IN
                 directives
                   List of Python info dictionaries (list)
6
              Returns:
 7
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              See PMIx Connect for description of all relevant attributes and behaviors
    A.4.16
               Client.disconnect
              Summary
11
              Disconnect namespaces.
12
13
              Format
                                                    Python —
   PMIx v4.0
14
              rc = myclient.disconnect(peers:list, directives:list)
                                                    Python
              IN
15
                   peers
                   List of Python proc dictionaries (list)
16
17
              IN
                  directives
                   List of Python info dictionaries (list)
18
19
              Returns:
20
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_Disconnect for description of all relevant attributes and behaviors
21
    A.4.17 Client.resolve_peers
              Summary
23
24
              Return list of processes within the specified nspace on the given node.
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc,procs = myclient.resolve_peers(node:str, nspace:str)
                                                      Python
               IN
3
                    node
 4
                   Name of node whose processes are being requested (str)
5
               IN
                    nspace
                   Python nspace whose processes are to be returned (str)
6
 7
               Returns:
8
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               • procs - List of Python proc dictionaries (list)
10
               See PMIx_Resolve_peers for description of all relevant attributes and behaviors
                Client.resolve nodes
    A.4.18
12
               Summary
13
               Return list of nodes hosting processes within the specified nspace.
               Format
14
                                                      Python –
   PMIx v4.0
               rc, nodes = myclient.resolve_nodes(nspace:str)
15
                                                      Python
16
               IN
                    nspace
17
                   Python nspace (str)
18
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
19
20
               • nodes - List of Python string node names (list)
21
               See PMIx Resolve nodes for description of all relevant attributes and behaviors
    A.4.19
                Client.query
23
               Summary
               Query information about the system in general
24
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc,info = myclient.query(queries:list)
                                                      Python
               IN
3
                    queries
                    List of Python query dictionaries (list)
 5
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 6
               • info - List of Python info containing results of the query (list)
               See PMIx Query info nb for description of all relevant attributes and behaviors
8
    A.4.20
                Client.log
               Summary
10
               Log data to a central data service/store
11
               Format
12
                                                      Python –
   PMIx v4.0
13
               rc = myclient.log(data:list, directives:list)
                                                      Python
14
               IN
                   data
                    List of Python info (list)
15
               IN
                    directives
16
17
                    Optional list of Python info (list)
18
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
19
20
               See PMIx_Log for description of all relevant attributes and behaviors
    A.4.21
                Client.allocate
22
               Summary
23
               Request an allocation operation from the host resource manager.
```

```
Format
1
                                                     Python —
   PMIx v4.0
2
              rc,info = myclient.allocate(request:integer, directives:list)
                                                     Python
              IN
3
                   request
4
                   Python allocdir specifying requested operation (integer)
5
              IN
                  directives
                   List of Python info describing request (list)
6
              Returns:
 7
8
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              • info - List of Python info containing results of the request (list)
10
              See PMIx Allocation request nb for description of all relevant attributes and behaviors
               Client.job ctrl
    A.4.22
              Summary
12
              Request a job control action
13
              Format
14
                                                     Python ————
   PMIx v4.0
              rc,info = myclient.job_ctrl(targets:list, directives:list)
15
                                                     Python
16
              IN
                   targets
17
                   List of Python proc specifying targets of requested operation (integer)
              IN
                   directives
18
                   List of Python info describing operation to be performed (list)
19
              Returns:
20
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
21
              • info - List of Python info containing results of the request (list)
22
23
              See PMIx_Job_control_nb for description of all relevant attributes and behaviors
    A.4.23 Client.monitor
              Summary
25
26
              Request that something be monitored
```

```
Format
1
                                                    Python —
   PMIx v4.0
2
              rc, info = myclient.monitor(monitor:dict, error_code:integer, directives:list
                                                     Python
              IN
3
                  monitor
4
                   Python info specifying specifying the type of monitor being requested (dict)
5
              IN error_code
                   Status code to be used when generating an event notification alerting that the monitor has
6
7
                   been triggered (integer)
              IN
8
                  directives
9
                   List of Python info describing request (list)
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
11
12
              • info - List of Python info containing results of the request (list)
13
              See PMIx Process monitor nb for description of all relevant attributes and behaviors
    A.4.24
               Client.get credential
              Summary
15
              Request a credential from the PMIx server/SMS
16
17
              Format
                                                     Python -
   PMIx v4.0
18
              rc,cred = myclient.get_credential(directives:list)
                                                     Python
              IN
19
                   directives
                   Optional list of Python info describing request (list)
20
21
              Returns:
22
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
              • cred - Python byteobject containing returned credential (dict)
24
              See PMIx_Get_credential for description of all relevant attributes and behaviors
    A.4.25
               Client.validate_credential
26
              Summary
27
              Request validation of a credential by the PMIx server/SMS
```

```
Format
1
                                          ----- Python ------
   PMIx v4.0
              rc,info = myclient.validate_credential(cred:dict, directives:list)
2
                                                   Python —
              IN
3
                   cred
4
                   Python byteobject containing credential (dict)
              IN
                 directives
 5
6
                   Optional list of Python info describing request (list)
 7
              Returns:
8
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              • info - List of Python info containing additional results of the request (list)
              See PMIx_Validate_credential for description of all relevant attributes and behaviors
10
    A.4.26
               Client.group construct
12
              Summary
              Construct a new group composed of the specified processes and identified with the provided group
13
14
              identifier
              Format
15
                                         _____ Python _____
   PMIx v4.0
              rc,info = myclient.construct_group(grp:string, members:list, directives:list
16
                                                   Python ———
17
              IN
                   grp
                   Python string identifier for the group (str)
18
              IN members
19
                  List of Python proc dictionaries identifying group members (list)
20
              IN
21
                   directives
22
                   Optional list of Python info describing request (list)
23
              Returns:
24
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
25
              • info - List of Python info containing results of the request (list)
26
              See PMIx_Group_construct for description of all relevant attributes and behaviors
    A.4.27 Client.group invite
              Summary
28
29
              Explicitly invite specified processes to join a group
```

```
Format
1
                                           ——— Python ——————
   PMIx v4.0
              rc, info = myclient.group_invite(grp:string, members:list, directives:list)
2
                                                     Python —
              IN
3
                   grp
4
                   Python string identifier for the group (str)
5
              IN
                  members
                   List of Python proc dictionaries identifying processes to be invited (list)
6
7
              IN
                  directives
                   Optional list of Python info describing request (list)
8
9
              Returns:
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
10
              • info - List of Python info containing results of the request (list)
11
12
              See PMIx Group invite for description of all relevant attributes and behaviors
    A.4.28 Client.group_join
13
              Summary
14
              Respond to an invitation to join a group that is being asynchronously constructed
15
              Format
16
                                           ——— Python ———————
   PMIx v4.0
              rc,info = myclient.group_join(grp:string, leader:dict, opt:integer, directiv
17
                                                     Pvthon
              IN
18
                   grp
                   Python string identifier for the group (str)
19
              IN
20
                   Python proc dictionary identifying process leading the group (dict)
21
22
              IN
                   One of the pmix group opt t values indicating decline/accept (integer)
23
24
              IN
                   directives
                   Optional list of Python info describing request (list)
25
26
              Returns:
27
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              • info - List of Python info containing results of the request (list)
28
29
              See PMIx_Group_join for description of all relevant attributes and behaviors
```

### A.4.29 Client.group\_leave Summary 2 3 Leave a PMIx Group **Format** Python PMIx v4.0 5 rc = myclient.group\_leave(grp:string, directives:list) Python -6 IN grp 7 Python string identifier for the group (str) IN directives 8 9 Optional list of Python **info** describing request (list) Returns: 10 11 • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer) 12 See PMIx\_Group\_leave for description of all relevant attributes and behaviors A.4.30 Client.group destruct Summary 14 Destruct a PMIx Group 15 16 Format Python -PMIx v4.0 17 rc = myclient.group\_destruct(grp:string, directives:list) Python — IN 18 grp 19 Python string identifier for the group (str) directives IN 20 Optional list of Python **info** describing request (list) 21 22 Returns: 23 • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer) 24 See PMIx Group destruct for description of all relevant attributes and behaviors A.4.31 Client.register event handler 25 Summarv 26

Register an event handler to report events.

| 1        |           | Format Python —  |
|----------|-----------|--|
|          | PMIx v4.0 |  |
| 2        |           | rc,id = myclient.register_event_handler(codes:list, directives:list, cbfunc)  Python         |
| _        |           |  |
| 3<br>4   |           | IN codes List of Python integer status codes that should be reported to this handler (llist) |
| 5        |           | IN directives  |
| 6        |           | Optional list of Python info describing request (list)                                       |
| 7<br>8   |           | N cbfunc Python evhandler to be called when event is received (func)                         |
|          |           |  |
| 9        |           | Returns:   |
| 10       |           | • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)     |
| 11       |           | • <i>id</i> - PMIx reference identifier for handler (integer)                                |
| 12       |           | See PMIx_Register_event_handler for description of all relevant attributes and behaviors     |
|          |           |  |
| 13       | A.4.32    | Client.deregister_event_handler  |
| 14<br>15 |           | Summary Deregister an event handler  |
| 16       | PMIx v4.0 | Format Python —  |
| 17       |           | <pre>myclient.deregister_event_handler(id:integer)</pre>                                     |
|          |           | Python —   |
| 18<br>19 |           | IN id PMIx reference identifier for handler (integer)  |
| 20       |           | Returns: None  |
| 21<br>22 |           | See PMIx_Deregister_event_handler for description of all relevant attributes and behaviors   |
| 23       | A.4.33    | Client.notify_event  |
| 24<br>25 |           | Summary Report an event for notification via any registered handler.                         |

```
Format
1
                                                      Python –
   PMIx v4.0
2
               rc = myclient.notify_event(status:integer, source:dict,
                                                  range:integer, directives:list)
3
                                                      Python
4
               IN
                  status
                   PMIx status code indicating the event being reported (integer)
 5
               IN
6
                    source
7
                   Python proc of the process that generated the event (dict)
8
               IN
                    range
9
                   Python range in which the event is to be reported (integer)
                    directives
10
               IN
                   Optional list of Python info dictionaries describing the event (list)
11
12
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
13
               See PMIx Notify event for description of all relevant attributes and behaviors
14
    A.4.34 Client.fabric_register
15
               Summary
16
               Register for access to fabric-related information, including communication cost matrix.
17
18
               Format
                                                — Python -
   PMIx v4.0
               rc, fabricinfo = myserver.fabric_register(directives:list)
19
                                                      Python
               IN
20
                    directives
                   Optional list of Python info containing directives (list)
21
22
               Returns:
23
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24
               • fabricinfo - List of Python info containing fabric info (list)
25
               See PMIx_Fabric_register for details
    A.4.35
               Client.fabric_update
26
               Summarv
27
28
               Update fabric-related information, including communication cost matrix.
```

```
Format
1
                                                  Python
   PMIx v4.0
2
              rc, fabricinfo = myserver.fabric_update()
                                                  Python
3
              Returns:
 4
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 5
              • fabricinfo - List of Python info containing updated fabric info (list)
              See PMIx Fabric update for details
 6
    A.4.36
              Client.fabric deregister
              Summary
8
9
              Deregister fabric
10
              Format
                                                  Python
   PMIx v4.0
              rc = myserver.fabric_deregister()
11
                                                  Python
12
              Returns:
13
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_Fabric_deregister for details
14
    A.4.37 Client.fabric get vertex info
              Summary
16
              Given a communication cost matrix index for a specified fabric, return an array of information
17
              describing the corresponding NIC.
18
19
              Format
                                              PMIx v4.0
              rc,nicinfo = myserver.fabric_get_vertex_info(index:integer)
20
                                                  Python
              Returns:
21
22
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
              • nicinfo - List of Python info describing the referenced NIC (list)
24
              See PMIx_Fabric_get_vertex_info for details
```

#### A.4.38 Client.fabric get device index Summary 2 3 Given info describing a given vertex, return the corresponding communication cost matrix index **Format** Python -PMIx v4.0 5 rc,index = myserver.fabric\_get\_device\_index(info:list) Python -IN info 6 List of Python **info** containing vertex description (list) 7 8 Returns: 9 • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer) • *index* - Index of corresponding NIC (integer) 10 See PMIx\_Fabric\_get\_device\_index for details 11 A.4.39 Client.error\_string 13 Summary Pretty-print string representation of pmix\_status\_t . 14 **Format** 15 Python ————— PMIx v4.0 rep = myclient.error\_string(status:integer) 16 Python IN 17 status PMIx status code (integer) 18 Returns: 19 20 • rep - String representation of the provided status code (str) 21 See PMIx Error string for further details

#### 2 A.4.40 Client.proc state string

23 Summary

Pretty-print string representation of pmix\_proc\_state\_t.

| 1        | PMIx v4.0      | Format Python   |
|----------|----------------|---|
| 2        |                | rep = myclient.proc_state_string(state:integer)                   |
|          |                | Python —  |
| 3<br>4   |                | IN state PMIx process state code (integer)                        |
| 5        |                | Returns:  |
| 6        |                | • rep - String representation of the provided process state (str) |
| 7        |                | See PMIx_Proc_state_string for further details                    |
| 8        | <b>A</b> .4.41 | Client.scope_string   |
| 9<br>10  |                | Summary Pretty-print string representation of pmix_scope_t.       |
| 11       |                | Format  |
|          | PMIx v4.0      | Python —  |
| 12       |                | <pre>rep = myclient.scope_string(scope:integer)</pre>             |
| 13<br>14 |                | IN scope PMIx scope value (integer)                               |
| 15       |                | Returns:  |
| 16       |                | • <i>rep</i> - String representation of the provided scope (str)  |
| 17       |                | See PMIx_Scope_string for further details                         |
| 18       | A.4.42         | Client.persistence_string   |
| 19<br>20 |                | Summary Pretty-print string representation of pmix_persistence_t. |
| 21       |                | Format  |
|          | PMIx v4.0      | Python —  |
| 22       |                | <pre>rep = myclient.persistence_string(persistence:integer)</pre> |
| 23<br>24 |                | IN persistence PMIx persistence value (integer)                   |
| 25       |                | Returns:  |
| 26       |                | • rep - String representation of the provided persistence (str)   |
| 27       |                | See PMIx_Persistence_string for further details                   |

#### A.4.43 Client.data range string Summary 3 Pretty-print string representation of pmix\_data\_range\_t. **Format** Python PMIx v4.0 5 rep = myclient.data\_range\_string(range:integer) Python IN 6 range 7 PMIx data range value (integer) 8 Returns: 9 • rep - String representation of the provided data range (str) See PMIx\_Data\_range\_string for further details 10 Client.info directives string A.4.44 Summary 12 13 Pretty-print string representation of **pmix\_info\_directives\_t**. **Format** 14 Python -PMIx v4.0 15 rep = myclient.info\_directives\_string(directives:integer) Python — IN 16 directives PMIx info directives value (integer) 17 Returns: 18 19 • rep - String representation of the provided info directives (str) 20 See PMIx Info directives string for further details

#### 21 A.4.45 Client.data type string

22 Summary

Pretty-print string representation of pmix\_data\_type\_t.

```
Format
1
                                                   Python
   PMIx v4.0
2
              rep = myclient.data_type_string(dtype:integer)
                                                   Python
3
              IN
                   dtype
                   PMIx datatype value (integer)
 4
              Returns:
              • rep - String representation of the provided datatype (str)
6
              See PMIx_Data_type_string for further details
 7
    A.4.46
               Client.alloc directive string
8
              Summary
9
              Pretty-print string representation of pmix alloc directive t.
10
              Format
11
                                                    Python
   PMIx v4.0
12
              rep = myclient.alloc_directive_string(adir:integer)
                                                    Python
13
              IN
                   adir
                   PMIx allocation directive value (integer)
14
15
              Returns:
              • rep - String representation of the provided allocation directive (str)
16
17
              See PMIx Alloc directive string for further details
    A.4.47
               Client.iof channel string
18
              Summary
19
              Pretty-print string representation of pmix_iof_channel_t.
20
              Format
21
                                                   Python
   PMIx v4.0
22
              rep = myclient.iof_channel_string(channel:integer)
                                                   Python
              IN
23
                   channel
                   PMIx IOF channel value (integer)
24
25
              Returns:
26
              • rep - String representation of the provided IOF channel (str)
27
              See PMIx_IOF_channel_string for further details
```

#### A.4.48 Client.job state string Summary 3 Pretty-print string representation of pmix\_job\_state\_t. **Format** Python PMIx v4.0 5 rep = myclient.job\_state\_string(state:integer) Python IN state 6 7 PMIx job state value (integer) 8 Returns: 9 • rep - String representation of the provided job state (str) See PMIx\_Job\_state\_string for further details 10 A.4.49 Client.get attribute string Summary 12 13 Pretty-print string representation of a PMIx attribute. **Format** 14 \_\_\_\_\_ Python \_ PMIx v4.0 15 rep = myclient.get\_attribute\_string(attribute:str) Python — IN 16 attribute PMIx attribute name (string) 17 Returns: 18 19 • rep - String representation of the provided attribute (str) 20 See PMIx Get attribute string for further details

### 21 A.4.50 Client.get\_attribute\_name

22 Summary

Pretty-print name of a PMIx attribute corresponding to the provided string

```
Format
1
                                                     Python
   PMIx v4.0
2
               rep = myclient.get_attribute_name(attribute:str)
                                                      Python
3
               IN
                    attributestring
                   Attribute string (string)
               Returns:
 5
               • rep - Attribute name corresponding to the provided string (str)
6
               See PMIx_Get_attribute_name for further details
    A.4.51
                Client.link state string
9
               Summary
               Pretty-print string representation of pmix_link_state_t.
10
               Format
11
                                                     Python
   PMIx v4.0
12
               rep = myclient.link_state_string(state:integer)
                                                      Python
13
               IN
                    state
14
                   PMIx link state value (integer)
               Returns:
15
16
               • rep - String representation of the provided link state (str)
17
               See PMIx_Link_state_string for further details
           PMIxServer
   A.5
19
               The server Python class inherits the Python "client" class as its parent. Thus, it includes all client
               functions in addition to the ones defined in this section.
20
              Server.init
    A.5.1
22
               Summary
23
               Initialize the PMIx server library after obtaining a new PMIxServer object
```

```
Format
 1
                                                   Python
   PMIx v4.0
 2
              rc = myserver.init(directives:list, map:dict)
                                                   Python
              IN
 3
                   directives
 4
                   List of Python info dictionaries (list)
 5
              IN
                   Python dictionary key-function pairs that map server module callback functions to
 6
                   provided implementations (dict)
 7
              Returns:
 8
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
 9
10
              See PMIx server init for description of all relevant attributes and behaviors
    A.5.2
             Server.finalize
              Summary
12
              Finalize the PMIx server library
13
              Format
14
                                         ——— Python —————
   PMIx v4.0
15
              rc = myserver.finalize()
                                                   Python
              Returns:
16
17
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
18
              See PMIx server finalize for details
    A.5.3
             Server.generate regex
20
              Summary
21
              Generate a regular expression representation of the input strings.
```

| 1              |           | Format Python   |
|----------------|-----------|---|
|                | PMIx v4.0 |   |
| 2              |           | rc, regex = myserver.generate_regex(input:list)  Python   |
|                |           |   |
| 3<br>4         |           | IN input List of Python strings (e.g., node names) (list)   |
| 5              |           | Returns:  |
| 6              |           | • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  |
| 7<br>8         |           | • regex - Python <b>bytearray</b> containing regular expression representation of the input list ( <b>bytearray</b> )   |
| 9              |           | See PMIx_generate_regex for details   |
| 10             | A.5.4     | Server.generate_ppn   |
| 11<br>12       |           | <b>Summary</b> Generate a regular expression representation of the input strings.   |
| 13             | PMIx v4.0 | Format Python   |
| 14             |           | rc,regex = myserver.generate_ppn(input:list)  Python  |
| 15<br>16<br>17 |           | IN input List of Python strings, each string consisting of a comma-delimited list of ranks on each node, with the strings being in the same order as the node names provided to "generate_regex" (list) |
| 18             |           | Returns:  |
| 19             |           | • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  |
| 20<br>21       |           | • regex - Python bytearray containing regular expression representation of the input list (bytearray)   |
| 22             |           | See PMIx_generate_ppn for details   |
| 23             | A.5.5     | Server.register_nspace  |
| 24<br>25       |           | Setup the data about a particular namespace.  |

| 1                           |                  | Format  |
|-----------------------------|------------------|---|
|                             | <i>PMIx v4.0</i> | Python —  |
| 2<br>3<br>4                 |                  | <pre>rc = myserver.register_nspace(nspace:str,</pre>  |
| 5<br>6<br>7<br>8<br>9<br>10 |                  | IN nspace Python string containing the namespace (str)  IN nlocalprocs Number of local processes (integer)  IN directives List of Python info dictionaries (list)  Returns: |
| 12                          |                  | • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  |
| 13                          |                  | See PMIx_server_register_nspace for description of all relevant attributes and behaviors  |
| 14<br>15<br>16              | A.5.6            | Server.deregister_nspace Summary Deregister a namespace.  |
| 17<br>18                    | PMIx v4.0        | Format  Python  myserver.deregister_nspace(nspace:str)  |
| 19<br>20                    |                  | IN nspace Python string containing the namespace (str)  |
| 21                          |                  | Returns: None   |
| 22                          |                  | See PMIx_server_deregister_nspace for details   |
| 23                          | A.5.7            | Server.register_client  |
| 24                          |                  | Summary   |
| 25                          |                  | Register a client process with the PMIx server library.   |

```
Format
1
                                         ——— Python ——————
   PMIx v4.0
2
              rc = myserver.register_client(proc:dict, uid:integer, gid:integer)
                                                   Python
              IN
3
                  proc
4
                  Python proc dictionary identifying the client process (dict)
5
              IN
                  Linux uid value for user executing client process (integer)
6
              IN
7
                  Linux gid value for user executing client process (integer)
8
9
              Returns:
10
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              See PMIx_server_register_client for details
11
             Server.deregister_client
    A.5.8
              Summary
13
14
              Dergister a client process and purge all data relating to it
              Format
15
                                                   Python -
   PMIx v4.0
              myserver.deregister_client(proc:dict)
16
                                                   Python
17
              IN
                   proc
                  Python proc dictionary identifying the client process (dict)
18
              Returns: None
19
20
              See PMIx_server_deregister_client for details
    A.5.9
             Server.setup_fork
22
              Summary
23
              Setup the environment of a child process that is to be forked by the host
```

```
Format
1
                                                     Python
   PMIx v4.0
2
              rc = myserver.setup_fork(proc:dict, envin:dict)
                                                     Python
              IN
3
                  proc
 4
                   Python proc dictionary identifying the client process (dict)
              INOUT envin
5
                   Python dictionary containing the environment to be passed to the client (dict)
6
 7
              Returns:
              • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              See PMIx server setup fork for details
    A.5.10
               Server.dmodex request
              Summary
11
              Function by which the host server can request modex data from the local PMIx server.
12
13
              Format
                                                     Python —
   PMIx v4.0
              rc,data = myserver.dmodex_request(proc:dict)
14
                                                     Python
              IN
15
                   proc
                   Python proc dictionary identifying the process whose data is requested (dict)
16
17
18
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
              • data - Python byteobject containing the returned data (dict)
19
20
              See PMIx_server_dmodex_request for details
               Server.setup_application
    A.5.11
22
              Summary
23
              Function by which the resource manager can request application-specific setup data prior to launch
24
              of a job.
```

| 1                          |           | Format Python  |
|----------------------------|-----------|--|
| _                          | PMIx v4.0 |  |
| 2                          |           | rc,info = myserver.setup_application(nspace:str, directives:list)  |
| 3<br>4<br>5<br>6<br>7<br>8 |           | IN nspace Namespace whose setup information is being requested (str) IN directives Python list of info directives  Returns:  • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  • info - Python list of info dictionaries containing the returned data (list) |
| 10                         |           | See PMIx_server_setup_application for details  |
| 11                         | A.5.12    | Server.register_attributes   |
| 12<br>13                   |           | <b>Summary</b> Register host environment attribute support for a function.   |
| 14                         |           | Format   |
|                            | PMIx v4.0 | Python —   |
| 15                         |           | <pre>rc = myserver.register_attributes(function:str, attrs:list)</pre>   |
| 16<br>17<br>18<br>19       |           | <ul> <li>IN function         Name of the function (str)</li> <li>IN attrs         Python list of regattr describing the supported attributes</li> </ul>  |
| 20                         |           | Returns:   |
| 21                         |           | • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)   |
| 22                         |           | See PMIx_Register_attributes for details   |
| 23                         | A.5.13    | Server.setup_local_support   |
| 24<br>25<br>26             |           | <b>Summary</b> Function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application  |

```
Format
1
                                                   Python —
   PMIx v4.0
2
              rc = myserver.setup_local_support(nspace:str, info:list)
                                                   Python
3
              IN
                  nspace
4
                  Namespace whose setup information is being requested (str)
 5
              IN
6
                  Python list of info containing the setup data (list)
 7
              Returns:
8
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
              See PMIx_server_setup_local_support for details
    A.5.14 Server.iof_deliver
              Summary
11
              Function by which the host environment can pass forwarded IO to the PMIx server library for
12
13
              distribution to its clients.
              Format
14
                                         _____ Pvthon _____
   PMIx v4.0
15
              rc = myserver.iof_deliver(source:dict, channel:integer,
                                              data:dict, directives:list)
16
                                                   Python ———
              IN
17
                 source
18
                  Python proc dictionary identifying the process who generated the data (dict)
              IN
                  channel
19
                  Python channel bitmask identifying IO channel of the provided data (integer)
20
              IN
21
22
                  Python byteobject containing the data (dict)
                 directives
23
                  Python list of info containing directives (list)
24
25
26
              • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
27
              See PMIx_server_IOF_deliver for details
    A.5.15 Server.collect inventory
29
              Summary
30
              Collect inventory of resources on a node
```

```
Format
1
                                                      Python -
   PMIx v4.0
2
               rc, info = myserver.collect_inventory(directives:list)
                                                      Python
               IN
3
                    directives
                    Optional Python list of info containing directives (list)
 5
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
6
 7
               • info - Python list of info containing the returned data (list)
               See PMIx_server_collect_inventory for details
8
    A.5.16 Server.deliver inventory
10
               Summary
               Pass collected inventory to the PMIx server library for storage
11
12
                                                      Python
   PMIx v4.0
13
               rc = myserver.deliver_inventory(info:list, directives:list)
                                                      Python
               IN
14
                    info
                    - Python list of info dictionaries containing the inventory data (list)
15
               IN
16
                    directives
                    Python list of info dictionaries containing directives (list)
17
               Returns:
18
19
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
20
               See PMIx_server_deliver_inventory for details
    A.6 PMIxTool
22
               The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and
23
               server functions in addition to the ones defined in this section.
    A.6.1 Tool.init
25
               Summary
26
               Initialize the PMIx tool library after obtaining a new PMIxTool object
```

```
Format
1
                                                      Python
   PMIx v4.0
               rc,proc = mytool.init(info:list)
2
                                                       Python
               IN
3
                    info
                    List of Python info directives (list)
 5
               Returns:
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
6
               • proc - a Python proc (dict)
 7
               See PMIx tool init for description of all relevant attributes and behaviors
8
    A.6.2
              Tool.finalize
               Summary
10
               Finalize the PMIx tool library, closing the connection to the server.
11
12
               Format
                                                      Python
   PMIx v4.0
13
               rc = mytool.finalize()
                                                      Python
14
               Returns:
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
15
16
               See PMIx_tool_finalize for description of all relevant attributes and behaviors
    A.6.3
              Tool.connect_to_server
18
               Summary
19
               Switch connection from the current PMIx server to another one, or initialize a connection to a
20
               specified server.
```

```
Format
1
                                                      Python –
   PMIx v4.0
2
               rc,proc = mytool.connect_to_server(info:list)
                                                      Python
               IN
                    info
3
 4
                   List of Python info dictionaries (list)
               Returns:
 5
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
6
 7
               • proc - a Python proc (dict)
               See PMIx tool connect to server for description of all relevant attributes and behaviors
8
    A.6.4 Tool.iof_pull
               Summary
10
               Register to receive output forwarded from a remote process.
11
               Format
12
                                            ----- Python -
   PMIx v4.0
               rc,id = mytool.iof_pull(sources:list, channel:integer, directives:list, cbfu
13
                                                      Python
               IN
14
                    sources
                   List of Python proc dictionaries of processes whose IO is being requested (list)
15
               IN channel
16
                   Python channel bitmask identifying IO channels to be forwarded (integer)
17
18
               IN directives
                   List of Python info dictionaries describing request (list)
19
               IN
20
                    cbfunc
                   Python iofcbfunc to receive IO payloads (func)
21
               Returns:
22
23
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24
               • id - PMIx reference identifier for request (integer)
               See PMIx_IOF_pull for description of all relevant attributes and behaviors
25
              Tool.iof_deregister
    A.6.5
               Summary
27
28
               Deregister from output forwarded from a remote process.
```

```
Format
1
                                                      Python
   PMIx v4.0
2
               rc = mytool.iof_deregister(id:integer, directives:list)
                                                      Python
               IN
3
                    id
 4
                    PMIx reference identifier returned by pull request (list)
5
               IN
                  directives
                    List of Python info dictionaries describing request (list)
6
 7
               Returns:
               • rc - PMIX SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9
               See PMIx IOF deregister for description of all relevant attributes and behaviors
              Tool.iof push
    A.6.6
               Summary
11
               Push data collected locally (typically from stdin) to stdin of target recipients
12
13
               Format
                                                      Python ———
   PMIx v4.0
               rc = mytool.iof_push(targets:list, data:dict, directives:list)
14
                                                      Python
               IN
15
                    sources
                    List of Python proc of target processes (list)
16
17
               IN
                    data
                    Python byteobject containing data to be delivered (dict)
18
               IN
                    directives
19
                    Optional list of Python info describing request (list)
20
21
               Returns:
22
               • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
23
               See PMIx_IOF_push for description of all relevant attributes and behaviors
```

#### **A.**7 **Example Usage**

The following examples are provided to illustrate the use of the Python bindings.

#### 1 A.7.1 Python Client

2

3

5

41

The following example contains a client program that illustrates a fairly common usage pattern. The program instantiates and initializes the PMIxClient class, posts some data that is to be shared across all processes in the job, executes a "fence" that circulates the data, and then retrieves a value posted by one of its peers. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *
6
7
8
            def main():
9
                # Instantiate a client object
                myclient = PMIxClient()
10
                print("Testing PMIx ", myclient.get version())
11
12
                # Initialize the PMIx client library, declaring the programming model
13
14
                # as "TEST" and the library name as "PMIX", just for the example
                info = ['key':PMIX_PROGRAMMING_MODEL,
15
16
                          'value': 'TEST', 'val type': PMIX STRING,
                         'key': PMIX_MODEL_LIBRARY_NAME,
17
18
                          'value':'PMIX', 'val_type':PMIX_STRING]
19
                rc, myname = myclient.init(info)
                if PMIX SUCCESS != rc:
20
21
                    print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
22
                    exit(1)
23
                # try posting a value
24
25
                rc = myclient.put(PMIX_GLOBAL, "mykey",
                                   'value':1, 'val_type':PMIX_INT32)
26
27
                if PMIX SUCCESS != rc:
28
                    print("PMIx Put FAILED WITH ERROR", myclient.error string(rc))
29
                    # cleanly finalize
                    myclient.finalize()
30
31
                    exit(1)
32
                # commit it
33
                rc = myclient.commit()
34
                if PMIX_SUCCESS != rc:
35
                    print("PMIx_Commit FAILED WITH ERROR",
36
37
                           myclient.error_string(rc))
38
                    # cleanly finalize
                    myclient.finalize()
39
40
                    exit(1)
```

```
1
                # execute fence across all processes in my job
2
                procs = []
3
                info = []
4
                rc = myclient.fence(procs, info)
5
                if PMIX SUCCESS != rc:
6
                    print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
7
                    # cleanly finalize
8
                    myclient.finalize()
9
                    exit(1)
10
11
                # Get a value from a peer
12
                if 0 != myname['rank']:
13
                    info = []
14
                    rc, get_val = myclient.get('nspace':"testnspace", 'rank': 0,
15
                                                 "mykey", info)
                    if PMIX_SUCCESS != rc:
16
17
                        print("PMIx_Commit FAILED WITH ERROR",
18
                               myclient.error_string(rc))
19
                        # cleanly finalize
20
                        myclient.finalize()
21
                        exit(1)
22
                    print("Get value returned: ", get val)
23
24
                # test a fence that should return not supported because
25
                # we pass a required attribute that the server is known
26
                # not to support
27
                procs = []
28
                info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
29
                          'value':10, 'val_type':PMIX_INT]
30
                rc = myclient.fence(procs, info)
31
                if PMIX_SUCCESS == rc:
32
                    print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
33
                    # cleanly finalize
34
                    myclient.finalize()
35
                    exit(1)
36
37
                # Publish something
38
                info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
                rc = myclient.publish(info)
39
40
                if PMIX SUCCESS != rc:
41
                    print ("PMIx Publish FAILED WITH ERROR",
42
                          myclient.error string(rc))
43
                    # cleanly finalize
```

```
1
                    myclient.finalize()
2
                    exit(1)
3
4
                # finalize
5
                info = []
6
                myclient.finalize(info)
7
                print("Client finalize complete")
8
9
            # Python main program entry point
            if __name__ == '__main__':
10
11
                main()
                                            Python
```

### 12 A.7.2 Python Server

13

14

15

16 17 The following example contains a minimum-level server host program that instantiates and initializes the PMIxServer class. The program illustrates passing several server module functions to the bindings and includes code to setup and spawn a simple client application, waiting until the spawned client terminates before finalizing and exiting itself. Note that the example has been formatted to fit the document layout.

Python

```
from pmix import *
18
19
            import signal, time
20
            import os
            import select
21
22
            import subprocess
23
            def clientconnected(proc:tuple is not None):
24
25
                print("CLIENT CONNECTED", proc)
26
                return PMIX_OPERATION_SUCCEEDED
27
28
            def clientfinalized(proc:tuple is not None):
29
                print("CLIENT FINALIZED", proc)
                return PMIX_OPERATION_SUCCEEDED
30
31
32
            def clientfence(procs:list, directives:list, data:bytearray):
33
                # check directives
                if directives is not None:
34
                    for d in directives:
35
                         # these are each an info dict
36
37
                         if "pmix" not in d['key']:
38
                             # we do not support such directives - see if
```

```
1
                             # it is required
2
                             try:
3
                                 if d['flags'] & PMIX INFO REQD:
4
                                     # return an error
5
                                     return PMIX ERR NOT SUPPORTED
6
                             except:
7
                                 #it can be ignored
8
                                 pass
9
                return PMIX OPERATION SUCCEEDED
10
11
            def main():
12
                try:
13
                    myserver = PMIxServer()
14
                except:
15
                    print("FAILED TO CREATE SERVER")
16
                    exit(1)
17
                print("Testing server version ", myserver.get_version())
18
19
                args = ['key':PMIX_SERVER_SCHEDULER,
20
                          'value':'T', 'val_type':PMIX_BOOL]
21
                map = 'clientconnected': clientconnected,
22
                        'clientfinalized': clientfinalized,
23
                        'fencenb': clientfence
24
                my result = myserver.init(args, map)
25
26
                # get our environment as a base
27
                env = os.environ.copy()
28
29
                # register an nspace for the client app
30
                (rc, regex) = myserver.generate_regex("test000, test001, test002")
31
                (rc, ppn) = myserver.generate_ppn("0")
32
                kvals = ['key':PMIX_NODE_MAP,
33
                           'value':regex, 'val_type':PMIX_STRING,
                          'key':PMIX_PROC_MAP,
34
35
                           'value':ppn, 'val_type':PMIX_STRING,
36
                          'key':PMIX_UNIV_SIZE,
37
                           'value':1, 'val type':PMIX UINT32,
38
                          'key':PMIX JOB SIZE,
39
                           'value':1, 'val type':PMIX UINT32]
40
                rc = foo.register_nspace("testnspace", 1, kvals)
41
                print("RegNspace ", rc)
42
43
                # register a client
```

```
1
                uid = os.getuid()
2
                gid = os.getgid()
3
                rc = myserver.register_client('nspace':"testnspace", 'rank':0,
4
                                               uid, gid)
5
                print("RegClient ", rc)
6
                # setup the fork
7
                rc = myserver.setup fork('nspace':"testnspace", 'rank':0, env)
8
                print("SetupFrk", rc)
9
10
                # setup the client argv
                args = ["./client.py"]
11
12
                # open a subprocess with stdout and stderr
13
                # as distinct pipes so we can capture their
14
                # output as the process runs
                p = subprocess.Popen(args, env=env,
15
16
                    stdout=subprocess.PIPE, stderr=subprocess.PIPE)
17
                # define storage to catch the output
18
                stdout = []
19
                stderr = []
20
                # loop until the pipes close
                while True:
21
22
                    reads = [p.stdout.fileno(), p.stderr.fileno()]
23
                    ret = select.select(reads, [], [])
24
25
                    stdout done = True
26
                    stderr_done = True
27
28
                    for fd in ret[0]:
29
                         # if the data
                         if fd == p.stdout.fileno():
30
31
                             read = p.stdout.readline()
32
                             if read:
33
                                 read = read.decode('utf-8').rstrip()
                                 print('stdout: ' + read)
34
35
                                 stdout done = False
                         elif fd == p.stderr.fileno():
36
37
                             read = p.stderr.readline()
38
                             if read:
39
                                 read = read.decode('utf-8').rstrip()
40
                                 print('stderr: ' + read)
41
                                 stderr done = False
42
43
                    if stdout done and stderr done:
```

#### **APPENDIX B**

# Acknowledgements

| 1<br>2<br>3<br>4 |     | This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations. |
|------------------|-----|--|
| 5                | B.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<br>13         |     | The following institutions supported this effort through time and travel support for the people listed 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)  |

## <sub>1</sub> B.2 Version 2.0

| 2        | The following list includes some of the active participants in the PMIx v2 standardization process.                                   |
|----------|---|
| 3<br>4   | <ul> <li>Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm<br/>and Terry Wilmarth</li> </ul> |
| 5        | • Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi  |
| 6        | Aurelien Bouteiller and George Bosilca  |
| 7        | <ul> <li>Artem Polyakov, Igor Ivanov and Boris Karasev</li> </ul>   |
| 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       | <ul> <li>Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt</li> </ul>  |
| 17       | Adam Moody and Martin Schulz  |
| 18       | Ryan Grant and Stephen Olivier  |
| 19       | Michael Karo  |
| 20<br>21 | The following institutions supported this effort through time and travel support for the people listed 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       | <ul> <li>Research Organization for Information Science and Technology</li> </ul>  |
| 29       | • HPE Co.   |
|          |   |

- 1 Allinea (ARM)
- SchedMD, Inc.
- Fujitsu Limited
- Los Alamos National Laboratory
- Adaptive Solutions, Inc.
- 6 INRIA
- Oak Ridge National Laboratory
- Lawrence Livermore National Laboratory
- Sandia National Laboratory
- 10 Altair

## B.3 Version 1.0

- The following list includes some of the active participants in the PMIx v1 standardization process.
- Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- Joshua Hursey and David Solt
  - Aurelien Bouteiller and George Bosilca
    - Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- Gilles Gouaillardet
- Gary Brown
- Moe Jette

- The following institutions supported this effort through time and travel support for the people listed above.
- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- Mellanox, Inc.
- Research Organization for Information Science and Technology
- Adaptive Solutions, Inc.
- SchedMD, Inc.

# **Bibliography**

[1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMIx: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.

# Index

General terms and other items not induced in the other indices.

```
application, 10, 12, <u>15</u>, 77, 136, 194, 257, 259
fabric, 16
fabric plane, <u>16</u>, 346
fabric planes, <u>16</u>, 336
fabrics, 16
host environment, \underline{16}
job, 10, 12, <u>15</u>, 77, 78, 136, 138, 194, 251, 256, 257, 259, 269, 271, 428
namespace, <u>15</u>
rank, 15, 138, 260
resource manager, 16
scheduler, <u>16</u>, 345
session, 10, 12, <u>15</u>, 77, 136, 194, 256
slot, <u>15</u>
slots, <u>15</u>
workflow, <u>15</u>
workflows, <u>15</u>, 108
```

## Index of APIs

```
PMIx_Abort, 8, 33, 157, 158, 282, 283, 388, 402
    PMIxClient.abort (Python), 401
PMIx Alloc directive string, 9, 113, 421
    PMIxClient.alloc directive string (Python), 421
PMIx Allocation request, 11–13, 90, 195, 196, 201
PMIx_Allocation_request_nb, 9, 90, 183, 199, 202, 410
    PMIxClient.allocate (Python), 409
PMIx Commit, 8, 106, 128, 129, 141, 141, 268, 288, 403
    PMIxClient.commit (Python), 403
PMIx_Connect, 8, 9, 24, 162, 168, 170, 172, 174, 352–354, 407
    PMIxClient.connect (Python), 406
PMIx_Connect_nb, 8, <u>170</u>, 170
pmix_connection_cbfunc_t, 106, 309
pmix_credential_cbfunc_t, 107, 242, 323
PMIx_Data_copy, 9, 236
PMIx_Data_copy_payload, 9, 237
PMIx_Data_pack, 9, 232, 233, 249, 250
PMIx_Data_print, 9, 236
PMIx Data range string, 9, 112, 420
    PMIxClient.data range string (Python), 420
PMIx_Data_type_string, 9, <u>112</u>, 421
    PMIxClient.data_type_string (Python), 420
PMIx Data unpack, 9, 234
PMIx Deregister event handler, 9, 13, 224, 415
    PMIxClient.deregister event handler (Python), 415
PMIx Disconnect, 8, 9, 24, 170, 172, 174, 176, 354, 407
    PMIxClient.disconnect (Python), 407
PMIx_Disconnect_nb, 8, 174, 176, 354
pmix_dmodex_response_fn_t, 105, 268
PMIx_Error_string, 8, 111, 418
    PMIxClient.error_string (Python), 418
pmix_event_notification_cbfunc_fn_t, 102, 102, 103
pmix_evhdlr_reg_cbfunc_t, 101, 101, 222
PMIx Fabric deregister, 347, 417
    PMIxClient.fabric deregister (Python), 417
PMIx_Fabric_get_device_index, 350, 418
    PMIxClient.fabric_get_device_index (Python), 418
PMIx Fabric get vertex info, 347, 417
```

```
PMIxClient.fabric_get_vertex_info (Python), 417
PMIx_Fabric_register, 339, 345, 416
    PMIxClient.fabric register (Python), 416
PMIx Fabric update, 346, 347, 417
    PMIxClient.fabric update (Python), 416
PMIx_Fence, 3, 6, 8, 14, 127, 128, 141, 143, 145, 170, 174, 268, 283, 286, 352, 358, 363, 388, 404
    PMIxClient.fence (Python), 403
PMIx Fence nb, 8, 12, 99, 143, 145, 283, 286, 388
PMIx Finalize, 8, 24, 26, 33, 81, 118, 119, 119, 167, 281, 282, 387, 401
    PMIxClient.finalize (Python), 401
PMIx_generate_ppn, 8, 249, 425
    PMIxServer.generate_ppn (Python), 425
PMIx_generate_regex, 8, 248, 250, 256, 425
    PMIxServer.generate_regex (Python), 424
PMIx_Get, 3, 8–10, 25, 38, 70–74, 78, 80–87, 89, 90, 92, 93, 118, 129, 130, 132, 134–138, 140,
         160–162, 164–167, 189, 194, 195, 251, 253, 254, 277, 296–298, 336, 343, 351, 358, 404
    PMIxClient.get (Python), 404
PMIx Get attribute name, 113, 423
    PMIxClient.get attribute name (Python), 422
PMIx_Get_attribute_string, 113, 422
    PMIxClient.get_attribute_string (Python), 422
PMIx Get credential, 11, 13, 94, 240, 324, 411
    PMIxClient.get credential (Python), 411
PMIx Get credential nb, 241
PMIx Get nb, 8, 18, 100, 132
PMIx Get version, 8, 18, 116, 401
    PMIxClient.get_version (Python), 401
PMIx_Group_construct, 353, 355, 357, 358, 361, 412
    PMIxClient.group_construct (Python), 412
PMIx_Group_construct_nb, 358, 361
PMIx_Group_destruct, 354, 361, 363, 365, 414
    PMIxClient.group_destruct (Python), 414
PMIx_Group_destruct_nb, 363, 365
PMIx_Group_invite, 353, 365, 368, 369, 371, 413
    PMIxClient.group_invite (Python), 412
PMIx_Group_invite_nb, 369
PMIx Group join, 61, 353, 368, 371, 371, 373, 374, 376, 413
    PMIxClient.group join (Python), 413
PMIx Group join nb, 371, 374, 376
PMIx_Group_leave, 354, 376, 377, 378, 414
    PMIxClient.group leave (Python), 414
PMIx Group leave nb, 377
pmix hdlr reg cbfunc t, 110, 177, 179
```

```
pmix_info_cbfunc_t, 97, 100, 100, 190, 199, 205, 208, 211, 212, 276, 310, 315, 317, 320, 331, 333,
         359, 369, 375
PMIx Info directives string, 9, 112, 420
    PMIxClient.info directives string (Python), 420
PMIx Init, 9, 85, 89, 116, 116, 118, 119, 161, 165, 280, 297, 400
    PMIxClient.init (Python), 400
PMIx Initialized, 8, 115, 400
    PMIxClient.initialized (Python), 400
pmix iof cbfunc t, 109, 177, 386
PMIx_IOF_channel_string, 11, 113, 421
    PMIxClient.iof_channel_string (Python), 421
PMIx_IOF_deregister, 11, 13, 179, 434
    PMIxTool.iof deregister (Python), 433
PMIx_IOF_pull, 11, 13, 177, 179, 433
    PMIxTool.iof_pull (Python), 433
PMIx_IOF_push, 11, 13, 180, 434
    PMIxTool.iof_push (Python), 434
PMIx_Job_control, 11, 13, 92, 202, 204, 207, 208, 319
PMIx_Job_control_nb, 9, 92, 183, 202, 205, 255, 410
    PMIxClient.job_ctrl (Python), 410
PMIx_Job_state_string, 113, 422
    PMIxClient.job state string (Python), 422
PMIx Link state string, 114, 423
    PMIxClient.link state string (Python), 423
PMIx_Log, 11, 213, 215, 409
    PMIxClient.log (Python), 409
PMIx_Log_nb, 9, 87, 215, 218
PMIx_Lookup, 8, 51, 146, 149, 152, 153, 404, 405
    PMIxClient.lookup (Python), 405
pmix_lookup_cbfunc_t, 99, 99, 291
PMIx_Lookup_nb, 99, 100, 152
pmix_modex_cbfunc_t, 97, 97, 284, 287
pmix_notification_fn_t, 103, 103, 222, 387
PMIx_Notify_event, 9, 13, 225, 416
    PMIxClient.notify_event (Python), 415
pmix_op_cbfunc_t, 99, 99, 102, 105, 148, 155, 171, 175, 181, 216, 225, 226, 251, 264–266, 273,
         275, 277, 280, 281, 283, 288, 293, 299, 302, 304, 306, 307, 313, 318, 321, 327, 329, 364,
         377
PMIx Persistence string, 9, 112, 419
    PMIxClient.persistence string (Python), 419
PMIx Proc state string, 9, 111, 419
    PMIxClient.proc state string (Python), 418
PMIx Process monitor, 11, 13, 208, 212
```

```
PMIx_Process_monitor_nb, 9, 93, 183, 210, 212, 411
    PMIxClient.monitor (Python), 410
PMIx Publish, 8, 36, 81, 146, 147–149, 289, 290, 405
    PMIxClient.publish (Python), 404
PMIx Publish nb, 8, 148, 149
PMIx_Put, 8, 35, 36, 38, 106, 128, 128, 129, 132, 135, 141, 143, 167, 194, 268, 288, 358, 368, 403
    PMIxClient.put (Python), 402
PMIx Query info, 185, 189, 194, 343
PMIx Query info nb, 9, 10, 56, 78, 86, 140, 167, 183, 189, 189, 194, 195, 272, 336, 351, 352, 409
    PMIxClient.query (Python), 408
PMIx_Register_attributes, 13, 271, 429
    PMIxServer.register_attributes (Python), 429
PMIx Register event handler, 9, 13, 102, 183, 221, 415
    PMIxClient.register_event_handler (Python), 414
pmix_release_cbfunc_t, 97, 97
PMIx_Resolve_nodes, 8, 184, 408
    PMIxClient.resolve_nodes (Python), 408
PMIx_Resolve_peers, 8, 183, 408
    PMIxClient.resolve_peers (Python), 407
PMIx_Scope_string, 9, <u>112</u>, 419
    PMIxClient.scope_string (Python), 419
pmix server abort fn t, 282, 388
pmix server alloc fn t, 315, 395
pmix server client connected fn t, 99, 239, 266, 279, 280, 387
pmix_server_client_finalized_fn_t, 281, 282, 387
PMIx server collect inventory, 11, 275, 431
    PMIxServer.collect_inventory (Python), 430
pmix_server_connect_fn_t, 167, 299, 301, 303, 391
PMIx_server_deliver_inventory, 11, 276, 431
    PMIxServer.deliver_inventory (Python), 431
PMIx_server_deregister_client, 8, 266, 427
    PMIxServer.deregister_client (Python), 427
pmix_server_deregister_events_fn_t, 305, 393
PMIx_server_deregister_nspace, 8, 263, 266, 426
    PMIxServer.deregister_nspace (Python), 426
pmix_server_disconnect_fn_t, 301, 303, 392
pmix server dmodex req fn t, 10, 11, 97, 287, 389
PMIx server dmodex request, 8, 105, 106, 267, 268, 428
    PMIxServer.dmodex request (Python), 428
pmix_server_fabric_fn_t, <u>332</u>, 339, 345, 399
pmix server fencenb fn t, 12, 97, 283, 286, 389
PMIx server finalize, 8, 127, 424
    PMIxServer.finalize (Python), 424
```

```
pmix_server_get_cred_fn_t, 323, 326, 397
pmix server grp fn t, 330, 399
PMIx server init, 8, 116, 125, 272, 278, 342, 424
    PMIxServer.init (Python), 423
PMIx server IOF deliver, 11, 176, 274, 430
    PMIxServer.iof_deliver (Python), 430
pmix server iof fn t, 326, 398
pmix server job control fn t, 317, 396
pmix server listener fn t, 308
pmix_server_log_fn_t, 313, 395
pmix_server_lookup_fn_t, 290, 390
pmix_server_module_t, 125, 127, 272, 273, 278, 278
pmix_server_monitor_fn_t, 320, 396
pmix_server_notify_event_fn_t, 104, 307, 393
pmix_server_publish_fn_t, 288, 389
pmix_server_query_fn_t, 309, 394
PMIx_server_register_client, 8, 239, 265, 266, 280, 282, 427
    PMIxServer.register_client (Python), 426
pmix_server_register_events_fn_t, 303, 392
PMIx_server_register_nspace, 8, 10, 18, 77, 78, 99, 249, 250, 256, 259, 426
    PMIxServer.register_nspace (Python), 425
PMIx server setup application, 9, 12, 104, 105, 269, 274, 277, 429
    PMIxServer.setup application (Python), 428
PMIx server setup fork, 8, 267, 428
    PMIxServer.setup fork (Python), 427
PMIx server setup local support, 9, 273, 430
    PMIxServer.setup_local_support (Python), 429
pmix_server_spawn_fn_t, 98, 294, 391
pmix_server_stdin_fn_t, 329, 398
pmix_server_tool_connection_fn_t, 239, 312, 394
pmix_server_unpublish_fn_t, 292, 390
pmix_server_validate_cred_fn_t, 324, 397
pmix_setup_application_cbfunc_t, 104, 269
PMIx_Spawn, 8, 12, 54, 75, 84, 89, 90, 158, 158, 159, 163, 164, 167, 199, 255, 267, 294, 299, 316,
         391, 406
    PMIxClient.spawn (Python), 406
pmix spawn cbfunc t, 98, 98, 163, 295
PMIx Spawn nb, 8, 54, 98, 163
PMIx Store internal, 8, 135, 402
    PMIxClient.store internal (Python), 402
PMIx tool connect to server, 11, 123, 433
    PMIxTool.connect to server (Python), 432
pmix tool connection cbfunc t, 106, 312
```

```
PMIx_tool_finalize, 9, 123, 432
PMIxTool.finalize (Python), 432
PMIx_tool_init, 9, 71, 116, 119, 123, 432
PMIx_Tool.init (Python), 431
PMIx_Unpublish, 8, 153, 155, 156, 406
PMIxClient.unpublish (Python), 405
PMIx_Unpublish_nb, 8, 155
PMIx_Validate_credential, 11, 13, 243, 412
PMIxClient.validate_credential (Python), 411
PMIx_Validate_credential_nb, 245
pmix_validation_cbfunc_t, 108, 246, 325
pmix_value_cbfunc_t, 18, 100, 100
```

## **Index of Support Macros**

```
PMIX APP CONSTRUCT, 55
PMIX APP CREATE, 55
PMIX APP DESTRUCT, 55
PMIX APP FREE, 55
PMIX APP INFO CREATE, 10, 11, 56
PMIX_ARGV_APPEND, 64
PMIX_ARGV_APPEND_UNIQUE, 65
PMIX ARGV COPY, 67
PMIX ARGV COUNT, 67
PMIX_ARGV_FREE, 66
PMIX_ARGV_JOIN, 67
PMIX_ARGV_SPLIT, 66
PMIX_BYTE_OBJECT_CREATE, 62
PMIX_BYTE_OBJECT_DESTRUCT, 61
PMIX BYTE OBJECT FREE, 62
PMIX_BYTE_OBJECT_LOAD, 62
PMIX CHECK KEY, 28
PMIX_CHECK_NSPACE, 29
PMIX CHECK PROCID, 32
PMIX COORD CONSTRUCT, 337
PMIX COORD CREATE, 337
PMIX COORD DESTRUCT, 337
PMIX COORD FREE, 337
PMIX DATA ARRAY CONSTRUCT, 37, 63
PMIX DATA ARRAY CREATE, 38, 64
PMIX DATA ARRAY DESTRUCT, 37, 63
PMIX_DATA_ARRAY_FREE, 38
PMIX_DATA_ARRAY_RELEASE, 64
PMIX_DATA_BUFFER_CONSTRUCT, 230, 233, 235
PMIX DATA BUFFER CREATE, 230, 233, 235
PMIX_DATA_BUFFER_DESTRUCT, 231
PMIX_DATA_BUFFER_LOAD, 231
PMIX_DATA_BUFFER_RELEASE, 230
PMIX DATA BUFFER UNLOAD, 232, 249, 250
PMIX ENVAR CONSTRUCT, 50
PMIX ENVAR CREATE, 50
PMIX ENVAR DESTRUCT, 50
PMIX ENVAR FREE, 51
```

```
PMIX_ENVAR_LOAD, 51
PMIx Heartbeat, 9, 212
PMIX_INFO_CONSTRUCT, 43
PMIX INFO CREATE, 44, 47, 48
PMIX INFO DESTRUCT, 44
PMIX_INFO_FREE, 44
PMIX INFO IS END, 10, 12, 48
PMIX_INFO_IS_OPTIONAL, 48
PMIX INFO IS REQUIRED, 46, 47, 48
PMIX_INFO_LOAD, 45
PMIX_INFO_OPTIONAL, 47
PMIX_INFO_REQUIRED, 46, 47
PMIX INFO TRUE, 46
PMIX_INFO_XFER, 45, 256
PMIX_PDATA_CONSTRUCT, 52
PMIX_PDATA_CREATE, 52
PMIX_PDATA_DESTRUCT, 52
PMIX_PDATA_FREE, 52
PMIX_PDATA_LOAD, 53
PMIX PDATA XFER, 54
PMIX_PROC_CONSTRUCT, 30, 61
PMIX PROC CREATE, 31
PMIX PROC DESTRUCT, 30
PMIX PROC FREE, 31, 184
PMIX_PROC_INFO_CONSTRUCT, 34
PMIX PROC INFO CREATE, 34
PMIX_PROC_INFO_DESTRUCT, 34
PMIX_PROC_INFO_FREE, 34
PMIX_PROC_LOAD, 31
PMIX_QUERY_CONSTRUCT, 56
PMIX_QUERY_CREATE, 57
PMIX_QUERY_DESTRUCT, 57
PMIX_QUERY_FREE, 57
PMIX_QUERY_QUALIFIERS_CREATE, 10, 11, <u>57</u>
PMIX_REGATTR_CONSTRUCT, 59
PMIX_REGATTR_CREATE, 59
PMIX REGATTR DESTRUCT, 59
PMIX REGATTR FREE, 60
PMIX REGATTR LOAD, 60
PMIX_REGATTR_XFER, 60
PMIX SETENV, 68
PMIX SYSTEM EVENT, 27
PMIX VALUE CONSTRUCT, 40
```

PMIX\_VALUE\_CREATE, 40
PMIX\_VALUE\_DESTRUCT, 40
PMIX\_VALUE\_FREE, 40
PMIX\_VALUE\_GET\_NUMBER, 43
PMIX\_VALUE\_LOAD, 41
PMIX\_VALUE\_UNLOAD, 41
PMIX\_VALUE\_XFER, 42

## **Index of Data Structures**

```
pmix_alloc_directive_t, 49, 49, 70, 113, 315, 383, 421
pmix app t, 10, 11, 54, 54–56, 65, 68, 159, 163, 295, 384
pmix byte object t, 61, 61, 62, 69, 108, 181, 240, 244, 246, 275, 325, 329, 382
pmix coord t, 70, 336, 336, 337
pmix coord view t, 338, 343
pmix data array t, 10, 11, 37, 37, 38, 63, 63, 64, 70, 86, 87, 91, 188, 193, 195, 198, 201, 258–261,
         270, 311, 317, 333, 336, 341, 343, 344, 346, 383
pmix data buffer t, 229, 229-234, 238
pmix data range t, 36, 36, 70, 112, 226, 307, 382, 420
pmix_data_type_t, 37, 38, 41, 43, 45, 53, 60, 63, 64, 69, 69, 70, 112, 233, 235–237, 381, 420
pmix_envar_t, 49, 50, 51, 70, 383
pmix_fabric_operation_t, 333, 339, 339
pmix_fabric_t, 334, 335, 339, 339, 342, 343, 345–348, 350
pmix_group_operation_t, 330, 354, 354, 372, 375
pmix_group_opt_t, 61, 61, 413
pmix_info_directives_t, 46, 46, 47, 70, 112, 383, 420
pmix_info_t, 3, 9, 10, 12, 14, 28, 36, 43, 43–49, 56–58, 60, 70, 77, 78, 88, 91, 92, 101–103, 109,
          110, 117–120, 125, 127, 147, 151, 177, 179, 181, 185, 189, 195–201, 203–205, 207–209,
         212, 215, 218, 226, 240, 242, 244, 246, 256, 258–261, 270, 275–277, 307, 312–314, 316,
         317, 320, 321, 327, 334, 340, 341, 344, 348, 350, 355, 357, 359, 362, 364, 366, 369, 372,
         375–377, 383, 385
pmix_iof_channel_t, 49, 49, 70, 110, 113, 177, 275, 327, 383, 421
pmix job state t, 35, 35, 113, 422
pmix key t, 27, 27, 28, 60, 128, 130, 381
pmix link state t, 114, 338, 338, 342, 344, 349, 384, 423
pmix nspace t, 28, 28, 29, 31, 98, 381, 382
pmix_pdata_t, <u>51</u>, 51–54, 99, 100, 151, 383
pmix_persistence_t, 36, 36, 70, 112, 382, 419
pmix_proc_info_t, 33, 33, 34, 70, 86, 87, 188, 193, 311, 382
pmix proc state t, 32, 32, 70, 111, 382, 418
pmix_proc_t, 29, 30, 30–32, 53, 60, 69, 76, 83, 103, 107, 110, 118, 120, 122, 124, 132, 142–144,
          157, 223, 226, 227, 233, 234, 254, 265–268, 275, 280, 281, 283, 284, 287, 288, 291, 293,
         295, 299, 302, 307, 309, 313, 315, 318, 321, 323, 325, 327, 329, 330, 333, 355, 359, 366,
         369, 373, 382
pmix query t, 10, 11, 56, 56, 57, 70, 187, 192, 194, 309, 311, 384
pmix_rank_t, 29, 29-31, 70, 382
pmix_regattr_t, 13, 58, 58-60, 70, 95, 195, 272, 384
pmix scope t, 35, 35, 70, 112, 129, 382, 419
```

pmix\_status\_t, **22**, 22, 27, 41–43, 64, 65, 68, 69, 101–103, 105–109, 111, 221, 226, 304, 306, 307, 381, 393, 418
pmix\_value\_t, **38**, 38–43, 69, 100, 128, 129, 383

## **Index of Constants**

```
PMIX ALLOC DIRECTIVE, 70
PMIX ALLOC EXTEND, 49
PMIX ALLOC EXTERNAL, 49
PMIX ALLOC NEW, 49
PMIX ALLOC REAQUIRE, 49
PMIX_ALLOC_RELEASE, 49
PMIX APP, 69
PMIX_APP_WILDCARD, 21
PMIX BOOL, 69
PMIX_BUFFER, 69
PMIX_BYTE, 69
PMIX_BYTE_OBJECT, 69
PMIX_COMMAND, 70
PMIX_COMPRESSED_STRING, 70
PMIX_CONNECT_REQUESTED, 24
PMIX_COORD, 70
PMIX COORD LOGICAL VIEW, 338
PMIX_COORD_PHYSICAL_VIEW, 338
PMIX COORD VIEW UNDEF, 338
PMIX DATA ARRAY, 70
PMIX DATA RANGE, 70
PMIX DATA TYPE, 70
PMIX DOUBLE, 69
PMIX ENVAR, 70
PMIX ERR BAD PARAM, 23
PMIX ERR COMM FAILURE, 23
PMIX_ERR_DATA_VALUE_NOT_FOUND, 23
PMIX_ERR_DEBUGGER_RELEASE, 22
PMIX_ERR_DUPLICATE_KEY, 24
PMIX ERR EVENT REGISTRATION, 24
PMIX_ERR_GET_MALLOC_REQD, 25
PMIX_ERR_HANDSHAKE_FAILED, 22
PMIX_ERR_IN_ERRNO, 23
PMIX ERR INIT, 23
PMIX ERR INVALID ARG, 23
PMIX ERR INVALID ARGS, 23
PMIX ERR INVALID CRED, 22
PMIX ERR INVALID KEY, 23
```

```
PMIX_ERR_INVALID_KEY_LENGTH, 23
PMIX ERR INVALID KEYVALP, 23
PMIX ERR INVALID LENGTH, 23
PMIX ERR INVALID NAMESPACE, 23
PMIX ERR INVALID NUM ARGS, 23
PMIX_ERR_INVALID_NUM_PARSED, 23
PMIX ERR INVALID OPERATION, 24
PMIX ERR INVALID SIZE, 23
PMIX ERR INVALID TERMINATION, 24
PMIX_ERR_INVALID_VAL, 23
PMIX_ERR_INVALID_VAL_LENGTH, 23
PMIX_ERR_IOF_COMPLETE, 25
PMIX ERR IOF FAILURE, 25
PMIX_ERR_JOB_ABORTED, 26
PMIX_ERR_JOB_ABORTED_BY_SIG, 26
PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT, 26
PMIX_ERR_JOB_ALLOC_FAILED, 26
PMIX_ERR_JOB_APP_NOT_EXECUTABLE, 26
PMIX ERR JOB CANCELED, 26
PMIX ERR JOB FAILED TO LAUNCH, 26
PMIX_ERR_JOB_FAILED_TO_MAP, 26
PMIX ERR JOB KILLED BY CMD, 26
PMIX ERR JOB NON ZERO TERM, 26
PMIX ERR JOB SENSOR BOUND EXCEEDED, 26
PMIX_ERR_JOB_TERM_WO_SYNC, 26
PMIX ERR JOB TERMINATED, 24
PMIX_ERR_LOST_CONNECTION_TO_CLIENT, 23
PMIX ERR LOST CONNECTION TO SERVER, 23
PMIX_ERR_LOST_PEER_CONNECTION, 23
PMIX ERR NO EXE SPECIFIED, 26
PMIX_ERR_NO_PERMISSIONS, 23
PMIX_ERR_NODE_DOWN, 26
PMIX ERR NODE OFFLINE, 26
PMIX ERR NOMEM, 23
PMIX_ERR_NOT_FOUND, 23
PMIX_ERR_NOT_IMPLEMENTED, 23
PMIX ERR NOT SUPPORTED, 23
PMIX ERR OUT OF RESOURCE, 23
PMIX ERR PACK FAILURE, 23
PMIX ERR PACK MISMATCH, 23
PMIX ERR PARTIAL SUCCESS, 24
PMIX ERR PROC ABORTED, 22
PMIX ERR PROC ABORTING, 22
```

```
PMIX_ERR_PROC_CHECKPOINT, 22
PMIX ERR PROC ENTRY NOT FOUND, 22
PMIX ERR PROC MIGRATE, 22
PMIX ERR PROC REQUESTED ABORT, 22
PMIX ERR PROC RESTART, 22
PMIX_ERR_READY_FOR_HANDSHAKE, 22
PMIX ERR REPEAT ATTR REGISTRATION, 25
PMIX ERR RESOURCE BUSY, 23
PMIX ERR SERVER FAILED REQUEST, 22
PMIX_ERR_SERVER_NOT_AVAIL, 23
PMIX_ERR_SILENT, 22
PMIX_ERR_SYS_BASE, 26
PMIX ERR SYS OTHER, 26
PMIX_ERR_TIMEOUT, 23
PMIX_ERR_TYPE_MISMATCH, 22
PMIX_ERR_UNKNOWN_DATA_TYPE, 22
PMIX_ERR_UNPACK_FAILURE, 23
PMIX ERR UNPACK INADEQUATE SPACE, 22
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, 23
PMIX ERR UNREACH, 23
PMIX_ERR_UPDATE_ENDPOINTS, 24
PMIX ERR WOULD BLOCK, 22
PMIX ERROR, 22
PMIX EVENT ACTION COMPLETE, 26
PMIX_EVENT_ACTION_DEFERRED, 26
PMIX EVENT JOB END, 26
PMIX_EVENT_JOB_START, 25
PMIX EVENT NO ACTION TAKEN, 26
PMIX_EVENT_PARTIAL_ACTION_TAKEN, 26
PMIX EVENT_SESSION_END, 26
PMIX_EVENT_SESSION_START, 26
PMIX_EXISTS, 22
PMIX_EXTERNAL_ERR_BASE, 27
PMIX_FABRIC_COORDS_UPDATED, 335
PMIX_FABRIC_GET_DEVICE_INDEX, 339
PMIX_FABRIC_GET_VERTEX_INFO, 339
PMIX FABRIC REQUEST INFO, 339
PMIX_FABRIC_UPDATE_INFO, 339
PMIX FABRIC UPDATE PENDING, 335
PMIX FABRIC UPDATED, 335
PMIX FLOAT, 69
PMIX FWD ALL CHANNELS, 49
PMIX FWD NO CHANNELS, 49
```

```
PMIX_FWD_STDDIAG_CHANNEL, 49
PMIX_FWD_STDERR_CHANNEL, 49
PMIX FWD STDIN CHANNEL, 49
PMIX FWD STDOUT CHANNEL, 49
PMIX_GDS_ACTION_COMPLETE, 24
PMIX GLOBAL, 35
PMIX GROUP ACCEPT, 61, 354
PMIX GROUP CONSTRUCT, 354
PMIX GROUP CONSTRUCT ABORT, 25
PMIX_GROUP_CONSTRUCT_COMPLETE, 25
PMIX_GROUP_CONTEXT_ID_ASSIGNED, 25
PMIX_GROUP_DECLINE, 61, 354
PMIX GROUP DESTRUCT, 354
PMIX_GROUP_INVITE_ACCEPTED, 25
PMIX_GROUP_INVITE_DECLINED, 25
PMIX_GROUP_INVITE_FAILED, 25
PMIX_GROUP_INVITED, 24
PMIX GROUP LEADER FAILED, 25
PMIX_GROUP_LEADER_SELECTED, 25
PMIX GROUP LEFT, 25
PMIX_GROUP_MEMBER_FAILED, 25
PMIX GROUP MEMBERSHIP UPDATE, 25
PMIX INFO, 69
PMIX INFO ARRAY END, 47
PMIX_INFO_DIRECTIVES, 70
PMIX INFO REQD, 47
PMIX_INT, 69
PMIX INT16, 69
PMIX_INT32, 69
PMIX INT64, 69
PMIX_INT8, 69
PMIX_INTERNAL, 35
PMIX IOF CHANNEL, 70
PMIX_JCTRL_CHECKPOINT, 23
PMIX_JCTRL_CHECKPOINT_COMPLETE, 23
PMIX_JCTRL_PREEMPT_ALERT, 23
PMIX JOB STATE CONNECTED, 35
PMIX JOB STATE LAUNCH UNDERWAY, 35
PMIX JOB STATE PREPPED, 35
PMIX_JOB_STATE_RUNNING, 35
PMIX JOB STATE SUSPENDED, 35
PMIX JOB STATE TERMINATED, 36
PMIX JOB STATE TERMINATED WITH ERROR, 36
```

```
PMIX_JOB_STATE_UNDEF, 35
PMIX JOB STATE UNTERMINATED, 35
PMIX KVAL, 69
PMIX LAUNCH COMPLETE, 24
PMIX LAUNCH DIRECTIVE, 24
PMIX_LAUNCHER_READY, 24
PMIX LINK DOWN, 338
PMIX LINK STATE UNKNOWN, 338
PMIX LINK UP, 338
PMIX_LOCAL, 35
PMIX_MAX_KEYLEN, 21
PMIX_MAX_NSLEN, 21
PMIX MODEL DECLARED, 24
PMIX_MODEL_RESOURCES, 24
PMIX_MONITOR_FILE_ALERT, 24
PMIX_MONITOR_HEARTBEAT_ALERT, 24
PMIX_NOTIFY_ALLOC_COMPLETE, 23
PMIX_OPENMP_PARALLEL_ENTERED, 24
PMIX OPENMP PARALLEL EXITED, 24
PMIX_OPERATION_IN_PROGRESS, 24
PMIX_OPERATION_SUCCEEDED, 24
PMIX PDATA, 69
PMIX PERSIST, 70
PMIX PERSIST APP, 36
PMIX_PERSIST_FIRST_READ, 36
PMIX PERSIST INDEF, 36
PMIX_PERSIST_INVALID, 36
PMIX PERSIST PROC, 36
PMIX_PERSIST_SESSION, 36
PMIX_PID, 69
PMIX_POINTER, 70
PMIX_PROC, 69
PMIX_PROC_HAS_CONNECTED, 24
PMIX_PROC_INFO, 70
PMIX_PROC_RANK, 70
PMIX_PROC_STATE, 70
PMIX PROC STATE ABORTED, 33
PMIX PROC STATE ABORTED BY SIG, 33
PMIX PROC STATE CALLED ABORT, 33
PMIX_PROC_STATE_CANNOT_RESTART, 33
PMIX PROC STATE COMM FAILED, 33
PMIX PROC STATE CONNECTED, 32
PMIX PROC STATE ERROR, 32
```

```
PMIX_PROC_STATE_FAILED_TO_LAUNCH, 33
PMIX PROC STATE FAILED TO START, 33
PMIX PROC STATE KILLED BY CMD, 33
PMIX PROC STATE LAUNCH UNDERWAY, 32
PMIX PROC STATE MIGRATING, 33
PMIX_PROC_STATE_PREPPED, 32
PMIX PROC STATE RESTART, 32
PMIX PROC STATE RUNNING, 32
PMIX PROC STATE TERM NON ZERO, 33
PMIX_PROC_STATE_TERM_WO_SYNC, 33
PMIX_PROC_STATE_TERMINATE, 32
PMIX_PROC_STATE_TERMINATED, 32
PMIX PROC STATE UNDEF, 32
PMIX_PROC_STATE_UNTERMINATED, 32
PMIX_PROC_TERMINATED, 24
PMIX_QUERY, 70
PMIX_QUERY_PARTIAL_SUCCESS, 23
PMIX_RANGE_CUSTOM, 36
PMIX_RANGE_GLOBAL, 36
PMIX RANGE INVALID, 36
PMIX_RANGE_LOCAL, 36
PMIX RANGE NAMESPACE, 36
PMIX RANGE PROC LOCAL, 36
PMIX RANGE RM, 36
PMIX_RANGE_SESSION, 36
PMIX RANGE UNDEF, 36
PMIX_RANK_INVALID, 30
PMIX RANK LOCAL NODE, 29
PMIX_RANK_LOCAL_PEERS, 30
PMIX_RANK_UNDEF, 29
PMIX_RANK_VALID, 30
PMIX_RANK_WILDCARD, 29
PMIX_REGATTR, 70
PMIX_REGEX, 70
PMIX_REMOTE, 35
PMIX_SCOPE, 70
PMIX SCOPE UNDEF, 35
PMIX SIZE, 69
PMIX STATUS, 69
PMIX STRING, 69
PMIX SUCCESS, 22
PMIX TIME, 69
PMIX TIMEVAL, 69
```

PMIX\_UINT, 69 PMIX\_UINT16, 69 PMIX\_UINT32, 69 PMIX\_UINT64, 69 PMIX\_UINT8, 69 PMIX\_UNDEF, 69 PMIX\_VALUE, 69

## **Index of Attributes**

```
PMIX ADD ENVAR, 90
PMIX ADD HOST, 84, 160, 165, 296
PMIX ADD HOSTFILE, 84, 160, 165, 296
PMIX ALLOC BANDWIDTH, 91, 91, 198, 201, 270, 317
PMIX ALLOC CPU LIST, 90, 197, 200, 316
PMIX ALLOC FABRIC, 90, 90, 197, 200, 270, 316
PMIX ALLOC FABRIC ENDPTS, 91, 91, 197, 198, 200, 201, 270, 316
PMIX ALLOC FABRIC_ENDPTS_NODE, 91, 91, 198, 201, 270
PMIX ALLOC FABRIC ID, 91, 91, 197, 200, 270, 316
PMIX_ALLOC_FABRIC_PLANE, 91, 91, 198, 201, 270, 317
PMIX_ALLOC_FABRIC_QOS, 91, 91, 198, 201, 270, 317
PMIX_ALLOC_FABRIC_SEC_KEY, 91, 92, 92, 198, 201, 270, 317
PMIX_ALLOC_FABRIC_TYPE, 91, 91, 197, 198, 200, 201, 270, 316, 317
PMIX_ALLOC_ID, 12, 90, 199, 316
PMIX ALLOC MEM SIZE, 90, 197, 200, 316
PMIX_ALLOC_NETWORK (Deprecated), 90
PMIX ALLOC NETWORK ENDPTS (Deprecated), 91
PMIX ALLOC NETWORK ENDPTS NODE (Deprecated), 91
PMIX ALLOC NETWORK ID (Deprecated), 91
PMIX ALLOC NETWORK PLANE (Deprecated), 91
PMIX ALLOC NETWORK QOS (Deprecated), 91
PMIX ALLOC NETWORK SEC KEY (Deprecated), 92
PMIX_ALLOC_NETWORK_TYPE (Deprecated), 91
PMIX ALLOC NODE LIST, 90, 197, 200, 316
PMIX ALLOC NUM CPU LIST, 90, 197, 200, 316
PMIX ALLOC NUM CPUS, 90, 197, 200, 316
PMIX_ALLOC_NUM_NODES, 90, 197, 200, 316
PMIX_ALLOC_REQ_ID, 12, 90, 197, 200
PMIX_ALLOC_TIME, 91, 197, 200, 270, 316
PMIX ALLOCATED NODELIST, 76, 253
PMIX_ANL_MAP, 82, 254
PMIX_APP_INFO, 77, 131, 134, 138, 186, 191
PMIX_APP_INFO_ARRAY, 78, 78, 260
PMIX APP MAP REGEX, 82
PMIX APP MAP TYPE, 82
PMIX APP RANK, 75, 253
PMIX APP SIZE, 79, 138, 253, 259
PMIX APPEND ENVAR, 90
```

```
PMIX_APPLDR, 76, 253, 259
PMIX APPNUM, 75, 77, 78, 131, 134, 138, 186, 191, 252, 253, 260
PMIX ARCH, 75
PMIX ATTR UNDEF, 70
PMIX AVAIL PHYS MEMORY, 79, 254
PMIX_BINDTO, 85, 161, 165, 254, 297
PMIX CLEANUP EMPTY, 93, 204, 206
PMIX CLEANUP IGNORE, 93, 204, 206
PMIX CLEANUP LEAVE TOPDIR, 93, 204, 207
PMIX_CLEANUP_RECURSIVE, 93, 204, 206
PMIX_CLIENT_ATTRIBUTES, 13, 95, 186, 191
PMIX_CLIENT_AVG_MEMORY, 79
PMIX CLIENT FUNCTIONS, 95
PMIX_CLUSTER_ID, 75
PMIX_COLLECT_DATA, 80, 142, 144, 285
PMIX_COLLECTIVE_ALGO, 9, 81, 142, 145, 169, 172, 285, 300
PMIX_COLLECTIVE_ALGO_REQD, 142, 145, 169, 172, 285, 300
PMIX COLLECTIVE ALGO REOD (Deprecated), 81
PMIX_CONNECT_MAX_RETRIES, 72, 121
PMIX CONNECT RETRY DELAY, 72, 120
PMIX CONNECT SYSTEM FIRST, 72, 120, 122, 124
PMIX CONNECT TO SYSTEM, 71, 120, 122, 124
PMIX COSPAWN APP, 85
PMIX CPU LIST, 86, 162, 166, 298
PMIX_CPUS_PER_PROC, <u>86</u>, 161, 166, 298
PMIX CPUSET, 74
PMIX_CRED_TYPE, 94, 324
PMIX CREDENTIAL, 74
PMIX_CRYPTO_KEY, 94
PMIX DAEMON MEMORY, 79
PMIX_DATA_SCOPE, 81, 130, 134
PMIX_DEBUG_APP_DIRECTIVES, 89
PMIX DEBUG JOB, 89
PMIX_DEBUG_JOB_DIRECTIVES, 89
PMIX_DEBUG_STOP_IN_INIT, 89
PMIX_DEBUG_STOP_ON_EXEC, 89
PMIX DEBUG WAIT FOR NOTIFY, 89
PMIX DEBUG WAITING FOR NOTIFY, 89
PMIX DEBUGGER DAEMONS, 85, 161, 166, 297
PMIX DISPLAY MAP, 84, 160, 165, 297
PMIX DSTPATH, 72
PMIX EMBED BARRIER, 81, 119
PMIX ENUM VALUE, 13, 58, 95
```

```
PMIX_EVENT_ACTION_TIMEOUT, 83, 223
PMIX EVENT AFFECTED PROC, 83, 223, 227
PMIX EVENT AFFECTED PROCS, 83, 223, 227
PMIX EVENT BASE, 71, 118, 121, 127
PMIX EVENT CUSTOM RANGE, 83, 223, 227
PMIX_EVENT_DO_NOT_CACHE, 83
PMIX EVENT HDLR AFTER, 83, 222
PMIX EVENT HDLR APPEND, 83, 223
PMIX EVENT HDLR BEFORE, 83, 222
PMIX_EVENT_HDLR_FIRST, 82, 222
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, 82, 222
PMIX_EVENT_HDLR_LAST, 82, 222
PMIX EVENT HDLR LAST IN CATEGORY, 83, 222
PMIX_EVENT_HDLR_NAME, 82, 222
PMIX EVENT HDLR PREPEND, 83, 223
PMIX_EVENT_NO_TERMINATION, 84
PMIX EVENT_NON_DEFAULT, 83, 227
PMIX EVENT PROXY, 83
PMIX_EVENT_RETURN_OBJECT, 83, 223
PMIX EVENT SILENT TERMINATION, 83, 223
PMIX EVENT TERMINATE JOB, 83, 223
PMIX EVENT TERMINATE NODE, 83, 223
PMIX EVENT TERMINATE PROC, 83, 223
PMIX EVENT TERMINATE SESSION, 83, 223
PMIX EVENT TEXT MESSAGE, 83
PMIX EVENT WANT TERMINATION, 84
PMIX_EXIT_CODE, 76
PMIX FABRIC COORDINATE, 343
PMIX_FABRIC_COST_MATRIX, 340, 342
PMIX FABRIC DEVICE, 341, 344
PMIX_FABRIC_DEVICE_ADDRESS, 341, 344, 349
PMIX_FABRIC_DEVICE_BUS_TYPE, 342, 344, 348
PMIX FABRIC DEVICE DRIVER, 341, 344, 349
PMIX_FABRIC_DEVICE_FIRMWARE, 341, 344, 349
PMIX FABRIC DEVICE ID, 341, 344, 349
PMIX_FABRIC_DEVICE_INDEX, 334, 339, 344
PMIX FABRIC DEVICE MTU, 342, 344, 349
PMIX FABRIC DEVICE NAME, 341, 344, 348
PMIX FABRIC DEVICE PCI DEVID, 342, 344, 344, 348–350
PMIX FABRIC DEVICE SPEED, 342, 344, 349
PMIX FABRIC DEVICE STATE, 342, 344, 349
PMIX FABRIC DEVICE TYPE, 342, 344, 349
PMIX_FABRIC_DEVICE_VENDOR, 341, 344, 348
```

```
PMIX_FABRIC_DIMS, 340, 343
PMIX FABRIC ENDPT, 343
PMIX FABRIC GROUPS, 340, 342
PMIX FABRIC IDENTIFIER, 333, 340, 343, 346
PMIX FABRIC INDEX, 339, 343
PMIX_FABRIC_NUM_VERTICES, 340, 343
PMIX FABRIC PLANE, 333, 336, 341, 343, 346
PMIX FABRIC SHAPE, 341, 343
PMIX FABRIC SHAPE STRING, 341, 343
PMIX_FABRIC_VENDOR, 333, 340, 343, 346
PMIX_FABRIC_VIEW, 343
PMIX_FWD_STDDIAG, 11, 85
PMIX FWD STDERR, 85, 161, 166, 297, 312
PMIX_FWD_STDIN, 85, 161, 166, 297, 312
PMIX_FWD_STDOUT, 85, 161, 166, 297, 312
PMIX_GDS_MODULE, 74, 118, 121, 127
PMIX_GET_STATIC_VALUES, 81, 131, 134
PMIX GLOBAL RANK, 75, 254
PMIX_GROUP_ASSIGN_CONTEXT_ID, 96, 331, 332, 356, 360, 367, 370
PMIX GROUP CONTEXT ID, 96, 332
PMIX GROUP ENDPT DATA, 96, 331, 332
PMIX GROUP ID, 96, 332
PMIX GROUP INVITE DECLINE, 96
PMIX GROUP LEADER, 96, 356, 357, 360, 369, 374
PMIX_GROUP_LOCAL_ONLY, 96, 331, 356, 360
PMIX GROUP MEMBERSHIP, 96, 332, 357
PMIX_GROUP_NOTIFY_TERMINATION, 96, 356–358, 360, 363, 367, 370
PMIX GROUP OPTIONAL, 96, 331, 356, 358, 360, 366, 370
PMIX_GRPID, 72, 109, 146, 148, 150, 152, 154, 156, 187, 192, 197, 200, 203, 206, 209, 211, 213,
        216, 241, 242, 244, 246, 289–292, 294, 296, 305, 310, 312, 314, 316, 319, 321, 323, 326,
        327, 330
PMIX_HOST, 84, 160, 164, 296
PMIX_HOST_ATTRIBUTES, 13, 95, 187, 191, 195
PMIX HOST FUNCTIONS, 95
PMIX HOSTFILE, 84, 160, 164, 296
PMIX_HOSTNAME, 76, 76–78, 131, 134, 140, 186, 191, 252, 254, 341, 342, 344, 348–350
PMIX HWLOC HOLE KIND, 80
PMIX HWLOC SHARE TOPO, 80
PMIX HWLOC SHMEM ADDR, 80
PMIX_HWLOC_SHMEM_FILE, 80
PMIX HWLOC SHMEM SIZE, 80
PMIX HWLOC XML V1, 80, 254
PMIX HWLOC XML V2, 80, 254
```

```
PMIX_IMMEDIATE, 80, 130, 134
PMIX INDEX ARGV, 85, 161, 166, 298
PMIX IOF BUFFERING SIZE, 94, 178, 182, 328
PMIX IOF BUFFERING TIME, 94, 178, 182, 328
PMIX IOF CACHE SIZE, 94, 178, 182, 328
PMIX IOF COMPLETE, 94, 110, 386
PMIX IOF DROP NEWEST, 94, 178, 182, 328
PMIX IOF DROP OLDEST, 94, 178, 182, 328
PMIX IOF TAG OUTPUT, 94, 178
PMIX_IOF_TIMESTAMP_OUTPUT, 94, 178
PMIX_IOF_XML_OUTPUT, 94, 179
PMIX_JOB_CONTINUOUS, 86, 162, 166, 298
PMIX JOB CTRL CANCEL, 92, 204, 207, 319
PMIX_JOB_CTRL_CHECKPOINT, 92, 204, 207, 319
PMIX_JOB_CTRL_CHECKPOINT_EVENT, 92, 204, 207, 319
PMIX_JOB_CTRL_CHECKPOINT_METHOD, 92, 204, 207, 320
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL, 92, 204, 207, 319
PMIX JOB CTRL CHECKPOINT TIMEOUT, 92, 204, 207, 319
PMIX JOB CTRL ID, 92, 92, 203, 204, 206, 207, 319
PMIX_JOB_CTRL_KILL, 92, 203, 206, 319
PMIX JOB CTRL PAUSE, 92, 203, 206, 319
PMIX JOB CTRL PREEMPTIBLE, 92, 204, 207, 320
PMIX JOB CTRL PROVISION, 92, 204, 207, 320
PMIX JOB CTRL PROVISION IMAGE, 92, 204, 207, 320
PMIX JOB CTRL RESTART, 92, 204, 207, 319
PMIX JOB CTRL RESUME, 92, 203, 206, 319
PMIX_JOB_CTRL_SIGNAL, 92, 203, 206, 319
PMIX JOB CTRL TERMINATE, 93, 203, 206, 319
PMIX_JOB_INFO, 77, 130, 134, 138, 186, 191
PMIX JOB INFO ARRAY, 10, 77, 78, 259
PMIX_JOB_NUM_APPS, 79, 138, 253, 259
PMIX_JOB_RECOVERABLE, 86, 162, 166, 298
PMIX JOB SIZE, 10, 12, 78, 132, 135, 138, 251, 259
PMIX_JOB_TERM_STATUS, 81
PMIX_JOBID, 75, 77, 78, 131, 134, 138, 186, 191, 251, 259
PMIX_LAUNCHER, 72
PMIX LOCAL CPUSETS, 76, 252, 262
PMIX LOCAL_PEERS, <u>76</u>, 252, 262
PMIX LOCAL PROCS, 76, 254
PMIX_LOCAL_RANK, <u>75</u>, 186, 187, 191, 192, 252
PMIX LOCAL SIZE, 79, 252
PMIX LOCAL TOPO, 79
PMIX LOCALITY, 76
```

```
PMIX_LOCALITY_STRING, 80
PMIX LOCALLDR, 76, 254
PMIX LOG EMAIL, 88, 215, 218, 314
PMIX LOG EMAIL ADDR, 88, 215, 218, 314
PMIX_LOG_EMAIL_MSG, 88, 215, 218, 314
PMIX_LOG_EMAIL_SENDER_ADDR, 88
PMIX LOG EMAIL SERVER, 88
PMIX LOG EMAIL SRVR PORT, 88
PMIX LOG EMAIL SUBJECT, 88, 215, 218, 314
PMIX_LOG_GENERATE_TIMESTAMP, 88, 214, 217
PMIX_LOG_GLOBAL_DATASTORE, 88, 215, 218
PMIX_LOG_GLOBAL_SYSLOG, 88, 214, 217
PMIX LOG JOB RECORD, 88, 215, 218
PMIX_LOG_LOCAL_SYSLOG, 87, 214, 217
PMIX_LOG_MSG, 88, 314
PMIX_LOG_ONCE, 88, 214, 217
PMIX_LOG_SOURCE, 87, 214, 217
PMIX_LOG_STDERR, 87, 214, 217, 314
PMIX_LOG_STDOUT, 87, 214, 217, 314
PMIX LOG SYSLOG, 87, 214, 217, 314
PMIX_LOG_SYSLOG_PRI, 88, 214, 217
PMIX LOG TAG OUTPUT, 88, 214, 217
PMIX LOG TIMESTAMP, 88, 214, 217
PMIX LOG TIMESTAMP OUTPUT, 88, 214, 217
PMIX LOG XML OUTPUT, 88, 214, 217
PMIX MAP BLOB, 82
PMIX_MAPBY, 84, 160, 165, 253, 297
PMIX MAPPER, 84, 84, 160, 165, 297
PMIX_MAX_PROCS, 12, 58, 78, 79, 79, 140, 251, 252
PMIX_MAX_RESTARTS, 86, 162, 167, 298
PMIX_MAX_VALUE, 13, 58, 95
PMIX_MERGE_STDERR_STDOUT, 85, 161, 166, 298
PMIX MIN VALUE, 13, 58, 95
PMIX_MODEL_AFFINITY_POLICY, 73
PMIX_MODEL_CPU_TYPE, 73
PMIX_MODEL_LIBRARY_NAME, 73, 255, 271
PMIX MODEL LIBRARY VERSION, 73, 255, 271
PMIX MODEL NUM CPUS, 73
PMIX MODEL NUM THREADS, 73
PMIX_MODEL_PHASE_NAME, 73
PMIX MODEL PHASE TYPE, 73
PMIX MONITOR APP CONTROL, 93, 209, 211, 322
PMIX MONITOR CANCEL, 93, 209, 211, 322
```

```
PMIX_MONITOR_FILE, 93, 209, 210, 212, 322
PMIX MONITOR FILE ACCESS, 93, 209, 212, 322
PMIX MONITOR FILE CHECK TIME, 94, 210, 212, 322
PMIX MONITOR FILE DROPS, 94, 210, 212, 322
PMIX MONITOR FILE MODIFY, 93, 210, 212, 322
PMIX MONITOR FILE SIZE, 93, 209, 212, 322
PMIX MONITOR HEARTBEAT, 93, 209, 211, 322
PMIX MONITOR HEARTBEAT DROPS, 93, 209, 212, 322
PMIX MONITOR HEARTBEAT TIME, 93, 209, 211, 322
PMIX_MONITOR_ID, 93, 209, 211, 322
PMIX_NO_OVERSUBSCRIBE, 86, 162, 166, 298
PMIX_NO_PROCS_ON_HEAD, 86, 162, 166, 298
PMIX NODE INFO, 77, 131, 134, 140, 186, 191
PMIX_NODE_INFO_ARRAY, 78, 78, 259, 261
PMIX NODE LIST, 76
PMIX_NODE_MAP, 12, 82, 252, 258–260, 271
PMIX_NODE_RANK, 75, 252
PMIX NODE SIZE, 79, 140, 254
PMIX NODEID, 76, 76–78, 131, 134, 140, 186, 191, 252, 342, 344, 348–350
PMIX NON PMI, 85, 161, 165, 297
PMIX NOTIFY COMPLETION, 81, 162
PMIX NPROC OFFSET, 75, 253
PMIX NSDIR, 75, 75
PMIX NSPACE, 75, 77, 78, 131, 134, 138, 186, 187, 191, 192, 259
PMIX_NUM_NODES, <u>79</u>, 132, 135, 136, 138, 258, 259
PMIX NUM SLOTS, 79
PMIX_OPTIONAL, 81, 130, 133
PMIX OUTPUT TO FILE, 85, 161, 166, 298
PMIX_PARENT_ID, 76, 159, 164, 296
PMIX_PERSISTENCE, 81, 147, 149, 289
PMIX_PERSONALITY, 84, 160, 165, 297
PMIX_PPR, 84, 160, 165, 297
PMIX_PREFIX, 84, 160, 164, 296
PMIX_PRELOAD_BIN, 85, 160, 165, 296
PMIX_PRELOAD_FILES, 85, 160, 165, 296
PMIX PREPEND ENVAR, 90
PMIX PROC BLOB, 82
PMIX PROC DATA, 82, 260
PMIX PROC MAP, 12, 82, 252, 258, 259, 271
PMIX PROC PID, 76
PMIX PROC STATE STATUS, 81
PMIX PROC TERM STATUS, 81
PMIX PROC URI, 76, 189, 193
```

```
PMIX_PROCDIR, 75
PMIX PROCID, 75, 186, 187, 191, 192, 254
PMIX PROGRAMMING MODEL, 73, 255, 271
PMIX PSET NAME, 72, 351
PMIX_QUERY_ALLOC_STATUS, 87, 188, 193, 311
PMIX_QUERY_ATTRIBUTE_SUPPORT, <u>87</u>, 186, 191, 194
PMIX QUERY AUTHORIZATIONS, 87
PMIX_QUERY_DEBUG_SUPPORT, <u>87</u>, 188, 193, 311
PMIX QUERY JOB STATUS, 86, 188, 193, 310
PMIX_QUERY_LOCAL_ONLY, 87, 311
PMIX_QUERY_LOCAL_PROC_TABLE, 86, 188, 193, 311
PMIX_QUERY_MEMORY_USAGE, 87, 188, 193, 311
PMIX OUERY NAMESPACES, 86, 188, 192, 310
PMIX_QUERY_NUM_PSETS, 87
PMIX_QUERY_PROC_TABLE, 86, 188, 193, 311
PMIX_QUERY_PSET_NAMES, 87
PMIX_QUERY_QUEUE_LIST, 86, 188, 193, 310
PMIX_QUERY_QUEUE_STATUS, 86, 188, 193, 310
PMIX_QUERY_REFRESH_CACHE, 86, 185, 189, 190, 194
PMIX_QUERY_REPORT_AVG, 87, 188, 193, 311
PMIX_QUERY_REPORT_MINMAX, 87, 188, 193, 311
PMIX QUERY SPAWN SUPPORT, 87, 188, 193, 311
PMIX RANGE, 81, 147, 149, 150, 153, 154, 156, 210, 223, 289, 291, 294, 308, 332
PMIX RANK, 75, 186, 187, 191, 192, 252
PMIX_RANKBY, 84, 161, 165, 253, 297
PMIX RECONNECT SERVER, 72
PMIX_REGISTER_CLEANUP, 93, 203, 206
PMIX_REGISTER_CLEANUP_DIR, 93, 203, 206
PMIX_REGISTER_NODATA, 82, 251
PMIX REPORT BINDINGS, 86, 162, 166, 298
PMIX_REQUESTOR_IS_CLIENT, 72, 159, 164
PMIX_REQUESTOR_IS_TOOL, 72, 159, 164
PMIX RM NAME, 89
PMIX_RM_VERSION, 89
PMIX_SEND_HEARTBEAT, 93
PMIX_SERVER_ATTRIBUTES, 13, 95, 186, 191
PMIX SERVER ENABLE MONITORING, 71
PMIX SERVER FUNCTIONS, 95
PMIX SERVER GATEWAY, 71
PMIX_SERVER_HOSTNAME, 72
PMIX SERVER NSPACE, 71, 124, 125, 253
PMIX SERVER PIDINFO, 71, 120, 122, 124
PMIX SERVER RANK, 71, 125, 253
```

```
PMIX_SERVER_REMOTE_CONNECTIONS, 71, 126
PMIX SERVER SCHEDULER, 342, 345
PMIX SERVER SYSTEM SUPPORT, 71, 125
PMIX SERVER TMPDIR, 71, 125
PMIX SERVER TOOL SUPPORT, 71, 125, 127, 239
PMIX_SERVER_URI, 72, 120, 122, 124, 188, 193
PMIX SESSION ID, 76, 77, 130, 134, 137, 186, 190, 253, 258, 259
PMIX SESSION INFO, 77, 130, 134, 136, 185, 190
PMIX SESSION INFO ARRAY, 10, 77, 78, 251, 258
PMIX_SET_ENVAR, 90
PMIX_SET_SESSION_CWD, 85, 160, 164, 296
PMIX_SETUP_APP_ALL, 95, 269
PMIX SETUP APP ENVARS, 95, 269
PMIX_SETUP_APP_NONENVARS, 95, 269
PMIX SINGLE LISTENER, 73, 117
PMIX_SOCKET_MODE, 73, 117, 121, 126
PMIX_SPAWN_TOOL, 86
PMIX SPAWNED, 74, 159, 164, 296
PMIX_STDIN_TGT, 85, 161, 165, 297
PMIX SYSTEM TMPDIR, 71, 125
PMIX TAG OUTPUT, 85, 161, 166, 297
PMIX TCP DISABLE IPV4, 74, 118, 121, 126
PMIX TCP DISABLE IPV6, 74, 118, 121, 126
PMIX TCP IF EXCLUDE, 74, 117, 121, 126
PMIX TCP IF INCLUDE, 74, 117, 121, 126
PMIX TCP IPV4 PORT, 74, 118, 121, 126
PMIX_TCP_IPV6_PORT, 74, 118, 121, 126
PMIX TCP REPORT URI, 74, 117, 121, 126
PMIX_TCP_URI, 74, 120, 122
PMIX TDIR RMCLEAN, 75
PMIX_THREADING_MODEL, 73
PMIX_TIME_REMAINING, 87, 183, 188, 193, 311
PMIX_TIMEOUT, 3, 14, 80, 131, 132, 134, 135, 142, 143, 145, 147, 149–151, 153, 154, 156, 169,
        172, 173, 175, 176, 241, 243, 245, 247, 285, 288, 289, 292, 294, 298, 300, 303, 324, 326,
        354, 356–358, 361–363, 365, 367, 371, 373, 375, 376
PMIX_TIMESTAMP_OUTPUT, 85, 161, 166, 298
PMIX TMPDIR, 75, 75
PMIX TOOL ATTRIBUTES, 13, 95, 187, 192
PMIX TOOL DO NOT CONNECT, 72, 120, 122
PMIX TOOL FUNCTIONS, 95
PMIX TOOL NSPACE, 71, 120
PMIX TOOL RANK, 71, 120
PMIX TOPOLOGY, 79
```

```
PMIX_TOPOLOGY_FILE, 79
PMIX_TOPOLOGY_SIGNATURE, 80
PMIX_TOPOLOGY_XML, 79
PMIX_UNIV_SIZE, 10, 12, 78, 132, 135, 136, 251, 258
PMIX_UNSET_ENVAR, 90
PMIX_USERID, 72, 109, 146, 148, 150, 152, 154, 156, 187, 192, 197, 200, 203, 206, 209, 211, 213, 216, 241, 242, 244, 246, 289–294, 296, 304, 310, 312, 314, 316, 318, 321, 323, 325, 327, 330
PMIX_USOCK_DISABLE, 73, 117, 126
PMIX_VERSION_INFO, 72
PMIX_WAIT, 80, 150, 151, 153, 291
PMIX_WOIR, 84, 159, 164, 296
```