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

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

Copyright © 2018-2020 PMIx Administrative Steering Committee (ASC). Permission to copy without fee all or part of this material is granted, provided the PMIx ASC copyright notice and the title of this document appear, and notice is given that copying is by permission of PMIx ASC.
# Contents

1. **Introduction** .......................................................... 1
   1.1. Background ...................................................... 1
   1.2. PMIx Architecture Overview .................................... 1
   1.3. Portability of Functionality ..................................... 3
       1.3.1. Attributes in PMIx ........................................ 3

2. **PMIx Terms and Conventions** .................................. 6
   2.1. Notational Conventions ......................................... 8
   2.2. Semantics ........................................................ 9
   2.3. Naming Conventions ............................................. 10
   2.4. Procedure Conventions .......................................... 10

3. **Data Structures and Types** ................................ 12
   3.1. Constants ........................................................ 13
       3.1.1. PMIx Return Status Constants ............................ 14
           3.1.1.1. User-Defined Error and Event Constants ............ 15
   3.2. Data Types ........................................................ 16
       3.2.1. Key Structure ............................................. 16
           3.2.1.1. Key support macros ................................ 16
       3.2.2. Namespace Structure ..................................... 18
           3.2.2.1. Namespace support macros ......................... 18
       3.2.3. Rank Structure ............................................ 19
           3.2.3.1. Rank support macros ............................... 19
       3.2.4. Process Structure ......................................... 20
           3.2.4.1. Process structure support macros ................. 20
       3.2.5. Process State Structure .................................. 23
       3.2.6. Process Information Structure ............................ 24
           3.2.6.1. Process information structure support macros ....... 25
       3.2.7. Job State Structure ....................................... 27
3.2.8. Value Structure .......................................................... 27
  3.2.8.1. Value structure support macros ................................ 28
3.2.9. Info Structure ............................................................. 32
  3.2.9.1. Info structure support macros .................................. 32
  3.2.9.2. Info structure list macros ....................................... 35
3.2.10. Info Type Directives .................................................. 37
  3.2.10.1. Info Directive support macros ................................. 38
3.2.11. Environmental Variable Structure ................................. 41
  3.2.11.1. Environmental variable support macros .................... 41
3.2.12. Byte Object Type ..................................................... 43
  3.2.12.1. Byte object support macros ................................... 43
3.2.13. Data Array Structure ................................................ 44
  3.2.13.1. Data array support macros ................................... 44
3.2.14. Argument Array Macros ............................................ 46
3.2.15. Set Environment Variable .......................................... 50
3.3. Generalized Data Types Used for Packing/Unpacking ............... 50
3.4. General Callback Functions ............................................ 52
  3.4.1. Release Callback Function ........................................ 52
  3.4.2. Op Callback Function .............................................. 53
  3.4.3. Value Callback Function .......................................... 53
  3.4.4. Info Callback Function ............................................ 54
  3.4.5. Handler registration callback function .......................... 55
3.5. PMIx Datatype Value String Representations ........................ 55

4. Client Initialization and Finalization .................................. 59
  4.1. PMIx_Initiated ......................................................... 59
  4.2. PMIx_Get_version ..................................................... 60
  4.3. PMIx_Init .............................................................. 60
    4.3.1. Initialization events ........................................... 63
    4.3.2. Initialization attributes ....................................... 63
      4.3.2.1. Connection attributes .................................... 63
      4.3.2.2. Programming model attributes ............................ 64
  4.4. PMIx_Finalize .......................................................... 65
    4.4.1. Finalize attributes ............................................ 65
4.5. **PMIx_Progress** ........................................... 65

5. **Synchronization and Data Access Operations** 67
   5.1. **PMIx_Fence** ........................................... 67
   5.2. **PMIx_Fence_nb** ....................................... 69
       5.2.1. Fence-related attributes ......................... 71
   5.3. **PMIx_Get** ............................................. 72
       5.3.1. **PMIx_Get_nb** .................................... 74
       5.3.2. Retrieval attributes ............................. 77
   5.4. **Query** .................................................. 78
       5.4.1. **PMIx.Resolve_peers** ......................... 79
       5.4.2. **PMIx.Resolve_nodes** ......................... 79
       5.4.3. **PMIx_Query.info** .............................. 80
       5.4.4. **PMIx_Query.info_nb** ......................... 85
       5.4.5. Query-specific constants ....................... 89
       5.4.6. Query attributes .................................. 90
       5.4.7. Query Structure ................................... 92
           5.4.7.1. Query structure support macros ........ 93
   5.5. Using **Get** vs **Query** ............................ 95
   5.6. Accessing attribute support information ........... 95

6. **Reserved Keys** 98
   6.1. Data realms ........................................... 98
       6.1.1. Session realm attributes ....................... 99
       6.1.2. Job realm attributes ............................ 101
       6.1.3. Application realm attributes .................. 103
       6.1.4. Process realm attributes ...................... 104
       6.1.5. Node realm keys .................................. 106
   6.2. Retrieval rules for reserved keys .................... 107
       6.2.1. Accessing information: examples ............... 107
           6.2.1.1. Session-level information ............... 108
           6.2.1.2. Job-level information ..................... 109
           6.2.1.3. Application-level information .......... 109
           6.2.1.4. Process-level information ............... 110
6.2.1.5. Node-level information .................................. 110

7. Process-Related Non-Reserved Keys .................................................. 112
  7.1. Posting Key/Value Pairs ......................................................... 113
    7.1.1. PMIx_Put .............................................................. 113
      7.1.1.1. Scope of Put Data ............................................. 114
    7.1.2. PMIx_Store_internal .................................................... 114
    7.1.3. PMIx_Commit ........................................................... 115
  7.2. Retrieval rules for non-reserved keys ....................................... 116

8. Publish/Lookup Operations .............................................................. 118
  8.1. PMIx_Publish ............................................................... 118
  8.2. PMIx_Publish_nb ............................................................ 120
  8.3. Publish-specific constants .................................................. 121
  8.4. Publish-specific attributes ................................................ 121
  8.5. Publish-Lookup Datatypes .................................................... 122
    8.5.1. Range of Published Data ............................................ 122
    8.5.2. Data Persistence Structure ........................................ 122
  8.6. PMIx_Lookup ................................................................. 123
  8.7. PMIx_Lookup_nb ............................................................. 125
    8.7.1. Lookup Returned Data Structure ................................... 127
      8.7.1.1. Lookup data structure support macros ........................ 127
    8.7.2. Lookup Callback Function .......................................... 130
  8.8. Retrieval rules for published data ......................................... 130
  8.9. PMIx_Unpublish ............................................................. 131
  8.10. PMIx_Unpublish_nb ........................................................ 133

9. Event Notification ................................................................. 135
  9.1. Notification and Management ................................................ 135
    9.1.1. Events versus status constants .................................... 137
    9.1.2. PMIx_Register_event_handler ...................................... 137
    9.1.3. Event registration constants ....................................... 140
    9.1.4. System events .......................................................... 141
    9.1.5. Event handler registration and notification attributes ............ 141
      9.1.5.1. Fault tolerance event attributes ............................. 142
9.1.5.2. Hybrid programming event attributes ........................................ 142
9.1.6. Notification Function ........................................................................ 143
9.1.7. PMIx_Deregister_event_handler ......................................................... 144
9.1.8. PMIx_Notify_event ............................................................................ 145
9.1.9. Notification Handler Completion Callback Function ......................... 149
9.1.9.1. Completion Callback Function Status Codes .................................. 149

10. Data Packing and Unpacking ................................................................. 150
10.1. Data Buffer Type ................................................................................. 150
10.2. Support Macros ................................................................................... 151
10.3. General Routines ................................................................................ 152
10.3.1. PMIx_Data_pack .............................................................................. 152
10.3.2. PMIx_Data_unpack .......................................................................... 154
10.3.3. PMIx_Data_copy .............................................................................. 156
10.3.4. PMIx_Data_print ............................................................................. 156
10.3.5. PMIx_Data_copy_payload ................................................................. 157

11. Process Management .............................................................................. 159
11.1. Abort .................................................................................................. 159
11.1.1. PMIx_Abort ..................................................................................... 159
11.2. Process Creation ................................................................................. 160
11.2.1. PMIx_Spawn ................................................................................... 160
11.2.2. PMIx_Spawn_nb ............................................................................ 166
11.2.3. Spawn-specific constants ................................................................. 171
11.2.4. Spawn attributes ............................................................................ 171
11.2.5. Application Structure .................................................................... 175
11.2.5.1. App structure support macros ...................................................... 175
11.2.5.2. Spawn Callback Function ............................................................. 177
11.3. Connecting and Disconnecting Processes ......................................... 178
11.3.1. PMIx_Connect ............................................................................... 179
11.3.2. PMIx_Connect_nb .......................................................................... 181
11.3.3. PMIx_Disconnect .......................................................................... 182
11.3.4. PMIx_Disconnect_nb ..................................................................... 184
11.4. Process Locality ........................................... 186
  11.4.1. PMIx_Load_topology .................................. 186
  11.4.2. PMIx_Get_relative_locality ......................... 187
    11.4.2.1. Topology description .......................... 187
    11.4.2.2. Topology support macros ....................... 188
    11.4.2.3. Relative locality of two processes ............. 189
    11.4.2.4. Locality keys .................................. 189
  11.4.3. PMIx_Parse_cpuset_string ......................... 189
  11.4.4. PMIx_Get_cpuset ................................... 190
    11.4.4.1. Binding envelope .............................. 190
  11.4.5. PMIx_Compute_distances ......................... 191
  11.4.6. PMIx_Compute_distances_nb ....................... 192
  11.4.7. Device Distance Callback Function ............... 193
  11.4.8. Device type ....................................... 193
  11.4.9. Device Distance Structure ....................... 194
  11.4.10. Device distance support macros ................. 195
  11.4.11. Device distance attributes ..................... 196

12. Job Management and Reporting ............................. 197
  12.1. Allocation Requests ................................. 197
    12.1.1. PMIx_Allocation_request ......................... 197
    12.1.2. PMIx_Allocation_request_nb .................... 200
    12.1.3. Job Allocation attributes ...................... 203
    12.1.4. Job Allocation Directives ...................... 205
  12.2. Job Control ........................................... 205
    12.2.1. PMIx_Job_control ................................ 206
    12.2.2. PMIx_Job_control_nb ............................. 208
    12.2.3. Job control constants ........................... 211
    12.2.4. Job control events .............................. 212
    12.2.5. Job control attributes ......................... 212
  12.3. Process and Job Monitoring .......................... 213
    12.3.1. PMIx_Process_monitor ............................ 213
    12.3.2. PMIx_Process_monitor_nb ....................... 216
    12.3.3. PMIx_Heartbeat .................................. 218
12.3.4. Monitoring events ............................................. 218
12.3.5. Monitoring attributes ........................................... 218
12.4. Logging ............................................................... 219
  12.4.1. PMIx_Log .................................................. 219
  12.4.2. PMIx_Log_nb ............................................... 222
  12.4.3. Log attributes ............................................. 225

13. Process Sets and Groups ............................................. 227
  13.1. Process Sets ................................................... 227
    13.1.1. Process Set Constants .................................... 228
    13.1.2. Process Set Attributes ................................... 229
  13.2. Process Groups ................................................ 229
    13.2.1. Relation to the host environment ...................... 229
    13.2.2. Construction procedure .................................. 230
    13.2.3. Destruct procedure ...................................... 231
    13.2.4. Process Group Events ..................................... 231
    13.2.5. Process Group Attributes ................................ 232
    13.2.6. PMIx_Group_construct .................................... 234
    13.2.7. PMIx_Group_construct_nb ................................. 237
    13.2.8. PMIx_Group_destruct ..................................... 240
    13.2.9. PMIx_Group_destruct_nb ................................ 241
    13.2.10. PMIx_Group_invite ....................................... 243
    13.2.11. PMIx_Group_invite_nb .................................. 246
    13.2.12. PMIx_Group_join ......................................... 249
    13.2.13. PMIx_Group_join_nb ..................................... 251
    13.2.13.1. Group accept/decline directives ................... 252
    13.2.14. PMIx_Group_leave ....................................... 252
    13.2.15. PMIx_Group_leave_nb .................................. 254

14. Fabric Support Definitions ......................................... 256
  14.1. Fabric Support Events ........................................ 259
  14.2. Fabric Support Datatypes ..................................... 259
    14.2.1. Fabric Endpoint Structure ............................... 259
    14.2.2. Fabric endpoint support macros ...................... 260
14.2.3. Fabric Coordinate Structure .................................................. 261
14.2.4. Fabric coordinate support macros ............................................. 261
14.2.5. Fabric Geometry Structure ...................................................... 263
14.2.6. Fabric geometry support macros ................................................. 263
14.2.7. Fabric Coordinate Views ........................................................ 265
14.2.8. Fabric Link State ................................................................. 265
14.2.9. Fabric Operation Constants ...................................................... 266
14.2.10. Fabric registration structure ................................................... 266
14.2.10.1. Initialize the fabric structure ............................................... 269
14.3. Fabric Support Attributes .......................................................... 269
14.4. Fabric Support Functions ........................................................... 272
14.4.1. PMIx_Fabric_register ............................................................ 273
14.4.2. PMIx_Fabric_register_nb ......................................................... 275
14.4.3. PMIx_Fabric_update .............................................................. 275
14.4.4. PMIx_Fabric_update_nb .......................................................... 276
14.4.5. PMIx_Fabric_deregister .......................................................... 277
14.4.6. PMIx_Fabric_deregister_nb ....................................................... 277

15. Security ............................................................................. 279
15.1. Obtaining Credentials ............................................................. 279
15.1.1. PMIx_Get_credential ............................................................. 280
15.1.2. PMIx_Get_credential_nb ......................................................... 281
15.1.3. Credential Attributes ............................................................ 282
15.2. Validating Credentials ............................................................ 283
15.2.1. PMIxValidate_credential ......................................................... 283
15.2.2. PMIxValidate_credential_nb .................................................... 284

16. Server-Specific Interfaces ......................................................... 287
16.1. Server Initialization and Finalization ........................................... 287
16.1.1. PMIx_server_init ................................................................. 287
16.1.2. PMIx_server_finalize ............................................................ 291
16.1.3. Server Initialization Attributes ............................................ 291
16.2. Server Support Functions ........................................................ 292
16.2.1. PMIx_generate_regex ......................................................... 292
16.2.2. PMIx_generate_ppn .............................................. 294
16.2.3. PMIx_server_register_nspace ....................................... 294
  16.2.3.1. Namespace registration attributes .......................... 305
  16.2.3.2. Assembling the registration information .................. 306
16.2.4. PMIx_server_deregister_nspace .................................. 315
16.2.5. PMIx_server_register_resources .................................. 315
16.2.6. PMIx_server_deregister_resources ................................. 316
16.2.7. PMIx_server_register_client ..................................... 317
16.2.8. PMIx_server_deregister_client .................................... 319
16.2.9. PMIx_server_setup_fork ......................................... 319
16.2.10. PMIx_server_dmodex_request ................................... 320
16.2.11. PMIx_server_setup_application .................................. 322
  16.2.11.1. Server Setup Application Callback Function .............. 325
  16.2.11.2. Server Setup Application Attributes ...................... 326
16.2.12. PMIx_Register_attributes ...................................... 326
  16.2.12.1. Attribute registration constants ........................ 328
  16.2.12.2. Attribute registration structure ........................ 328
  16.2.12.3. Attribute registration structure descriptive attributes .... 329
  16.2.12.4. Attribute registration structure support macros ....... 329
16.2.13. PMIx_server_setup_local_support ............................... 331
16.2.14. PMIx_server_IOF_deliver ..................................... 333
16.2.15. PMIx_server_collect_inventory .................................. 334
16.2.16. PMIx_server_deliver_inventory ................................ 335
16.2.17. PMIx_server_generate_locality_string ......................... 336
16.2.18. PMIx_server_generate_cpuset_string ............................. 337
  16.2.18.1. Cpuset Structure ...................................... 338
  16.2.18.2. Cpuset support macros .................................. 338
16.2.19. PMIx_server_define_process_set ................................ 339
16.2.20. PMIx_server_delete_process_set ................................ 340
16.3. Server Function Pointers ............................................. 340
  16.3.1. pmix_server_module_t Module .................................. 341
  16.3.2. pmix_server_client_connected_fn_t .......................... 342
16.3.3. `pmix_server_client_connected2_fn_t` .......................... 343
16.3.4. `pmix_server_client_finalized_fn_t` .......................... 345
16.3.5. `pmix_server_abort_fn_t` ........................................ 346
16.3.6. `pmix_server_fencenb_fn_t` ..................................... 348
   16.3.6.1. Modex Callback Function ................................. 351
16.3.7. `pmix_server_dmodex_req_fn_t` ............................... 351
   16.3.7.1. Dmodex attributes ........................................ 353
16.3.8. `pmix_server_publish_fn_t` .................................... 353
16.3.9. `pmix_server_lookup_fn_t` .................................... 355
16.3.10. `pmix_server_unpublish_fn_t` ............................... 358
16.3.11. `pmix_server_spawn_fn_t` .................................... 360
   16.3.11.1. Server spawn attributes ................................. 365
16.3.12. `pmix_server_connect_fn_t` .................................. 365
16.3.13. `pmix_server_disconnect_fn_t` .............................. 366
16.3.14. `pmix_server_register_events_fn_t` ....................... 368
16.3.15. `pmix_server_deregister_events_fn_t` .................... 370
16.3.16. `pmix_server_notify_event_fn_t` ........................... 372
16.3.17. `pmix_server_listener_fn_t` ................................ 373
   16.3.17.1. PMIx Client Connection Callback Function .......... 374
16.3.18. `pmix_server_query_fn_t` .................................... 375
16.3.19. `pmix_server_tool_connection_fn_t` ....................... 377
   16.3.19.1. Tool connection attributes ............................ 380
   16.3.19.2. PMIx Tool Connection Callback Function ............ 380
16.3.20. `pmix_server_log_fn_t` ..................................... 380
16.3.21. `pmix_server_alloc_fn_t` ................................... 382
16.3.22. `pmix_server_job_control_fn_t` ............................ 385
16.3.23. `pmix_server_monitor_fn_t` .................................. 388
16.3.24. `pmix_server_get_cred_fn_t` ............................... 391
   16.3.24.1. Credential callback function .......................... 392
16.3.25. `pmix_server_validate_cred_fn_t` .......................... 393
16.3.26. Credential validation callback function .................. 395
16.3.27. `pmix_server_iof_fn_t` ..................................... 396
   16.3.27.1. IOF delivery function ................................. 399
16.3.28. pmix_server_stdin_fn_t .................................................................. 400
16.3.29. pmix_server_grp_fn_t .................................................................. 401
    16.3.29.1. Group Operation Constants .................................................. 404
16.3.30. pmix_server_fabric_fn_t ................................................................. 404

17. Tools and Debuggers .......................................................................... 406
    17.1. Connection Mechanisms ................................................................. 406
        17.1.1. Rendezvousing with a local server ......................................... 409
        17.1.2. Connecting to a remote server ............................................... 410
        17.1.3. Attaching to running jobs ..................................................... 411
        17.1.4. Tool initialization attributes .................................................. 411
        17.1.5. Tool initialization environmental variables ............................. 412
        17.1.6. Tool connection attributes .................................................... 412
    17.2. Launching Applications with Tools ................................................ 413
        17.2.1. Direct launch ....................................................................... 414
        17.2.2. Indirect launch .................................................................... 418
            17.2.2.1. Initiator-based command line parsing .............................. 418
            17.2.2.2. Intermediate Launcher (IL)-based command line parsing .. 421
        17.2.3. Tool spawn-related attributes ............................................... 422
        17.2.4. Tool rendezvous-related events ............................................. 423
    17.3. IO Forwarding ............................................................................ 424
        17.3.1. Forwarding stdout/stderr ...................................................... 424
        17.3.2. Forwarding stdin ................................................................. 426
        17.3.3. IO Forwarding Channels ....................................................... 428
        17.3.4. IO Forwarding constants ...................................................... 428
        17.3.5. IO Forwarding attributes ...................................................... 428
    17.4. Debugger Support ..................................................................... 429
        17.4.1. Co-Location of Debugger Daemons ...................................... 432
        17.4.2. Co-Spawn of Debugger Daemons ......................................... 433
        17.4.3. Debugger Agents ................................................................ 434
        17.4.4. Tracking the job lifecycle ..................................................... 435
            17.4.4.1. Job lifecycle events ....................................................... 436
            17.4.4.2. Job lifecycle attributes .................................................. 437
        17.4.5. Debugger-related constants .................................................. 437
17.4.6. Debugger attributes ................................................. 437
17.5. Tool-Specific APIs ...................................................... 439
  17.5.1. PMIx_tool_init .................................................. 439
  17.5.2. PMIx_tool_finalize ............................................. 442
  17.5.3. PMIx_tool_disconnect ......................................... 443
  17.5.4. PMIx_tool_attach_to_server .................................. 444
  17.5.5. PMIx_tool_get_servers ........................................ 446
  17.5.6. PMIx_tool_set_server ......................................... 446
  17.5.7. PMIx_IOF_pull .................................................. 447
  17.5.8. PMIx_IOF_deregister .......................................... 449
  17.5.9. PMIx_IOF_push .................................................. 450

A. Python Bindings 453
  A.1. Design Considerations ............................................ 453
    A.1.1. Error Codes vs Python Exceptions .......................... 453
    A.1.2. Representation of Structured Data .......................... 453
  A.2. Datatype Definitions ............................................. 454
    A.2.1. Example ...................................................... 460
  A.3. Callback Function Definitions .................................. 461
    A.3.1. IOF Delivery Function ....................................... 461
    A.3.2. Event Handler ............................................... 461
    A.3.3. Server Module Functions .................................... 462
      A.3.3.1. Client Connected ....................................... 462
      A.3.3.2. Client Finalized ........................................ 463
      A.3.3.3. Client Aborted .......................................... 463
      A.3.3.4. Fence .................................................... 464
      A.3.3.5. Direct Modex ............................................ 465
      A.3.3.6. Publish .................................................. 465
      A.3.3.7. Lookup .................................................. 466
      A.3.3.8. Unpublish ............................................... 466
      A.3.3.9. Spawn ................................................... 467
      A.3.3.10. Connect ................................................ 467
      A.3.3.11. Disconnect .............................................. 468
      A.3.3.12. Register Events ........................................ 468
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.4.19</td>
<td>Client.query</td>
</tr>
<tr>
<td>A.4.20</td>
<td>Client.log</td>
</tr>
<tr>
<td>A.4.21</td>
<td>Client.allocation_request</td>
</tr>
<tr>
<td>A.4.22</td>
<td>Client.job_ctrl</td>
</tr>
<tr>
<td>A.4.23</td>
<td>Client.monitor</td>
</tr>
<tr>
<td>A.4.24</td>
<td>Client.get_credential</td>
</tr>
<tr>
<td>A.4.25</td>
<td>Client.validate_credential</td>
</tr>
<tr>
<td>A.4.26</td>
<td>Client.group_construct</td>
</tr>
<tr>
<td>A.4.27</td>
<td>Client.group_invite</td>
</tr>
<tr>
<td>A.4.28</td>
<td>Client.group_join</td>
</tr>
<tr>
<td>A.4.29</td>
<td>Client.group_leave</td>
</tr>
<tr>
<td>A.4.30</td>
<td>Client.group_destruct</td>
</tr>
<tr>
<td>A.4.31</td>
<td>Client.register_event_handler</td>
</tr>
<tr>
<td>A.4.32</td>
<td>Client.deregister_event_handler</td>
</tr>
<tr>
<td>A.4.33</td>
<td>Client.notify_event</td>
</tr>
<tr>
<td>A.4.34</td>
<td>Client.fabric_register</td>
</tr>
<tr>
<td>A.4.35</td>
<td>Client.fabric_update</td>
</tr>
<tr>
<td>A.4.36</td>
<td>Client.fabric_deregister</td>
</tr>
<tr>
<td>A.4.37</td>
<td>Client.load_topology</td>
</tr>
<tr>
<td>A.4.38</td>
<td>Client.get_relative_locality</td>
</tr>
<tr>
<td>A.4.39</td>
<td>Client.get_cpuset</td>
</tr>
<tr>
<td>A.4.40</td>
<td>Client.parse_cpuset_string</td>
</tr>
<tr>
<td>A.4.41</td>
<td>Client.compute_distances</td>
</tr>
<tr>
<td>A.4.42</td>
<td>Client.error_string</td>
</tr>
<tr>
<td>A.4.43</td>
<td>Client.proc_state_string</td>
</tr>
<tr>
<td>A.4.44</td>
<td>Client.scope_string</td>
</tr>
<tr>
<td>A.4.45</td>
<td>Client.persistence_string</td>
</tr>
<tr>
<td>A.4.46</td>
<td>Client.data_range_string</td>
</tr>
<tr>
<td>A.4.47</td>
<td>Client.info_directives_string</td>
</tr>
<tr>
<td>A.4.48</td>
<td>Client.data_type_string</td>
</tr>
<tr>
<td>A.4.49</td>
<td>Client.alloc_directive_string</td>
</tr>
<tr>
<td>A.4.50</td>
<td>Client.iof_channel_string</td>
</tr>
<tr>
<td>A.4.51</td>
<td>Client.job_state_string</td>
</tr>
</tbody>
</table>

484 485 485 486 486 487 487 488 489 489 490 490 491 491 492 492 493 493 493 494 494 495 495 496 496 497 497 498 498 498 499 499 500
CHAPTER 1

Introduction

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

1.1 Background

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 [2]) have been released as part of the MPICH effort, with PMI-2 demonstrating better scaling properties than its PMI-1 predecessor.

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

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

More information about the PMIx standard and affiliated projects can be found at the PMIx website: https://pmix.org

1.2 PMIx Architecture Overview

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

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

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

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

Recognizing the burden this places on SMS vendors, the PMIx community has included interfaces
by which the host SMS (containing the local PMIx service instance) can request support from local
SMS elements via the PMIx API. 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 Fabric Manager. 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.

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

1.3 Portability of Functionality

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

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

1.3.1 Attributes in PMIx

An area where differences between support on different systems can be challenging is regarding the
attributes that provide information to the client process and/or control the behavior of a PMIx API.
Most PMIx API calls can accept additional information or attributes specified in the form of key/value pairs. These attributes provide information to the PMIx implementation that influence the behavior of the API call. In addition to API calls being optional, support for the individual attributes of an API call can vary between systems or implementations.

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

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).

The application can query the attribute support for `PMIX_Fence` and search whether `PMIX_TIMEOUT` is listed as a supported attribute. The application can also set the required flag in the `pmix_info_t` for that attribute when making the `PMIX_Fence` call. This will return an error if this attribute is not supported. If the required flag is not set, 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.

PMIx clients (e.g., tools, parallel programming 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.

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

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.

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

```
PMIX_ATTR_UNDEF "pmix.undef" (NULL)
```

A default attribute name signifying that the attribute field of a PMIx structure (e.g., a `pmix_info_t`) has not yet been defined.
In this chapter we describe some common terms and conventions used throughout this document. The PMIx Standard has adopted the widespread use of key-value attributes to add flexibility to the functionality expressed in the existing APIs. Accordingly, the ASC has chosen to require that the definition of each standard API include the passing of an array of attributes. These provide a means of customizing the behavior of the API as future needs emerge without having to alter or create new variants of it. In addition, attributes provide a mechanism by which researchers can easily explore new approaches to a given operation without having to modify the API itself.

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

The following terminology is used throughout this document:

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

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

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

- **application** refers to a single executable (binary, script, etc.) member of a *job*. 
• process refers to an operating system process, also commonly referred to as a heavy weight process. A process is often comprised of multiple lightweight threads, commonly known as simply threads.

• client refers to a process that was registered with the PMIx server prior to being started, and connects to that PMIx server via PMIx_Init using its assigned namespace and rank with the information required to connect to that server being provided to the process at time of start of execution.

• clone refers to a process that was directly started by a PMIx client (e.g., using fork/exec) and calls PMIx_Init, thus connecting to its local PMIx server using the same namespace and rank as its parent process.

• rank refers to the numerical location (starting from zero) of a process within the defined scope. Thus, job rank is the rank of a process within its job and is synonymous with its unqualified rank, while application rank is the rank of that process within its application.

• peer refers to another process within the same job.

• workflow refers to an orchestrated execution plan frequently involving multiple jobs carried out under the control of a workflow manager process. An example workflow might first execute a computational job to generate the flow of liquid through a complex cavity, followed by a visualization job that takes the output of the first job as its input to produce an image output.

• scheduler refers to the component of the SMS responsible for scheduling of resource allocations. This is also generally referred to as the system workflow manager - for the purposes of this document, the WLM acronym will be used interchangeably to refer to the scheduler.

• resource manager is used in a generic sense to represent the subsystem that will host the PMIx server library. This could be a vendor-supplied resource manager or a third-party agent such as a programming model’s runtime library.

• host environment is used interchangeably with resource manager to refer to the process hosting the PMIx server library.

• node refers to a single operating system instance. Note that this may encompass one or more physical objects.

• package refers to a single object that is either soldered or connected to a printed circuit board via a mechanical socket. Packages may contain multiple chips that include (but are not limited to) processing units, memory, and peripheral interfaces.

• processing unit, or PU, is the electronic circuitry within a computer that executes instructions. Depending upon architecture and configuration settings, it may consist of a single hardware thread or multiple hardware threads collectively organized as a core.

• fabric is used in a generic sense to refer to the networks within the system regardless of speed or protocol. Any use of the term network in the document should be considered interchangeable with fabric.
• fabric device (or fabric devices) refers to an operating system fabric interface, which may be physical or virtual. Any use of the term Network Interface Card (NIC) in the document should be considered interchangeable with fabric device.

• fabric plane refers to a collection of fabric devices in a common logical or physical configuration. Fabric planes are often implemented in HPC clusters as separate overlay or physical networks controlled by a dedicated fabric manager.

• attribute refers to a key-value pair comprised of a string key (represented by a pmix_key_t structure) and an associated value containing a PMIx data type (e.g., boolean, integer, or a more complex PMIx structure). Attributes are used both as directives when passed as qualifiers to APIs (e.g., in a pmix_info_t array), and to identify the contents of information (e.g., to specify that the contents of the corresponding pmix_value_t in a pmix_info_t represent the PMIX_UNIV_SIZE).

• key refers to the string component of a defined attribute. The PMIx Standard will often refer to passing of a key to an API (e.g., to the PMIx_Query_info or PMIx_Get APIs) as a means of identifying requested information. In this context, the data type specified in the attribute's definition indicates the data type the caller should expect to receive in return. Note that not all attributes can be used as keys as some have specific uses solely as API qualifiers.

• instant on refers to a PMIx concept defined as: "All information required for setup and communication (including the address vector of endpoints for every process) is available to each process at start of execution"

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

2.1 Notational Conventions

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

```
C specific text...
```

```
int foo = 42;
```

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

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

Advice to users

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

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

2.2 Semantics

The following terms will be taken to mean:

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

The PMIx standard has adopted the following conventions:

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

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

2.4 Procedure Conventions

While the current APIs are 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*)` callback data object passed to the function that is then passed to the associated callback. On the client side, the callback data object is an opaque, client-provided context that the client can pass to a non-blocking call. When the nonblocking call completes, the callback data object 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 16.3.1). In this case, the PMIx library is making an upcall into its host via the PMIx server module callback function and passing a specific callback function pointer and callback data object. The PMIx library expects the host to call the cbfunc with the necessary arguments and pass back the original callback data object 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 callback data object, nor is permitted to alter it in any way.
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:

```c
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:

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

since the index is cited more than once in the macro. The PMIx standard only governs the existence and syntax of macros - it does not specify their implementation. Given the freedom of implementation, a safer call sequence might be as follows:
PMIX_INFO_LOAD(&array[n], "mykey", &mystring, PMIX_STRING);
++n;
PMIX_INFO_LOAD(&array[n], "mykey2", &myint, PMIX_INT);
++n;

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

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.

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.

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 namespace.
3.1.1 PMIx Return Status Constants

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

--- Advice to PMIx library implementers ---

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

--- Advice to users ---

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

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

- **PMIX_SUCCESS** Success.
- **PMIX_ERROR** General Error.
- **PMIX_ERR_EXISTS** Requested operation would overwrite an existing value - typically returned when an operation would overwrite an existing file or directory.
- **PMIX_ERR_EXISTS_OUTSIDE_SCOPE** The requested key exists, but was posted in a `scope` (see Section 7.1.1.1) that does not include the requester
- **PMIX_ERR_INVALID_CRED** Invalid security credentials.
- **PMIX_ERR_WOULD_BLOCK** Operation would block.
- **PMIX_ERR_UNKNOWN_DATA_TYPE** The data type specified in an input to the PMIx library is not recognized by the implementation.
- **PMIX_ERR_TYPE_MISMATCH** The data type found in an object does not match the expected data type as specified in the API call - e.g., a request to unpack a `PMIX_BOOL` value from a buffer that does not contain a value of that type in the current unpack location.
- **PMIX_ERR_UNPACK_INADEQUATE_SPACE** Inadequate space to unpack data - the number of values in the buffer exceeds the specified number to unpack.
- **PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER** Unpacking past the end of the provided buffer - the number of values in the buffer is less than the specified number to unpack, or a request was made to unpack a buffer beyond the buffer’s end.
- **PMIX_ERR_UNPACK_FAILURE** The unpack operation failed for an unspecified reason.
- **PMIX_ERR_PACK_FAILURE** The pack operation failed for an unspecified reason.
- **PMIX_ERR_NO_PERMISSIONS** The user lacks permissions to execute the specified operation.
- **PMIX_ERR_TIMEOUT** Either a user-specified or system-internal timeout expired.
**PMIX_ERR_UNREACH** The specified target server or client process is not reachable - i.e., a suitable connection either has not been or can not be made.

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

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

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

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

**PMIX_ERR_INIT** Error during initialization.

**PMIX_ERR_NOMEM** Out of memory.

**PMIX_ERR_NOT_FOUND** The requested information was not found.

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

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

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

**PMIX_ERR_LOST_CONNECTION** Lost connection between server and client or tool.

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

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

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

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

### 3.1.1.1 User-Defined Error and Event Constants

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

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

3.2 Data Types

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

3.2.1 Key Structure

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

```
PMIx v2.0
typedef char pmix_key_t[PMIX_MAX_KEYLEN+1];
```

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

Advice to users

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

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

3.2.1.1 Key support macros

The following macros are provided for convenience when working with PMIx keys.
Check key macro

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

```
PMIx v3.0
PMIX_CHECK_KEY(a, b)
```

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

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

Returns `true` if the key matches the given value.

Check reserved key macro

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

```
PMIx v4.0
PMIX_CHECK_RESERVED_KEY(a)
```

**IN** `a`
String value to be checked (char*)

Returns `true` if the key is reserved by the Standard.

Load key macro

Load a key into a `pmix_info_t`.

```
PMIx v4.0
PMIX_LOAD_KEY(a, b)
```

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

**IN** `b`
String value to be loaded (char*)

No return value.
### 3.2.2 Namespace Structure

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

```c
PMIx v2.0
typedef char pmix_nspace_t[PMIX_MAX_NSLEN+1];
```

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

**Advice to users**

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

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

### 3.2.2.1 Namespace support macros

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

**Check namespace macro**

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

```c
PMIx v3.0
PMIX_CHECK_NSPACE(a, b)
```

- **IN a**
  - Pointer to the structure whose value is to be checked (pointer to `pmix_nspace_t`)
- **IN b**
  - String value to be compared against (char*)

Returns true if the namespace matches the given value.
Load namespace macro
Load a namespace into a `pmix_nspace_t`.

```c
PMIx v4.0

PMIX_LOAD_NSPACE(a, b)
```

```
IN a
Pointer to the target structure (pointer to `pmix_nspace_t`)

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

No return value.

### 3.2.3 Rank Structure

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

```c
PMIx v1.0

typedef uint32_t pmix_rank_t;
```

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

- **PMIX_RANK_UNDEF**: A value to request job-level data where the information itself is not associated with any specific rank, or when passing a `pmix_proc_t` identifier to an operation that only references the namespace field of that structure.
- **PMIX_RANK_WILDCARD**: A value to indicate that the user wants the data for the given key from every rank that posted that key.
- **PMIX_RANK_LOCAL_NODE**: Special rank value used to define groups of ranks. This constant defines the group of all ranks on a local node.
- **PMIX_RANK_LOCAL_PEERS**: Special rank value used to define groups of ranks. This constant defines the group of all ranks on a local node within the same namespace as the current process.
- **PMIX_RANK_INVALID**: An invalid rank value.
- **PMIX_RANK_VALID**: Define an upper boundary for valid rank values.

### 3.2.3.1 Rank support macros

The following macros are provided for convenience when working with PMIx ranks.
Check rank macro

Check two ranks for equality, taking into account wildcard values

```c
PMIX_CHECK_RANK(a, b)
```

<table>
<thead>
<tr>
<th>IN</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rank to be checked (pmix_rank_t)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rank to be checked (pmix_rank_t)</td>
</tr>
</tbody>
</table>

Returns `true` if the ranks are equal, or at least one of the ranks is `PMIX_RANK_WILDCARD`.

### 3.2.4 Process Structure

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

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

### 3.2.4.1 Process structure support macros

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

**Initialize the proc structure**

Initialize the `pmix_proc_t` fields.

```c
PMIX_PROC_CONSTRUCT(m)
```

<table>
<thead>
<tr>
<th>IN</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointer to the structure to be initialized (pointer to pmix_proc_t)</td>
</tr>
</tbody>
</table>
Destruct the proc structure

Destruct the pmix_proc_t fields.

```c
PMIX_PROC_DESTRUCT(m)
```

**IN**  
Pointer to the structure to be destructed (pointer to pmix_proc_t)

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

Create a proc array

Allocate and initialize an array of pmix_proc_t structures.

```c
PMIX v1.0
PMIX_PROC_CREATE(m, n)
```

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

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

Free a proc structure

Release a pmix_proc_t structure.

```c
PMIX v4.0
PMIX_PROC_RELEASE(m)
```

**IN**  
Pointer to a pmix_proc_t structure (handle)

Free a proc array

Release an array of pmix_proc_t structures.

```c
PMIX v1.0
PMIX_PROC_FREE(m, n)
```

**IN**  
Pointer to the array of pmix_proc_t structures (handle)

**IN**  
Number of structures in the array (size_t)
Load a proc structure
Load values into a `pmix_proc_t`.

`PMIx v2.0`

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

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

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

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

No return value. Deprecated in favor of `PMIX_LOADPROCID`

Compare identifiers
Compare two `pmix_proc_t` identifiers.

`PMIx v3.0`

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

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

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

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

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

Load a procID structure
Load values into a `pmix_proc_t`.

`PMIx v4.0`

```c
PMIX_LOADPROCID(m, n, r)
```

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

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

**IN** `r`
Rank to be assigned (`pmix_rank_t`)
Construct a multi-cluster namespace
Construct a multi-cluster identifier containing a cluster ID and a namespace.

PMIx v4.0
PMIX_MULTICLUSTER_NSPACE_CONSTRUCT(m, n, r)

IN  m
   pmix_namespace_t structure that will contain the multi-cluster identifier (pmix_namespace_t)
IN  n
   Cluster identifier (char*)
IN  n
   Namespace to be loaded (pmix_namespace_t)

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

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

PMIx v4.0
PMIX_MULTICLUSTER_NSPACE_PARSE(m, n, r)

IN  m
   pmix_namespace_t structure containing the multi-cluster identifier (pointer to pmix_namespace_t)
IN  n
   Location where the cluster ID is to be stored (pmix_namespace_t)
IN  n
   Location where the namespace is to be stored (pmix_namespace_t)

3.2.5 Process State Structure

PMIx v2.0
The pmix_proc_state_t structure is a uint8_t type for process state values. The following constants can be used to set a variable of the type pmix_proc_state_t.

Advice to users
The fine-grained nature of the following constants may exceed the ability of an RM to provide updated process state values during the process lifetime. This is particularly true of states for short-lived processes.
PMIX_PROC_STATE_UNDEF    Undefined process state.
PMIX_PROC_STATE_PREPPED    Process is ready to be launched.
PMIX_PROC_STATE_LAUNCH_UNDERWAY    Process launch is underway.
PMIX_PROC_STATE_RESTART    Process is ready for restart.
PMIX_PROC_STATE_TERMINATE    Process is marked for termination.
PMIX_PROC_STATE_RUNNING    Process has been locally forked by the RM.
PMIX_PROC_STATE_CONNECTED    Process has connected to PMIx server.
PMIX_PROC_STATE_UNTERMINATED    Define a “boundary” between the terminated states and PMIX_PROC_STATE_CONNECTED so users can easily and quickly determine if a process is still running or not. Any value less than this constant means that the process has not terminated.
PMIX_PROC_STATE_TERMINATED    Process has terminated and is no longer running.
PMIX_PROC_STATE_ERROR    Define a boundary so users can easily and quickly determine if a process abnormally terminated. Any value above this constant means that the process has terminated abnormally.
PMIX_PROC_STATE_KILLED_BY_CMD    Process was killed by a command.
PMIX_PROC_STATE_ABORTED    Process was aborted by a call to PMIx_Abort.
PMIX_PROC_STATE_FAILED_TO_START    Process failed to start.
PMIX_PROC_STATE_ABORTED_BY_SIG    Process aborted by a signal.
PMIX_PROC_STATE_TERM_WO_SYNC    Process exited without calling PMIx_Finalize.
PMIX_PROC_STATE_COMM_FAILED    Process communication has failed.
PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED    Process exceeded a specified sensor limit.
PMIX_PROC_STATE_CALLED_ABORT    Process called PMIx_Abort.
PMIX_PROC_STATE_HEARTBEAT_FAILED    Process failed to send heartbeat within specified time limit.
PMIX_PROC_STATE_MIGRATING    Process failed and is waiting for resources before restarting.
PMIX_PROC_STATE_CANNOT_RESTART    Process failed and cannot be restarted.
PMIX_PROC_STATE_TERM_NON_ZERO    Process exited with a non-zero status.
PMIX_PROC_STATE_FAILED_TO_LAUNCH    Unable to launch process.

3.2.6 Process Information Structure

The pmix_proc_info_t structure defines a set of information about a specific process including its name, location, and state.
typedef struct pmix_proc_info {
    /** Process structure */
    pmix_proc_t proc;
    /** Hostname where process resides */
    char *hostname;
    /** Name of the executable */
    char *executable_name;
    /** Process ID on the host */
    pid_t pid;
    /** Exit code of the process. Default: 0 */
    int exit_code;
    /** Current state of the process */
    pmix_proc_state_t state;
} pmix_proc_info_t;

3.2.6.1 Process information structure support macros

The following macros are provided to support the pmix_proc_info_t structure.

**Initialize the process information structure**
Initialize the pmix_proc_info_t fields.

```c
PMIx v2.0

PMIX_PROC_INFO_CONSTRUCT(m)
```

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

**Destruct the process information structure**
Destruct the pmix_proc_info_t fields.

```c
PMIx v2.0

PMIX_PROC_INFO_DESTRUCT(m)
```

**IN**  m
Pointer to the structure to be destructed (pointer to pmix_proc_info_t)
Create a process information array
Allocate and initialize a `pmix_proc_info_t` array.

```
PMIx v2.0
PMIX_PROC_INFO_CREATE(m, n)
```

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

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

Free a process information structure
Release a `pmix_proc_info_t` structure.

```
PMIx v2.0
PMIX_PROC_INFO_RELEASE(m)
```

**IN** m
Pointer to a `pmix_proc_info_t` structure (handle)

Free a process information array
Release an array of `pmix_proc_info_t` structures.

```
PMIx v2.0
PMIX_PROC_INFO_FREE(m, n)
```

**IN** m
Pointer to the array of `pmix_proc_info_t` structures (handle)
**IN** n
Number of structures in the array (`size_t`)
3.2.7  Job State Structure

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

**Advice to users**

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

- `PMIX_JOB_STATE_UNDEF` Undefined job state.
- `PMIX_JOB_STATE_AWAITING_ALLOC` Job is waiting for resources to be allocated to it.
- `PMIX_JOB_STATE_LAUNCH_UNDERWAY` Job launch is underway.
- `PMIX_JOB_STATE_RUNNING` All processes in the job have been spawned and are executing.
- `PMIX_JOB_STATE_SUSPENDED` All processes in the job have been suspended.
- `PMIX_JOB_STATE_CONNECTED` All processes in the job have connected to their PMIx server.
- `PMIX_JOB_STATE_UNTERMINATED` Define a “boundary” between the terminated states and `PMIX_JOB_STATE_TERMINATED` so users can easily and quickly determine if a job is still running or not. Any value less than this constant means that the job has not terminated.
- `PMIX_JOB_STATE_TERMINATED` All processes in the job have terminated and are no longer running - typically will be accompanied by the job exit status in response to a query.
- `PMIX_JOB_STATE_TERMINATED_WITH_ERROR` Define a boundary so users can easily and quickly determine if a job abnormally terminated - typically will be accompanied by a job-related error code in response to a query. Any value above this constant means that the job terminated abnormally.

3.2.8  Value Structure

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

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

*PMIx v1.0*
### 3.2.8.1 Value structure support macros

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

Initialize the \texttt{pmix\_value\_t} fields.

\texttt{PMIx v1.0} \hspace{2cm} \texttt{C} \\
\texttt{PMIX\_VALUE\_CONSTRUCT(m)} \hspace{2cm} \texttt{C}

\textbf{IN} \hspace{1cm} m \\
Pointer to the structure to be initialized (pointer to \texttt{pmix\_value\_t})

Destruct the value structure

Destruct the \texttt{pmix\_value\_t} fields.

\texttt{PMIx v1.0} \hspace{2cm} \texttt{C} \\
\texttt{PMIX\_VALUE\_DESTRUCT(m)} \hspace{2cm} \texttt{C}

\textbf{IN} \hspace{1cm} m \\
Pointer to the structure to be destructed (pointer to \texttt{pmix\_value\_t})

Create a value array

Allocate and initialize an array of \texttt{pmix\_value\_t} structures.

\texttt{PMIx v1.0} \hspace{2cm} \texttt{C} \\
\texttt{PMIX\_VALUE\_CREATE(m, n)} \hspace{2cm} \texttt{C}

\textbf{INOUT} \hspace{1cm} m \\
Address where the pointer to the array of \texttt{pmix\_value\_t} structures shall be stored (handle)

\textbf{IN} \hspace{1cm} n \\
Number of structures to be allocated (\texttt{size\_t})

Free a value structure

Release a \texttt{pmix\_value\_t} structure.

\texttt{PMIx v4.0} \hspace{2cm} \texttt{C} \\
\texttt{PMIX\_VALUE\_RELEASE(m)} \hspace{2cm} \texttt{C}

\textbf{IN} \hspace{1cm} m \\
Pointer to a \texttt{pmix\_value\_t} structure (handle)
**Free a value array**
Release an array of `pmix_value_t` structures.

*PMIx v1.0*

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

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

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

**Load a value structure**
Load data into a `pmix_value_t` structure.

*PMIx v2.0*

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

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

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

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

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

---

**Advice to users**

The data will be copied into the `pmix_value_t` - thus, any data stored in the source value can be modified or free’d without affecting the copied data once the macro has completed.
Unload a value structure
Unload data from a `pmix_value_t` structure.

```
PMIx v2.2
PMIX_VALUE_UNLOAD(r, v, d, t);
```

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

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

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

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

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

--- Advice to users ---

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

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

```
PMIx v2.0
PMIX_VALUE_XFER(r, d, s);
```

- **OUT** `r`
  Status code indicating success or failure of the transfer (`pmix_status_t`)  

- **IN** `d`
  Pointer to the `pmix_value_t` destination (handle)  

- **IN** `s`
  Pointer to the `pmix_value_t` source (handle)  

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

--- Advice to users ---

The data will be copied into the destination `pmix_value_t` - thus, any data stored in the source value can be modified or free’d without affecting the copied data once the macro has completed.
Retrieve a numerical value from a value struct

Retrieve a numerical value from a `pmix_value_t` structure.

```c
PMIx v3.0

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

**OUT** `s`
- Status code for the request (`pmix_status_t`)

**IN** `m`
- Pointer to the `pmix_value_t` structure (handle)

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

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

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

### 3.2.9 Info Structure

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

```c
PMIx v1.0

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

### 3.2.9.1 Info structure support macros

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

**Initialize the info structure**
Initialize the `pmix_info_t` fields.

```c
PMIx v1.0

PMIX_INFO_CONSTRUCT(m)
```

**IN** `m`
- Pointer to the structure to be initialized (pointer to `pmix_info_t`)
Destruct the info structure
Destruct the \texttt{pmix\_info\_t} fields.

\textit{PMIx v1.0}
\begin{verbatim}
PMIX\_INFO\_DESTRUCT(m)
\end{verbatim}

\textbf{IN} \hspace{1cm} m
Pointer to the structure to be destructed (pointer to \texttt{pmix\_info\_t})

Create an info array
Allocate and initialize an array of info structures.

\textit{PMIx v1.0}
\begin{verbatim}
PMIX\_INFO\_CREATE(m, n)
\end{verbatim}

\textbf{INOUT} \hspace{1cm} m
Address where the pointer to the array of \texttt{pmix\_info\_t} structures shall be stored (handle)

\textbf{IN} \hspace{1cm} n
Number of structures to be allocated (\texttt{size\_t})

Free an info array
Release an array of \texttt{pmix\_info\_t} structures.

\textit{PMIx v1.0}
\begin{verbatim}
PMIX\_INFO\_FREE(m, n)
\end{verbatim}

\textbf{IN} \hspace{1cm} m
Pointer to the array of \texttt{pmix\_info\_t} structures (handle)

\textbf{IN} \hspace{1cm} n
Number of structures in the array (\texttt{size\_t})

Load key and value data into a info struct
PMIX_INFO_LOAD(v, k, d, t);

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

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

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

IN  t
    Type of the provided data value (pmix_data_type_t)

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

Advice to users

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

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

PMIX v2.0
PMIX_INFO_XFER(d, s);

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

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

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

Advice to users

All data (including key, value, and directives) will be copied into the destination pmix_info_t - thus, the source pmix_info_t may be free’d without affecting the copied data once the macro has completed.
Test a boolean info struct
A special macro for checking if a boolean `pmix_info_t` is `true`.

**PMIx v2.0**

```c
PMIX_INFO_TRUE(m)
```

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

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

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

### 3.2.9.2 Info structure list macros

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

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

**PMIx v4.0**

```c
PMIX_INFO_LIST_START(m)
```

**IN** `m`
A `void*` pointer (handle)

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

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

**PMIx v4.0**
PMIX_INFO_LIST_ADD(rc, m, k, d, t)

**INOUT** rc
Return status for the operation (**pmix_status_t**)

**IN** m
A **void** pointer initialized via **PMIX_INFO_LIST_START** (handle)

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

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

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

Advice to users
Both key and data will be copied into the **pmix_info_t** on the list - thus, the key and any data stored in the source value can be modified or free’d without affecting the copied data once the macro has completed.

Transfer a **pmix_info_t** structure to a list
Transfer the information in a **pmix_info_t** structure to the provided list.

PMIX v4.0

PMIX_INFO_LIST_XFER(rc, m, s)

**INOUT** rc
Return status for the operation (**pmix_status_t**)

**IN** m
A **void** pointer initialized via **PMIX_INFO_LIST_START** (handle)

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

Advice to users
All data (including key, value, and directives) will be copied into the destination **pmix_info_t** on the list - thus, the source **pmix_info_t** may be free’d without affecting the copied data once the macro has completed.
Convert a `pmix_info_t` list to an array

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

```c
PMIx v4.0
PMIX_INFO_LIST_CONVERT(rc, m, d)
```

**INOUT** rc

Return status for the operation (`pmix_status_t`)

**IN** m

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

**IN** d

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

Release a `pmix_info_t` list

Release the provided `pmix_info_t` list

```c
PMIx v4.0
PMIX_INFO_LIST_RELEASE(m)
```

**IN** m

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

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

### 3.2.10 Info Type Directives

*PMIx v2.0*

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

Advice to users

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

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

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

- `PMIX_INFO_REQD` The behavior defined in the `pmix_info_t` array is required, and not optional. This is a bit-mask value.
- `PMIX_INFO_REQD_PROCESSED` Mark that this required attribute has been processed. A required attribute can be handled at any level - the PMIx client library might take care of it, or it may be resolved by the PMIx server library, or it may pass up to the host environment for handling. If a level does not recognize or support the required attribute, it is required to pass it upwards to give the next level an opportunity to process it. Thus, the host environment (or the server library if the host does not support the given operation) must know if a lower level has handled the requirement so it can return a `PMIX_ERR_NOT_SUPPORTED` error status if the host itself cannot meet the request. Upon processing the request, the level must therefore mark the attribute with this directive to alert any subsequent levels that the requirement has been met.
- `PMIX_INFO_ARRAY_END` Mark that this `pmix_info_t` struct is at the end of an array created by the `PMIX_INFO_CREATE` macro. This is a bit-mask value.
- `PMIX_INFO_DIR_RESERVED` A bit-mask identifying the bits reserved for internal use by implementers - these currently are set as `0xffffffff`.

Advice to PMIx server hosts

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

3.2.10.1 Info Directive support macros

The following macros are provided to support the setting and testing of `pmix_info_t` directives.
Mark an info structure as required
Set the PMIX_INFO_REQD flag in a pmix_info_t structure.

PMIx v2.0
PMIX_INFO_REQUIRED(info);  C

IN info
Pointer to the pmix_info_t (pointer to pmix_info_t)

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

Mark an info structure as optional
Unsets the PMIX_INFO_REQD flag in a pmix_info_t structure.

PMIx v2.0
PMIX_INFO_OPTIONAL(info);  C

IN info
Pointer to the pmix_info_t (pointer to pmix_info_t)

This macro simplifies marking a pmix_info_t structure as optional.

Test an info structure for required directive
Test the PMIX_INFO_REQD flag in a pmix_info_t structure, returning true if the flag is set.

PMIx v2.0
PMIX_INFO_IS_REQUIRED(info);  C

IN info
Pointer to the pmix_info_t (pointer to pmix_info_t)

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

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

PMIx v2.0
PMIX_INFO_IS_OPTIONAL(info);  C

IN info
Pointer to the pmix_info_t (pointer to pmix_info_t)

Test the PMIX_INFO_REQD flag in a pmix_info_t structure, returning true if the flag is not set.
Mark a required attribute as processed
Mark that a required pmix_info_t structure has been processed.

PMIx v4.0
PMIX_INFO_PROCESSED(info);

IN  info
  Pointer to the pmix_info_t (pointer to pmix_info_t)

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

Test if a required attribute has been processed
Test that a required pmix_info_t structure has been processed.

PMIx v4.0
PMIX_INFO_WAS_PROCESSED(info);

IN  info
  Pointer to the pmix_info_t (pointer to pmix_info_t)

Test the PMIX_INFO_REQD_PROCESSED flag in a pmix_info_t structure.

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

PMIx v2.2
PMIX_INFO_IS_END(info);

IN  info
  Pointer to the pmix_info_t (pointer to pmix_info_t)

This macro simplifies the testing of the end-of-array flag in pmix_info_t structures.
3.2.11 Environmental Variable Structure

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

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

3.2.11.1 Environmental variable support macros

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

**Initialize the envar structure**

Initialize the `pmix_envar_t` fields.

```
PMIx v3.0

PMIX_ENVAR_CONSTRUCT(m)
```

**IN** m

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

**Destruct the envar structure**

Clear the `pmix_envar_t` fields.

```
PMIx v3.0

PMIX_ENVAR_DESTRUCT(m)
```

**IN** m

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

CHAPTER 3. DATA STRUCTURES AND TYPES 41
Create an envar array
Allocate and initialize an array of `pmix_envar_t` structures.

```c
PMIx v3.0
PMIX_ENVAR_CREATE(m, n)
```

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

Free an envar array
Release an array of `pmix_envar_t` structures.

```c
PMIx v3.0
PMIX_ENVAR_FREE(m, n)
```

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

Load an envar structure
Load values into a `pmix_envar_t`.

```c
PMIx v2.0
PMIX_ENVAR_LOAD(m, e, v, s)
```

- **IN m**: Pointer to the structure to be loaded (pointer to `pmix_envar_t`)
- **IN e**: Environmental variable name (`char*`)
- **IN v**: Value of variable (`char*`)
- **IN s**: Separator character (`char`)
3.2.12 Byte Object Type

The `pmix_byte_object_t` structure describes a raw byte sequence.

```c
PMIx v1.0
typedef struct pmix_byte_object {
    char *bytes;
    size_t size;
} pmix_byte_object_t;
```

3.2.12.1 Byte object support macros

The following macros support the `pmix_byte_object_t` structure.

**Initialize the byte object structure**

Initialize the `pmix_byte_object_t` fields.

```c
PMIx v2.0
PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

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

**Destruct the byte object structure**

Clear the `pmix_byte_object_t` fields.

```c
PMIx v2.0
PMIX_BYTE_OBJECT_DESTRUCT(m)
```

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

**Create a byte object structure**

Allocate and initialize an array of `pmix_byte_object_t` structures.

```c
PMIx v2.0
PMIX_BYTE_OBJECT_CREATE(m, n)
```

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

**IN** `n`  
Number of structures to be allocated (`size_t`)
Free a byte object array
Release an array of `pmix_byte_object_t` structures.

```c
PMIx v2.0
PMIX_BYTE_OBJECT_FREE(m, n)
```

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

Load a byte object structure
Load values into a `pmix_byte_object_t`.

```c
PMIx v2.0
PMIX_BYTE_OBJECT_LOAD(b, d, s)
```

- **IN b**
  - Pointer to the structure to be loaded (pointer to `pmix_byte_object_t`)
- **IN d**
  - Pointer to the data to be loaded (`char*`)
- **IN s**
  - Number of bytes in the data array (`size_t`)

### 3.2.13 Data Array Structure
The `pmix_data_array_t` structure defines an array data structure.

```c
PMIx v2.0
typedef struct pmix_data_array {
    pmix_data_type_t type;
    size_t size;
    void *array;
} pmix_data_array_t;
```

### 3.2.13.1 Data array support macros
The following macros support the `pmix_data_array_t` structure.
Initialize a data array structure
Initialize the \texttt{pmix\_data\_array\_t} fields, allocating memory for the array of the indicated type.

\textit{PMIx v2.2}\n
\begin{verbatim}
PMIX\_DATA\_ARRAY\_CONSTRUCT(m, n, t)
\end{verbatim}

\textbf{IN} \hspace{1em} \texttt{m}
Pointer to the structure to be initialized (pointer to \texttt{pmix\_data\_array\_t})

\textbf{IN} \hspace{1em} \texttt{n}
Number of elements in the array (\textit{size\_t})

\textbf{IN} \hspace{1em} \texttt{t}
PMIx data type of the array elements (\texttt{pmix\_data\_type\_t})

Destruct a data array structure
Destruct the \texttt{pmix\_data\_array\_t}, releasing the memory in the array.

\textit{PMIx v2.2}\n
\begin{verbatim}
PMIX\_DATA\_ARRAY\_CONSTRUCT(m)
\end{verbatim}

\textbf{IN} \hspace{1em} \texttt{m}
Pointer to the structure to be destructed (pointer to \texttt{pmix\_data\_array\_t})

Create a data array structure
Allocate memory for the \texttt{pmix\_data\_array\_t} object itself, and then allocate memory for the array of the indicated type.

\textit{PMIx v2.2}\n
\begin{verbatim}
PMIX\_DATA\_ARRAY\_CREATE(m, n, t)
\end{verbatim}

\textbf{INOUT} \hspace{1em} \texttt{m}
Variable to be set to the address of the structure (pointer to \texttt{pmix\_data\_array\_t})

\textbf{IN} \hspace{1em} \texttt{n}
Number of elements in the array (\textit{size\_t})

\textbf{IN} \hspace{1em} \texttt{t}
PMIx data type of the array elements (\texttt{pmix\_data\_type\_t})
Free a data array structure
Release the memory in the array, and then release the `pmix_data_array_t` object itself.

```
PMIx v2.2
PMIX_DATA_ARRAY_FREE (m)
```

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

3.2.14 Argument Array Macros
The following macros support the construction and release of `NULL`-terminated argv arrays of strings.

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

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

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

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

IN b
Argument to append to the list (string)
```

This function helps the caller build the `argv` portion of `pmix_app_t` structure, arrays of keys for querying, or other places where argv-style string arrays are required.

Advice to users
The provided argument is copied into the destination array - thus, the source string can be free’d without affecting the array once the macro has completed.
**Argument array prepend**

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

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

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

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

**IN**  
Argument to append to the list (string)

This function helps the caller build the argv portion of pmix_app_t structure, arrays of keys for querying, or other places where argv-style string arrays are required.

**Advice to users**

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

---

**Argument array extension - unique**

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

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

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

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

**IN**  
Argument to append to the list (string)

This function helps the caller build the argv portion of pmix_app_t structure, arrays of keys for querying, or other places where argv-style string arrays are required.

**Advice to users**

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

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

```c
PMIX_ARGV_FREE(a);
```

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

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

**Argument array split**

Split a string into a NULL-terminated argv array.

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

OUT `a`  
Resulting argv-style array (`char**`)

IN `b`  
String to be split (`char*`)

IN `c`  
Delimiter character (`char`)

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

---

**Advice to users**

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

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

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

- **OUT** `a`
  - Resulting string (`char*`)
- **IN** `b`
  - Argv-style array to be joined (`char**`)
- **IN** `c`
  - Delimiter character (`char`)

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

**Argument array count**

Return the length of a NULL-terminated argv array.

```c
PMIX_Argv_Count(r, a);
```

- **OUT** `r`
  - Number of strings in the array (integer)
- **IN** `a`
  - Argv-style array (`char**`)

Count the number of elements in an argv array.

**Argument array copy**

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

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

- **OUT** `a`
  - New argv-style array (`char**`)
- **IN** `b`
  - Argv-style array (`char**`)

Copy an argv array, including copying all of its strings.
3.2.15 Set Environment Variable

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

```c
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

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

**Advice to PMIx library implementers**
The following constants can be used to set a variable of the type `pmix_data_type_t`. Data types in the PMIx Standard are defined in terms of the C-programming language. Implementers wishing to support other languages should provide the equivalent definitions in a language-appropriate manner. Additionally, a PMIx implementation may choose to add additional types.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMIX_UNDEF</td>
<td>Undefined.</td>
</tr>
<tr>
<td>PMIX_BOOL</td>
<td>Boolean (converted to/from native true/false) (bool).</td>
</tr>
<tr>
<td>PMIX_BYTE</td>
<td>A byte of data (uint8_t).</td>
</tr>
<tr>
<td>PMIX_STRING</td>
<td>NULL terminated string (char*).</td>
</tr>
<tr>
<td>PMIX_SIZE</td>
<td>Size size_t.</td>
</tr>
<tr>
<td>PMIX_PID</td>
<td>Operating Process IDentifier (PID) (pid_t).</td>
</tr>
<tr>
<td>PMIX_INT</td>
<td>Integer (int).</td>
</tr>
<tr>
<td>PMIX_INT8</td>
<td>8-byte integer (int8_t).</td>
</tr>
<tr>
<td>PMIX_INT16</td>
<td>16-byte integer (int16_t).</td>
</tr>
<tr>
<td>PMIX_INT32</td>
<td>32-byte integer (int32_t).</td>
</tr>
<tr>
<td>PMIX_INT64</td>
<td>64-byte integer (int64_t).</td>
</tr>
<tr>
<td>PMIX_UINT</td>
<td>Unsigned integer (unsigned int).</td>
</tr>
<tr>
<td>PMIX_UINT8</td>
<td>Unsigned 8-byte integer (uint8_t).</td>
</tr>
<tr>
<td>PMIX_UINT16</td>
<td>Unsigned 16-byte integer (uint16_t).</td>
</tr>
<tr>
<td>PMIX_UINT32</td>
<td>Unsigned 32-byte integer (uint32_t).</td>
</tr>
<tr>
<td>PMIX_UINT64</td>
<td>Unsigned 64-byte integer (uint64_t).</td>
</tr>
<tr>
<td>PMIX_FLOAT</td>
<td>Float (float).</td>
</tr>
<tr>
<td>PMIX_DOUBLE</td>
<td>Double (double).</td>
</tr>
<tr>
<td>PMIX_TIMEVAL</td>
<td>Time value (struct timeval).</td>
</tr>
<tr>
<td>PMIX_TIME</td>
<td>Time (time_t).</td>
</tr>
<tr>
<td>PMIX_STATUS</td>
<td>Status code pmix_status_t.</td>
</tr>
<tr>
<td>PMIX_VALUE</td>
<td>Value (pmix_value_t).</td>
</tr>
<tr>
<td>PMIX_PROC</td>
<td>Process (pmix_proc_t).</td>
</tr>
<tr>
<td>PMIX_APP</td>
<td>Application context.</td>
</tr>
<tr>
<td>PMIX_INFO</td>
<td>Info object.</td>
</tr>
<tr>
<td>PMIX_PDATA</td>
<td>Pointer to data.</td>
</tr>
<tr>
<td>PMIX_BUFFER</td>
<td>Buffer.</td>
</tr>
<tr>
<td>PMIX_BYTE_OBJECT</td>
<td>Byte object (pmix_byte_object_t).</td>
</tr>
<tr>
<td>PMIX_KVAL</td>
<td>Key/value pair.</td>
</tr>
<tr>
<td>PMIX_PERSIST</td>
<td>Persistance (pmix_persistence_t).</td>
</tr>
<tr>
<td>PMIX_POINTER</td>
<td>Pointer to an object (void*).</td>
</tr>
<tr>
<td>PMIX_SCOPE</td>
<td>Scope (pmix_scope_t).</td>
</tr>
<tr>
<td>PMIX_DATA_RANGE</td>
<td>Range for data (pmix_data_range_t).</td>
</tr>
<tr>
<td>PMIX_COMMAND</td>
<td>PMIx command code (used internally).</td>
</tr>
<tr>
<td>PMIX_INFO_DIRECTIVES</td>
<td>Directives flag for pmix_info_t (pmix_info_directives_t).</td>
</tr>
<tr>
<td>PMIX_DATA_TYPE</td>
<td>Data type code (pmix_data_type_t).</td>
</tr>
<tr>
<td>PMIX_PROC_STATE</td>
<td>Process state (pmix_proc_state_t).</td>
</tr>
<tr>
<td>PMIX_PROC_INFO</td>
<td>Process information (pmix_proc_info_t).</td>
</tr>
<tr>
<td>PMIX_DATA_ARRAY</td>
<td>Data array (pmix_data_array_t).</td>
</tr>
<tr>
<td>PMIX_PROC_RANK</td>
<td>Process rank (pmix_rank_t).</td>
</tr>
</tbody>
</table>
3.4 General Callback Functions

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

3.4.1 Release Callback Function

Summary
The `pmix_release_cbfunc_t` is used by the `pmix_modex_cbfunc_t` and `pmix_info_cbfunc_t` operations to indicate that the callback data may be reclaimed/freed by the caller.
3.4.2 Op Callback Function

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

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

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

3.4.3 Value Callback Function

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

```
typedef void (*pmix_value_cbfunc_t)(pmix_status_t status,
                                      pmix_value_t *kv, void *cbdata);
```
3.4.4 Info Callback Function

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

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

Description
The `status` indicates if requested data was found or not. An array of `pmix_info_t` will contain the key/value pairs.
3.4.5 Handler registration callback function

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

Format

```c
typedef void (*pmix_hdlr_reg_cbfunc_t)(pmix_status_t status,
                                       size_t refid,
                                       void *cbdata);
```

IN  status
    PMIX_SUCCESS or an appropriate error constant (pmix_status_t)

IN  refid
    reference identifier assigned to the handler by PMIx, used to deregister the handler (size_t)

IN  cbdata
    object provided to the registration call (pointer)

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

3.5 PMIx Datatype Value String Representations

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

Summary
String representation of a pmix_status_t.

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

Summary
String representation of a pmix_proc_state_t.

```c
const char*
PMIx_Proc_state_string(pmix_proc_state_t state);
```
**Summary**
String representation of a `pmix_scope_t`.

```c
PMIx v2.0
const char*
PMIx_Scope_string(pmix_scope_t scope);
```

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

```c
PMIx v2.0
const char*
PMIx_Persistence_string(pmix_persistence_t persist);
```

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

```c
PMIx v2.0
const char*
PMIx_Data_range_string(pmix_data_range_t range);
```

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

```c
PMIx v2.0
const char*
PMIx_Info_directives_string(pmix_info_directives_t directives);
```

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

```c
PMIx v2.0
const char*
PMIx_Data_type_string(pmix_data_type_t type);
```
Summary
String representation of a pmix_alloc_directive_t.

PMIx v2.0

const char*
PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);

Summary
String representation of a pmix_iof_channel_t.

PMIx v3.0

const char*
PMIx_IOF_channel_string(pmix_iof_channel_t channel);

Summary
String representation of a pmix_job_state_t.

PMIx v4.0

const char*
PMIx_Job_state_string(pmix_job_state_t state);

Summary
String representation of a PMIx attribute.

PMIx v4.0

const char*
PMIx_Get_attribute_string(char *attributename);

Summary
Return the PMIx attribute name corresponding to the given attribute string.

PMIx v4.0

const char*
PMIx_Get_attribute_name(char *attributestring);
Summary
String representation of a \texttt{pmix\_link\_state\_t}.

\begin{verbatim}
PMIx v4.0
\texttt{const char*}
\PMIx\_Link\_state\_string(pmix\_link\_state\_t state);
\end{verbatim}

Summary
String representation of a \texttt{pmix\_device\_type\_t}.

\begin{verbatim}
PMIx v4.0
\texttt{const char*}
\PMIx\_Device\_type\_string(pmix\_device\_type\_t type);
\end{verbatim}
CHAPTER 4

Client Initialization and Finalization

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

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

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

4.1 PMIx_Initialized

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

Format

PMIx v1.0

int PMIx_Initialized(void)

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

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

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

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

Format
PMIx v1.0
const char* PMIx_Get_version(void)

Description
Get the PMIx version string. Note that the provided string is statically defined and must not be free’d.

4.3 PMIx_Init

Summary
Initialize the PMIx client library

Format
PMIx v1.2
pmix_status_t PMIx_Init(pmix_proc_t *proc,
                          pmix_info_t info[], size_t ninfo)

INOUT proc
proc structure (handle)

IN info
Array of pmix_info_t structures (array of handles)

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

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

- **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)
  - Disable legacy UNIX socket (usock) support. If the library supports Unix socket connections, this attribute may be supported for disabling it.

- **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)
  - POSIX `mode_t` (9 bits valid). If the library supports socket connections, this attribute may be supported for setting the socket mode.

- **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)
  - Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport. If the library supports multiple methods for clients to connect to servers, this attribute may be supported for disabling all but one of them.

- **PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)
  - If provided, directs that the TCP Uniform Resource Identifier (URI) be reported and indicates the desired method of reporting: ‘-’ for stdout, ‘+’ for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

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

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

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

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

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

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

- **PMIX_EXTERNAL_PROGRESS** "pmix.evext" (bool)
The host shall progress the PMIx library via calls to `PMIx_Progress`

```
PMIX_EVENT_BASE "pmix.evbase" (void*)
```

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

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

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

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

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

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

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

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

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

```
PMIX_MODEL_AFFINITY_POLICY "pmix.mdl.tap" (char*)
```
Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread located on cpu close to master thread), "spread" (threads load-balanced across available cpus).
Description
Initialize the PMIx client, returning the process identifier assigned to this client’s application in the
provided `pmix_proc_t` struct. Passing a value of `NULL` for this parameter is allowed if the user
wishes solely to initialize the PMIx system and does not require return of the identifier at that time.

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

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

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

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

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

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

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

4.3.2 Initialization attributes
The following attributes influence the behavior of `PMIx_Init`.

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

- `PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)`
  If provided, directs that the TCP URI be reported and indicates the desired method of
  reporting: ‘-’ for stdout, ‘+’ for stderr, or filename.
- `PMIX_TCP_URI "pmix.tcp.uri" (char*)`
The URI of the PMIx server to connect to, or a file name containing it in the form of

    file:<name of file containing it>.

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

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

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

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

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

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

### 4.3.2.2 Programming model attributes

These attributes are associated with programming models.

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

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

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

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

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

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

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

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

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

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

Summary
Finalize the PMIx client library.

Format

PMIx_Finalize

\[
\text{pmix_status_t PMIx_Finalize(const pmix_info_t info[], size_t ninfo)}
\]

IN info
Array of pmix_info_t structures (array of handles)

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

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

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

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

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

4.4.1 Finalize attributes

The following attribute influences the behavior of PMIx_Finalize.

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

4.5 PMIx_Progress

Summary
Progress the PMIx library.
Format

PMIx v4.0

void

PMIx_Progress(void)

Description

Progress the PMIx library. Note that special care must be taken to avoid deadlocking in PMIx callback functions and acpAPI.
Applications may need to synchronize their operations at various points in their execution. Depending on a variety of factors (e.g., the programming model and where the synchronization point lies), the application may choose to execute the operation using PMIx. This is particularly useful in situations where communication by other means is not yet available since PMIx relies on the host environment’s infrastructure for such operations.

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

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

## 5.1 PMIx_Fence

### Summary

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

### Format

```
PMIx v1.0

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

**IN** `procs`
Array of `pmix_proc_t` structures (array of handles)

**IN** `nprocs`
Number of elements in the `procs` array (integer)

**IN** `info`
Array of info structures (array of handles)

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

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

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

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

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

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

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

Optional Attributes

The following attributes are optional for PMIx implementations:

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

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

The following attributes are optional for host environments:

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

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.
Description
Passing a NULL pointer as the proc parameter indicates that the fence is to span all processes in
the client’s namespace. Each provided pmix_proc_t struct can pass PMIX_RANK_WILDCARD
to indicate that all processes in the given namespace are participating.

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

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

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

5.2 PMIx_Fence_nb

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

PMIx v1.0

C

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

IN  procs
   Array of pmix_proc_t structures (array of handles)

IN  nprocs
   Number of elements in the procs array (integer)

IN  info
   Array of info structures (array of handles)

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

IN  cbfunc
   Callback function (function reference)

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

Returns one of the following:

- PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
  will be returned in the provided cbfunc. Note that the library must not invoke the callback
  function prior to returning from the API.

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

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called.

Required Attributes

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

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

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

**Optional Attributes**

The following attributes are optional for PMIx implementations:

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

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

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

**Description**

Nonblocking version of the `PMIx_Fence` routine. See the `PMIx_Fence` description for further details.

### 5.2.1 Fence-related attributes

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

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

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

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

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

Format

PMIx v1.0

C

pmix_status_t
PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,
const pmix_info_t info[], size_t ninfo,
pmix_value_t **val);

C

IN proc
Process identifier - a NULL value may be used in place of the caller’s ID (handle)
IN key
Key to retrieve (pmix_key_t)
IN info
Array of info structures (array of handles)
IN ninfo
Number of elements in the info array (integer)
OUT val
value (handle)

Returns one of the following:

- **PMIX_SUCCESS** The requested data has been returned in the manner requested (i.e., in a provided static memory location )
- **PMIX_ERR_BAD_PARAM** A bad parameter was passed to the function call - e.g., the request included the PMIX_GET_STATIC_VALUES directive, but the provided storage location was NULL
- **PMIX_ERR_EXISTS_OUTSIDE_SCOPE** The requested key exists, but was posted in a scope (see Section 7.1.1.1) that does not include the requester.
- **PMIX_ERR_NOT_FOUND** The requested data was not available.
- a non-zero PMIx error constant indicating a reason for the request’s failure.

Required Attributes

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

- **PMIX_OPTIONAL** "pmix.optional" (bool)
  Look only in the client’s local data store for the requested value - do not request data from the PMIx server if not found.
- **PMIX_IMMEDIATE** "pmix.immediate" (bool)
Specified operation should immediately return an error from the PMIx server if the requested data cannot be found - do not request it from the host RM.

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

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

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

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

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

**PMIX_GET_STATIC_VALUES** "pmix.get.static" (bool)
Request that the data be returned in the provided storage location. The caller is responsible for destructing the pmix_value_t using the PMIX_VALUE_DESTRUCT macro when done.

**PMIX_GET_POINTER_VALUES** "pmix.get.pntrs" (bool)
Request that any pointers in the returned value point directly to values in the key-value store. The user must not release any returned data pointers.

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

The following attributes are optional for host environments:

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

Description

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

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

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

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

### 5.3.1 PMIx_Get_nb

Summary

Nonblocking **PMIx_Get** operation.
Format

```
PMIx v1.0
```

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

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

**IN**  
key  
Key to retrieve (string)

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

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

**IN**  
`cbfunc`  
Callback function (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called.

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

- **PMIX_SUCCESS** The requested data has been returned.

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

- **PMIX_ERR_NOT_FOUND** The requested data was not available.

- a non-zero PMIx error constant indicating a reason for the request’s failure.

---

**Required Attributes**

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

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

Look only in the client’s local data store for the requested value - do not request data from the PMIx server if not found.
**PMIX_IMMEDIATE** "pmix.immediate" (bool)
Specified operation should immediately return an error from the PMIx server if the requested data cannot be found - do not request it from the host RM.

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

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

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

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

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

**PMIX_GET_POINTER_VALUES** "pmix.get.pntrs" (bool)
Request that any pointers in the returned value point directly to values in the key-value store. The user must not release any returned data pointers.

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

The following attributes are required for host environments that support this operation:
PMIX_WAIT  "pmix.wait" (int)
Callers request that the PMIx server wait until at least the specified number of values are
found (a value of zero indicates all and is the default).

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

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

Description
The callback function will be executed once the retrieval rules of Chapters 6 or 7 are met. See
PMIx_Get for a full description. Note that the non-blocking form of this function cannot support
the PMIX_GET_STATIC_VALUES attribute as the user cannot pass in the required pointer to
storage for the result.

5.3.2 Retrieval attributes

The following attributes are defined for use by retrieval APIs:

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

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

PMIX_GET_POINTER_VALUES  "pmix.get.pntrs" (bool)
Request that any pointers in the returned value point directly to values in the key-value store.
The user must not release any returned data pointers.

PMIX_GET_STATIC_VALUES  "pmix.get.static" (bool)
Request that the data be returned in the provided storage location. The caller is responsible
for destructing the pmix_value_t using the PMIX_VALUE_DESTRUCT macro when
done.

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

```plaintext
PMIX_DATA_SCOPE "pmix.scope" (pmix_scope_t)
Scope of the data to be searched in a PMIx_Get call.
PMIX_TIMEOUT "pmix.timeout" (int)
Time in seconds before the specified operation should time out (zero indicating infinite) and return the `PMIX_ERR_TIMEOUT` error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.
PMIX_WAIT "pmix.wait" (int)
Caller requests that the PMIx server wait until at least the specified number of values are found (a value of zero indicates all and is the default).
```

### 5.4 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.

5.4.1 PMIx.Resolve_peers

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

**Format**

```
Pmx_status_t
PMIx.Resolve_peers(const char *nodename,
                   const pmix_namespace_t nspace,
                   pmix_proc_t **procs, size_t *nprocs);
```

**Parameters**

- **IN nodename**
  Name of the node to query - **NULL** can be used to denote the current local node (string)
- **IN nspace**
  namespace (string)
- **OUT procs**
  Array of process structures (array of handles)
- **OUT nprocs**
  Number of elements in the `procs` array (integer)

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

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

5.4.2 PMIx.Resolve_nodes

**Summary**
Return a list of nodes hosting processes within the given namespace.
```
1  Format
2  PMIx v1.0
3    pmix_status_t
4    PMIx_Resolve_nodes(const char *nspace, char **nodelist);

5  IN  nspace
6      Namespace (string)
7  OUT nodelist
8      Comma-delimited list of nodenames (string)

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

10 Description
11    Given a nspace, return the list of nodes hosting processes within that namespace. The returned
12    string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the
13    string when done with it.

13  5.4.3 PMIx_Query_info

15 Summary
16    Query information about the system in general.

17  Format
18  PMIx v4.0
19    pmix_status_t
20    PMIx_Query_info(pmix_query_t queries[], size_t nqueries,
21                         pmix_info_t *info[], size_t *ninfo);

20  IN  queries
21      Array of query structures (array of handles)
22  IN  nqueries
23      Number of elements in the queries array (integer)
24  INOUT info
25      Address where a pointer to an array of pmix_info_t containing the results of the query can
26      be returned (memory reference)
27  INOUT ninfo
28      Address where the number of elements in info can be returned (handle)

29  Returns one of the following:
30  • PMIX_SUCCESS All data was found and has been returned.
31  • PMIX_ERR_NOT_FOUND None of the requested data was available. The info array will be
32    NULL and ninfo zero.
```
• **PMIX_ERR_PARTIAL_SUCCESS** Some of the requested data was found. The *info* array shall contain an element for each query key that returned a value.

• **PMIX_ERR_NOT_SUPPORTED** The host RM does not support this function. The *info* array will be **NULL** and *ninfo* zero.

• a non-zero PMIx error constant indicating a reason for the request’s failure. The *info* array will be **NULL** and *ninfo* zero.

**Required Attributes**

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

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

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

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

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

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

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

**PMIX_PROCID** "pmix.procid" (pmix_proc_t)
Process identifier. Used as a key in PMIX_Get to retrieve the caller’s own process identifier in a portion of the program that doesn’t have access to the memory location in which it was originally stored (e.g., due to a call to PMIx_Init). The process identifier in the PMIx_Get call is ignored in this instance. In this context, specifies the process ID whose information is being requested - e.g., a query asking for the pmix_proc_info_t 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 - may be a numerical value expressed as a string, but is often an alphanumeric string carrying information solely of use to the system. Required to be unique within the scope of the host environment. Specifies the namespace of the process whose information is being requested. 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, starting from zero. Specifies the rank of the process whose information is being requested. 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. REQUIRED QUALIFIERS: one or more of PMIX_CLIENT_FUNCTIONS, PMIX_SERVER_FUNCTIONS, PMIX_TOOL_FUNCTIONS, and PMIX_HOST_FUNCTIONS.

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

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

PMIX_HOST_ATTRIBUTES  "pmix.hostattrs" (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 both 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, all provided attributes must be passed to the host SMS daemon for processing. The PMIx library is required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process making the request.
The following attributes are optional for host environments that support this operation:

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

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

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

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

**PMIX_QUERY_PROC_TABLE** "pmix.qry.ptable" (char*)
Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried.

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

**PMIX_QUERY_SPAWN_SUPPORT** "pmix.qry.spawn" (bool)
Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

**PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

**PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)
Return information on memory usage for the processes indicated in the qualifiers. OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of specific process(es) whose memory usage is being requested.

**PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
Report only average values for sampled information. NO QUALIFIERS.

**PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)
Report minimum and maximum values. NO QUALIFIERS.

**PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (char*)
String identifier of the allocation whose status is being requested. NO QUALIFIERS.

**PMIX_TIME_REMAINING** "pmix.time.remaining" (char*)
Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
OPTIONAL QUALIFIERS: **PMIX_NSPACE** of the namespace whose info is being requested (defaults to allocation containing the caller).

**PMIX_SERVER_URI** "pmix.srvr.uri" (char*)
URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.

**PMIX_CLIENT_AVG_MEMORY** "pmix.cl.mem.avg" (float)
Average Megabytes of memory used by client processes on node. OPTIONAL QUALIFIERS: **PMIX_HOSTNAME** or **PMIX_NODEID** (defaults to caller’s node).

**PMIX_DAEMON_MEMORY** "pmix.dmn.mem" (float)
Megabytes of memory currently used by the RM daemon on the node. OPTIONAL QUALIFIERS: **PMIX_HOSTNAME** or **PMIX_NODEID** (defaults to caller’s node).

**PMIX_QUERY_AUTHORIZATIONS** "pmix.qry.auths" (bool)
Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.

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

**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.

---

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

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

Information returned from **PMIx_Query_info** shall be locally cached so that retrieval by subsequent calls to **PMIx_Get**, **PMIx_Query_info**, or **PMIx_Query_info_nb** can succeed with minimal overhead. The local cache shall be checked prior to querying the PMIx server and/or the host environment. Queries that include the **PMIX_QUERY_REFRESH_CACHE** attribute shall bypass the local cache and retrieve a new value for the query, refreshing the values in the cache upon return.

5.4.4 **PMIx_Query_info_nb**

**Summary**

Query information about the system in general.

**Format**

```c
PMIx v2.0

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

**IN queries**

Array of query structures (array of handles)

**IN nqueries**

Number of elements in the `queries` array (integer)

**IN cbfunc**

Callback function `pmix_info_cbfunc_t` (function reference)

**IN cbdata**

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

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.

- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed.

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

- **PMIX_SUCCESS** All data was found and has been returned.
• **PMIX_ERR_NOT_FOUND** None of the requested data was available. The info array will be NULL and ninfo zero.

• **PMIX_ERR_PARTIAL_SUCCESS** Some of the requested data was found. The info array shall contain an element for each query key that returned a value.

• **PMIX_ERR_NOT_SUPPORTED** The host RM does not support this function. The info array will be NULL and ninfo zero.

• a non-zero PMIx error constant indicating a reason for the request’s failure. The info array will be NULL and ninfo zero.

## Required Attributes

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

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

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

- **PMIX_JOB_INFO** "pmix.job.info" (bool)
  - Return information regarding the job realm corresponding to the namespace in the target process’ identifier.

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

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

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

- **PMIX_PROCID** "pmix.procid" (pmix_proc_t)
Process identifier. Used as a key in \texttt{PMIx\_Get} to retrieve the caller's own process identifier in a portion of the program that doesn't have access to the memory location in which it was originally stored (e.g., due to a call to \texttt{PMIx\_Init}). The process identifier in the \texttt{PMIx\_Get} call is ignored in this instance. In this context, specifies the process ID whose information is being requested - e.g., a query asking for the \texttt{pmix\_proc\_info\_t} of a specified process. Only required when the request is for information on a specific process.

\texttt{PMIX\_NAMESPACE} "pmix.nspace" (char*)

Namespace of the job - may be a numerical value expressed as a string, but is often an alphanumeric string carrying information solely of use to the system. Required to be unique within the scope of the host environment. Specifies the namespace of the process whose information is being requested. Must be accompanied by the \texttt{PMIX\_RANK} attribute. Only required when the request is for information on a specific process.

\texttt{PMIX\_RANK} "pmix.rank" (pmix\_rank\_t)

Process rank within the job, starting from zero. Specifies the rank of the process whose information is being requested. Must be accompanied by the \texttt{PMIX\_NAMESPACE} attribute. Only required when the request is for information on a specific process.

\texttt{PMIX\_QUERY\_ATTRIBUTE\_SUPPORT} "pmix.qry.attrs" (bool)

Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or more of \texttt{PMIX\_CLIENT\_FUNCTIONS, PMIX\_SERVER\_FUNCTIONS, PMIX\_TOOL\_FUNCTIONS,} and \texttt{PMIX\_HOST\_FUNCTIONS}.

\texttt{PMIX\_CLIENT\_ATTRIBUTES} "pmix.client.attrs" (bool)

Request attributes supported by the PMIx client library.

\texttt{PMIX\_SERVER\_ATTRIBUTES} "pmix.srvr.attrs" (bool)

Request attributes supported by the PMIx server library.

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

Request attributes supported by the host environment.

\texttt{PMIX\_TOOL\_ATTRIBUTES} "pmix.setup.env" (bool)

Request attributes supported by the PMIx tool library functions.

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

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

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

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

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

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

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

- **PMIX_QUERY_PROC_TABLE** "pmix.qry.ptable" (char*)
  - Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried.

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

- **PMIX_QUERY_SPAWN_SUPPORT** "pmix.qry.spawn" (bool)
  - Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

- **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
  - Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

- **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)
  - Return information on memory usage for the processes indicated in the qualifiers. OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of specific process(es) whose memory usage is being requested.

- **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
  - Report only average values for sampled information. NO QUALIFIERS.

- **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)
  - Report minimum and maximum values. NO QUALIFIERS.

- **PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (char*)
String identifier of the allocation whose status is being requested. NO QUALIFIERS.

**PMIX_TIME_REMAINING** "pmix.time.remaining" (char*)
Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
OPTIONAL QUALIFIERS: **PMIX_NSPACE** of the namespace whose info is being requested (defaults to allocation containing the caller).

**PMIX_SERVER_URI** "pmix.srvr.uri" (char*)
URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.

**PMIX_CLIENT_AVG_MEMORY** "pmix.cl.mem.avg" (float)
Average Megabytes of memory used by client processes on node. OPTIONAL QUALIFIERS: **PMIX_HOSTNAME** or **PMIX_NODEID** (defaults to caller’s node).

**PMIX_DAEMON_MEMORY** "pmix.dmn.mem" (float)
Megabytes of memory currently used by the RM daemon on the node. OPTIONAL QUALIFIERS: **PMIX_HOSTNAME** or **PMIX_NODEID** (defaults to caller’s node).

**PMIX_QUERY_AUTHORIZATIONS** "pmix.qry.auths" (bool)
Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.

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

**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.

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

**5.4.5 Query-specific constants**

**PMIX_QUERY_PARTIAL_SUCCESS** Some, but not all, of the requested information was returned.
5.4.6 Query attributes

Attributes used to direct behavior of the PMIx_Query_info APIs.

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

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

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

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

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

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

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

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

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

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

PMIX_QUERY_PROC_TABLE "pmix qry ptable" (char*)
Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
PMIX_NSPACE indicating the namespace whose process table is being queried.

PMIX_QUERY_LOCAL_PROC_TABLE "pmix qry lptable" (char*)
Returns a \((\text{pmix\_data\_array\_t})\) array of \(\text{pmix\_proc\_info\_t}\), one entry for each process in the specified namespace executing on the same node as the requester, ordered by process job rank. REQUIRED QUALIFIER: \text{PMIX\_NSPACE} indicating the namespace whose local process table is being queried. OPTIONAL QUALIFIER: \text{PMIX\_hostname} indicating the host whose local process table is being queried. By default, the query assumes that the host upon which the request was made is to be used.

\text{PMIX\_QUERY\_AUTHORIZATIONS} \ "\text{pmix.qry.auths}" (bool)
Return operations the PMIx tool is authorized to perform. NO QUALIFIERS.

\text{PMIX\_QUERY\_SPAWN\_SUPPORT} \ "\text{pmix.qry.spawn}" (bool)
Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.

\text{PMIX\_QUERY\_DEBUG\_SUPPORT} \ "\text{pmix.qry.debug}" (bool)
Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.

\text{PMIX\_QUERY\_MEMORY\_USAGE} \ "\text{pmix.qry.mem}" (bool)
Return information on memory usage for the processes indicated in the qualifiers.
OPTIONAL QUALIFIERS: \text{PMIX\_NSPACE} and \text{PMIX\_RANK}, or \text{PMIX\_PROCID} of specific process(es) whose memory usage is being requested.

\text{PMIX\_QUERY\_LOCAL\_ONLY} \ "\text{pmix.qry.local}" (bool)
Constrain the query to local information only. NO QUALIFIERS.

\text{PMIX\_QUERY\_REPORT\_AVG} \ "\text{pmix.qry.avg}" (bool)
Report only average values for sampled information. NO QUALIFIERS.

\text{PMIX\_QUERY\_REPORT\_MINMAX} \ "\text{pmix.qry.minmax}" (bool)
Report minimum and maximum values. NO QUALIFIERS.

\text{PMIX\_QUERY\_ALLOC\_STATUS} \ "\text{pmix.query.alloc}" (char*)
String identifier of the allocation whose status is being requested. NO QUALIFIERS.

\text{PMIX\_TIME\_REMAINING} \ "\text{pmix.time.remaining}" (char*)
Query number of seconds (\text{uint32\_t}) remaining in allocation for the specified namespace.
OPTIONAL QUALIFIERS: \text{PMIX\_NSPACE} of the namespace whose info is being requested (defaults to allocation containing the caller).

\text{PMIX\_QUERY\_ATTRIBUTE\_SUPPORT} \ "\text{pmix.qry.attrs}" (bool)
Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or more of \text{PMIX\_CLIENT\_FUNCTIONS}, \text{PMIX\_SERVER\_FUNCTIONS}, \text{PMIX\_TOOL\_FUNCTIONS}, and \text{PMIX\_HOST\_FUNCTIONS}.

\text{PMIX\_QUERY\_NUM\_PSETS} \ "\text{pmix.qry.psetnum}" (size\_t)
Return the number of process sets defined in the specified range (defaults to \text{PMIX\_RANGE\_SESSION}).

\text{PMIX\_QUERY\_PSET\_NAMES} \ "\text{pmix.qry.psets}" (pmix\_data\_array\_t*)
Return a \text{pmix\_data\_array\_t} containing an array of strings of the process set names defined in the specified range (defaults to \text{PMIX\_RANGE\_SESSION}).

\text{PMIX\_QUERY\_PSET\_MEMBERSHIP} \ "\text{pmix.qry.pmems}" (pmix\_data\_array\_t*)
Return an array of \text{pmix\_proc\_t} containing the members of the specified process set.

\text{PMIX\_QUERY\_AVAIL\_SERVERS} \ "\text{pmix.qry.asrvrs}" (pmix\_data\_array\_t*)
Return an array of `pmix_info_t`, each element itself containing a `PMIX_SERVER_INFO_ARRAY` entry holding all available data for a server on this node to which the caller might be able to connect.

```
PMIX_SERVER_INFO_ARRAY "pmix.srv.arr" (pmix_data_array_t)
```

Array of `pmix_info_t` about a given server, starting with its `PMIX_NSPACE` and including at least one of the rendezvous-required pieces of information.

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

```
PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)
```

Total available physical memory on a node. OPTIONAL QUALIFIERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller’s node).

```
PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float)
```

Megabytes of memory currently used by the RM daemon on the node. OPTIONAL QUALIFIERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller’s node).

```
PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float)
```

Average Megabytes of memory used by client processes on node. OPTIONAL QUALIFIERS: `PMIX_HOSTNAME` or `PMIX_NODEID` (defaults to caller’s node).

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

```
PMIX_CLIENT_FUNCTIONS "pmix.client.fns" (bool)
```

Request a list of functions supported by the PMIx client library.

```
PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
```

Request attributes supported by the PMIx client library.

```
PMIX_SERVER_FUNCTIONS "pmix.srvr.fns" (bool)
```

Request a list of functions supported by the PMIx server library.

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

Request attributes supported by the PMIx server library.

```
PMIX_HOST_FUNCTIONS "pmix.srvr.fns" (bool)
```

Request a list of functions supported by the host environment.

```
PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
```

Request attributes supported by the host environment.

```
PMIX_TOOL_FUNCTIONS "pmix.tool.fns" (bool)
```

Request a list of functions supported by the PMIx tool library.

```
PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
```

Request attributes supported by the PMIx tool library functions.

### 5.4.7 Query Structure

The `pmix_query_t` structure is used by the `PMIX_Query_info` APIs to describe a single query operation.
typedef struct pmix_query {
    char **keys;
    pmix_info_t *qualifiers;
    size_t nqual;
} pmix_query_t;

where:

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

5.4.7.1 Query structure support macros

The following macros are provided to support the pmix_query_t structure.

Initialize the query structure
Initialize the pmix_query_t fields

PMIx v2.0

PMIX_QUERY_CONSTRUCT(m)

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

Destruct the query structure
Destruct the pmix_query_t fields

PMIx v2.0

PMIX_QUERY_DESTRUCT(m)

IN m
Pointer to the structure to be destructed (pointer to pmix_query_t)
Create a query array
Allocate and initialize an array of `pmix_query_t` structures

```c
PMIx v2.0
PMIX_QUERY_CREATE(m, n)

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

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

Free a query structure
Release a `pmix_query_t` structure

```c
PMIx v4.0
PMIX_QUERY_RELEASE(m)

IN m
Pointer to a `pmix_query_t` structure (handle)
```

Free a query array
Release an array of `pmix_query_t` structures

```c
PMIx v2.0
PMIX_QUERY_FREE(m, n)

IN m
Pointer to the array of `pmix_query_t` structures (handle)

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

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

```c
PMIx v2.2
PMIX_QUERY_QUALIFIERS_CREATE(m, n)

IN m
Pointer to the `pmix_query_t` structure (handle)

IN n
Number of qualifiers to be allocated (size_t)```
5.5 Using Get vs Query

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

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

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

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

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

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

5.6 Accessing attribute support information

Information as to which attributes are supported by either the PMIx implementation or its host environment can be obtained via the `PMIx_Query_info` APIs. The `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute must be listed as the first entry in the `keys` field of the `pmix_query_t` structure, followed by the name of the function whose attribute support is being requested - support for multiple functions can be requested simultaneously by simply adding...
the function names to the array of keys. Function names must be given as user-level API names -
e.g., “PMIx_Get”, “PMIx_server_setup_application”, or “PMIx_tool_attach_to_server”.

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

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

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

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

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

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

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

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

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

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

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.
Figure 5.1.: Returned information hierarchy for attribute support request
CHAPTER 6

Reserved Keys

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

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

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

6.1 Data realms

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

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

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

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

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

---

**Advice to users**

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
session. This is required as many environments only guarantee unique namespaces within a
session, and not across sessions.

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

### 6.1.1 Session realm attributes

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

Note that the *proc* argument of **PMIx_Get** is ignored when referencing session-related
information.

Session-level information includes the following attributes:

- **PMIX_SESSION_ID** "pmix.session.id" (uint32_t)
  
  Session identifier assigned by the scheduler.

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

- **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)
Maximum number of process that can be simultaneously executing in a session. Note that this attribute is equivalent to the `PMIX_MAX_PROCS` attribute for the `session` realm - it is included in the PMIx Standard for historical reasons.

`PMIX_TMPDIR "pmix.tmpdir" (char*)`
Full path to the top-level temporary directory assigned to the session.

`PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)`
Resource Manager will cleanup assigned temporary directory trees.

`PMIX_HOSTNAME_KEEP_FQDN "pmix.fqdn" (bool)`
Fully Qualified Domain Names (FQDNs) are being retained by the PMIx library.

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

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

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

The remaining session-related information can only be retrieved by including the `PMIX_SESSION_INFO` attribute in the `info` array passed to `PMIx_Get`:

`PMIX_ALLOCATED_NODELIST "pmix.alist" (char*)`
Comma-delimited list or regular expression of all nodes in the specified realm regardless of whether or not they currently host processes. Defaults to the `job` realm.

`PMIX_NUM_ALLOCATED_NODES "pmix.num.anodes" (uint32_t)`
Number of nodes in the specified realm regardless of whether or not they currently host processes. Defaults to the `job` realm.

`PMIX_MAX_PROCS "pmix.max.size" (uint32_t)`
Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Defaults to the `job` realm.

`PMIX_NODE_LIST "pmix.nlist" (char*)`
Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults to the `job` realm.

`PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)`
Maximum number of processes that can simultaneously be executing in the specified realm. Note that this attribute is the equivalent to `PMIX_MAX_PROCS` - it is included in the PMIx Standard for historical reasons. Defaults to the `job` realm.

`PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)`
Number of nodes currently hosting processes in the specified realm. Defaults to the `job` realm.

`PMIX_NODE_MAP "pmix.nmap" (char*)`
Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the `job` realm.
**PMIX_NODE_MAP_RAW** "pmix.nmap.raw" (char*)
Comma-delimited list of nodes containing procs within the specified realm. Defaults to the job realm.

**PMIX_PROC_MAP** "pmix.pmap" (char*)
Regular expression describing processes on each node in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the job realm.

**PMIX_PROC_MAP_RAW** "pmix.pmap.raw" (char*)
Semi-colon delimited list of strings, each string containing a comma-delimited list of ranks on the corresponding node within the specified realm. Defaults to the job realm.

**PMIX_ANL_MAP** "pmix.anlmap" (char*)
Process map equivalent to **PMIX_PROC_MAP** expressed in Argonne National Laboratory’s PMI-1/PMI-2 notation. Defaults to the job realm.

### 6.1.2 Job realm attributes

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

Job-level information includes the following attributes:

**PMIX_NSPACE** "pmix.nspace" (char*)
Namespace of the job - may be a numerical value expressed as a string, but is often an alphanumeric string carrying information solely of use to the system. Required to be unique within the scope of the host environment.

**PMIX_JOBID** "pmix.jobid" (char*)
Job identifier assigned by the scheduler to the specified job - may be identical to the namespace, but is often a numerical value expressed as a string (e.g., "12345.3").

**PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
Starting global rank of the specified job.

**PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Defaults to the job realm. In this context, this is the maximum number of processes that can be simultaneously executed in the specified job, which may be a subset of the number allocated to the overall session.

**PMIX_NUM_SLOTS** "pmix.num.slots" (uint32_t)
Maximum number of processes that can simultaneously be executing in the specified realm. Note that this attribute is the equivalent to **PMIX_MAX_PROCS** - it is included in the PMIx Standard for historical reasons. Defaults to the job realm. In this context, this is the
maximum number of process that can be simultaneously executing within the specified job, which may be a subset of the number allocated to the overall session. Jobs may reserve a subset of their assigned maximum processes for dynamic operations such as \texttt{PMIx\_Spawn}.

\textbf{PMIX\_NUM\_NODES } "pmix.num.nodes" (\texttt{uint32\_t})

Number of nodes currently hosting processes in the specified realm. Defaults to the \textit{job} realm. In this context, this is the number of nodes currently hosting processes in the specified job, which may be a subset of the nodes allocated to the overall session. Jobs may reserve a subset of their assigned nodes for dynamic operations such as \texttt{PMIx\_Spawn} - i.e., not all nodes may have executing processes from this job at a given point in time.

\textbf{PMIX\_NODE\_MAP } "pmix.nmap" (\texttt{char\*})

Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the \textit{job} realm. In this context, this is the regular expression of nodes currently hosting processes in the specified job.

\textbf{PMIX\_NODE\_LIST } "pmix.nlist" (\texttt{char\*})

Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults to the \textit{job} realm. In this context, this is the comma-delimited list of nodes currently hosting processes in the specified job.

\textbf{PMIX\_PROC\_MAP } "pmix.pmap" (\texttt{char\*})

Regular expression describing processes on each node in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the \textit{job} realm. In this context, this is the regular expression describing processes on each node in the specified job.

\textbf{PMIX\_ANL\_MAP } "pmix.anlmap" (\texttt{char\*})

Process map equivalent to \texttt{PMIX\_PROC\_MAP} expressed in Argonne National Laboratory’s PMI-1/PMI-2 notation. Defaults to the \textit{job} realm. In this context, this is the process mapping in Argonne National Laboratory’s PMI-1/PMI-2 notation of the processes in the specified job.

\textbf{PMIX\_CMD\_LINE } "pmix.cmd.line" (\texttt{char\*})

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

\textbf{PMIX\_NSDIR } "pmix.nsdir" (\texttt{char\*})

Full path to the temporary directory assigned to the specified job, under \texttt{PMIX\_TMPDIR}.

\textbf{PMIX\_JOB\_SIZE } "pmix.job.size" (\texttt{uint32\_t})

Total number of processes in the specified job across all contained applications. Note that this value can be different from \texttt{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 \textit{job} on-the-fly. In the latter case, PMIx events may be used to notify processes within the job that the job size has changed.

\textbf{PMIX\_JOB\_NUM\_APPS } "pmix.job.napps" (\texttt{uint32\_t})

Number of applications in the specified job.
6.1.3 Application realm attributes

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

Application-level information includes the following attributes:

- **PMIX_APPNUM** "pmix.appnum" (*uint32_t*)
  The application number within the job in which the specified process is a member.

- **PMIX_NUM_NODES** "pmix.num.nodes" (*uint32_t*)
  Number of nodes currently hosting processes in the specified realm. Defaults to the *job* realm. In this context, this is the number of nodes currently hosting processes in the specified application, which may be a subset of the nodes allocated to the overall session.

- **PMIX_APPLDR** "pmix.aldr" (*pmix_rank_t*)
  Lowest rank in the specified application.

- **PMIX_APP_SIZE** "pmix.app.size" (*uint32_t*)
  Number of processes in the specified application, regardless of their execution state - i.e., this number may include processes that either failed to start or have already terminated.

- **PMIX_APP_ARGV** "pmix.app.argv" (*char*)
  Consolidated argv passed to the spawn command for the given application (e.g., "./myapp arg1 arg2 arg3”).

- **PMIX_MAX_PROCS** "pmix.max.size" (*uint32_t*)
  Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Defaults to the *job* realm. In this context, this is the maximum number of processes that can be executed in the specified application, which may be a subset of the number allocated to the overall session and job.

- **PMIX_NUM_SLOTS** "pmix.num.slots" (*uint32_t*)
  Maximum number of processes that can simultaneously be executing in the specified realm. Note that this attribute is the equivalent to `PMIX_MAX_PROCS` - it is included in the PMIx Standard for historical reasons. Defaults to the *job* realm. In this context, this is the number of slots assigned to the specified application, which may be a subset of the slots allocated to the overall session and job.

- **PMIX_NODE_MAP** "pmix.nmap" (*char*)
  Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the *job* realm. In this context, this is the regular expression of nodes currently hosting processes in the specified application.
**PMIX_NODE_LIST** "pmix.nlist" (char*)
- Comma-delimited list of nodes currently hosting processes in the specified realm. Defaults to the *job* realm. In this context, this is the comma-delimited list of nodes currently hosting processes in the specified application.

**PMIX_PROC_MAP** "pmix.pmap" (char*)
- Regular expression describing processes on each node in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the *job* realm. In this context, this is the regular expression describing processes on each node in the specified application.

**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.

### 6.1.4 Process realm attributes

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

Process-level information includes the following attributes:

**PMIX_APPNUM** "pmix.appnum" (uint32_t)
- The application number within the job in which the specified process is a member.

**PMIX_RANK** "pmix.rank" (pmix_rank_t)
- Process rank within the job, starting from zero.

**PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)
- Rank of the specified process spanning across all jobs in this session, starting with zero. Note that no ordering of the jobs is implied when computing this value. As jobs can start and end at random times, this is defined as a continually growing number - i.e., it is not dynamically adjusted as individual jobs and processes are started or terminated.

**PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)
- Rank of the specified process within its application.

**PMIX_PARENT_ID** "pmix.parent" (pmix_proc_t)
- Process identifier of the parent process of the specified process - typically used to identify the application process that caused the job containing the specified process to be spawned (e.g., the process that called **PMIx_Spawn**).

**PMIX_EXIT_CODE** "pmix.exit.code" (int)
- Exit code returned when the specified process terminated.

**PMIX_PROCID** "pmix.procid" (pmix_proc_t)
Process identifier. Used as a key in `PMIx_Get` to retrieve the caller's own process identifier in a portion of the program that doesn't have access to the memory location in which it was originally stored (e.g., due to a call to `PMix_Init`). The process identifier in the `PMix_Get` call is ignored in this instance.

**PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)
Rank of the specified process on its node - refers to the numerical location (starting from zero) of the process on its node when counting only those processes from the same job that share the node, ordered by their overall rank within that job.

**PMIX_NODE_RANK** "pmix.nrank" (uint16_t)
Rank of the specified process on its node spanning all jobs - refers to the numerical location (starting from zero) of the process on its node when counting all processes (regardless of job) that share the node, ordered by their overall rank within the job. The value represents a snapshot in time when the specified process was started on its node and is not dynamically adjusted as processes from other jobs are started or terminated on the node.

**PMIX_PACKAGE_RANK** "pmix.pkgrank" (uint16_t)
Rank of the specified process on the package where this process resides - refers to the numerical location (starting from zero) of the process on its package when counting only those processes from the same job that share the package, ordered by their overall rank within that job. Note that processes that are not bound to Processing Units (PUs) within a single specific package cannot have a package rank.

**PMIX_PROC_PID** "pmix.pid" (pid_t)
Operating system PID of specified process.

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

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

**PMIX_CUPSEET_BITMAP** "pmix.bitmap" (pmix_cpuset_t*)
Bitmap applied to the process upon launch.

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

**PMIX_SPAWNED** "pmix.spawned" (bool)
`true` if this process resulted from a call to `PMix_Spawn`. Lack of inclusion (i.e., a return status of `PMIX_ERR_NOT_FOUND`) corresponds to a value of `false` for this attribute.

**PMIX_REINCARNATION** "pmix.reinc" (uint32_t)
Number of times this process has been re-instantiated - i.e., a value of zero indicates that the process has never been restarted.

In addition, process-level information includes functional attributes directly associated with a process - for example, the process-related fabric attributes included in Section 14.3 or the distance attributes of Section 11.4.11.
6.1.5 Node realm keys

Information regarding the local node can be retrieved by directly requesting the node realm key in
the call to PMix_Get - the keys for node-related information are not shared across other realms.
The target process identifier will be ignored for keys that are not dependent upon it. Information
about a node other than the local node can be retrieved by specifying the PMIX_NODE_INFO
attribute in the info array along with either the PMIX_HOSTNAME or PMIX_NODEID qualifiers for
the node of interest.

Node-level information includes the following keys:

- **PMIX_HOSTNAME** "pmix.hname" (char*)
  - Name of the host, as returned by the gethostname utility or its equivalent.

- **PMIX_HOSTNAME_ALIASES** "pmix.alias" (char*)
  - Comma-delimited list of names by which the target node is known.

- **PMIX_NODEID** "pmix.nodeid" (uint32_t)
  - Node identifier expressed as the node’s index (beginning at zero) in an array of nodes within
    the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME
    of the node - i.e., users can interchangeably reference the same location using either the
    PMIX_HOSTNAME or corresponding PMIX_NODEID.

- **PMIX_NODE_SIZE** "pmix.node.size" (uint32_t)
  - Number of processes across all jobs that are executing upon the node.

- **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)
  - Total available physical memory on a node.

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

- **PMIX_LOCAL_PEERS** "pmix.lpeers" (char*)
  - Comma-delimited list of ranks that are executing on the local node within the specified
    namespace – shortcut for PMix_Resovle_peers for the local node.

- **PMIX_LOCAL_PROCS** "pmix.lprocs" (pmix_proc_t array)
  - Array of pmix Proc of all processes executing on the local node – shortcut for
    PMix_Resovle_peers for the local node and a NULL namespace argument. The process
    identifier is ignored for this attribute.

- **PMIX_LOCAL_LDR** "pmix.lldr" (pmix_rank_t)
  - Lowest rank within the specified job on the node (defaults to current node in absence of
    PMIX_HOSTNAME or PMIX_NODEID qualifier).

- **PMIX_LOCAL_CPUS** "pmix.lcpus" (pmix_data_array_t)
  - A pmix data array_t array of string representations of the PU binding bitmaps
    applied to each local peer on the caller’s node upon launch. Each string shall begin with the
    name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap string
    itself. The array shall be in the same order as the processes returned by
    PMIX_LOCAL_PEERS for that namespace.
PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)

Number of processes in the specified job or application realm on the caller’s node. Defaults to job realm unless the PMIX_APP_INFO and the PMIX_APPNUM qualifiers are given.

In addition, node-level information includes functional attributes directly associated with a node - for example, the node-related fabric attributes included in Section 14.3.

6.2 Retrieval rules for reserved keys

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

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

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

1. If the request included the PMIX_IMMEDIATE attribute, then the server will respond to the client with the PMIX_ERR_NOT_FOUND status.

2. If the host has provided the Direct Business Card Exchange (DBCX) module function interface (pmix_server_dmodex_req_fn_t), then the server shall pass the request to its host for servicing. The host is responsible for identifying a source of information on the specified namespace and retrieving it. The host is required to retrieve all of the information regarding the target namespace and return it to the requesting server in anticipation of follow-on requests. If the host cannot retrieve the namespace information, then it must respond with the PMIX_ERR_NOT_FOUND error constant unless the PMIX_TIMEOUT is given and reached (in which case, the host must respond with the PMIX_ERR_TIMEOUT constant).

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

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

6.2.1 Accessing information: examples

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

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

Example requests are shown below:

```c
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;

pmix_proc_t myproc, wildcard;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #slots in our session */
PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);

/* get the #nodes in our session */
PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

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

```c
pmix_info_t info[2];
pmix_value_t *value;
pmix_status_t rc;

pmix_proc_t myproc;
uint32_t sid;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #nodes in a different session */
sid = 12345;
PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
rc = PMIx_Get(NULL, PMIX_NUM_NODES, info, 2, &value);
```
6.2.1.2 Job-level information

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

```c
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, wildcard;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #apps in our job */
PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);

/* get the #nodes in our job */
PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

6.2.1.3 Application-level information

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

```c
pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t appsize, appnum;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #processes in our application */
rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
appsize = value->data.uint32;

/* get the #nodes in an application containing "otherproc".*/
* For this use-case, assume that we are in the first application
* and we want the #nodes in the second application - use the
* rank of the first process in that application, remembering
* that ranks start at zero */
PMIX_PROC_LOAD(&otherproc, myproc.nspace, appsize);

/* Since "otherproc" refers to a process in the second application,
* we can simply mark that we want the info for this key from the
* application realm */
PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);

/* alternatively, we can directly ask for the #nodes in
* the second application in our job, again remembering that
* application numbers start with zero. Since we are asking
* for application realm information about a specific appnum
* within our own namespace, the process identifier can be NULL */
appnum = 1;
PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
rc = PMIx_Get(NULL, PMIX_NUM_NODES, appinfo, 2, &value);

6.2.1.4 Process-level information

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

6.2.1.5 Node-level information

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

```c
pmix_info_t info[2];
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t nodeid;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #procs on our node */
```
rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);

/* get the #slots on another node */
PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
rc = PMIx_Get(NULL, PMIX_MAX_PROCS, info, 2, &value);

/* get the total #procs on the remote node - note that we don’t
 * actually need to include the "PMIX_NODE_INFO" attribute here,
 * but (a) it does no harm and (b) it allowed us to simply reuse
 * the prior info array
rc = PMIx_Get(NULL, PMIX_NODE_SIZE, info, 2, &value);
CHAPTER 7

Process-Related Non-Reserved Keys

Non-reserved keys are keys whose string representation begin with a prefix other than "pmix". Such keys are typically defined by an application when information needs to be exchanged between processes (e.g., where connection information is required and the host environment does not support the instant on option) or where the host environment does not provide a required piece of data. Beyond the restriction on name prefix, non-reserved keys are required to be unique across conflicting scopes as defined in Section 7.1.1.1 - e.g., a non-reserved key cannot be posted by the same process in both the PMIX_LOCAL and PMIX_REMOTE scopes (note that posting the key in the PMIX_GLOBAL scope would have met the desired objective).

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

- Global, collective exchange of the information prior to retrieval. This is accomplished by executing a barrier operation that includes collection and exchange of the data provided by each process such that each process has access to the full set of data from all participants once the operation has completed. PMIx provides the PMIx_Fence function (or its non-blocking equivalent) for this purpose, accompanied by the PMIX_COLLECT_DATA qualifier.

- Direct, on-demand retrieval of the information. No barrier or global exchange is conducted in this case. Instead, information is retrieved from the host where that process is executing upon request - i.e., a call to PMIx_Get results in a data exchange with the PMIx server on the remote host. Various caching strategies may be employed by the host environment and/or PMIx implementation to reduce the number of retrievals. Note that this method requires that the host environment both know the location of the posting process and support direct information retrieval.

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

Note that the retrieval rules for non-reserved keys (detailed in Section 7.2) differ significantly from those used for reserved keys.
7.1 Posting Key/Value Pairs

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

7.1.1 PMIx_Put

Summary
Post a key/value pair for distribution.

Format

```
PMIx v1.0

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

IN scope
Distribution scope of the provided value (handle)

IN key
key (pmix_key_t)

IN value
Reference to a pmix_value_t structure (handle)

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

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

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

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

Advice to users

Note that keys starting with a string of “`pmix`” must not be used in calls to `PMIx_Put`. Thus, applications should never use a defined “PMIX” attribute as the key in a call to `PMIx_Put`. 
7.1.1.1 Scope of Put Data

The `pmix_scope_t` structure is a `uint8_t` type that defines the availability of data passed to `PMIx_Put`. The following constants can be used to set a variable of the type `pmix_scope_t`.

All definitions were introduced in version 1 of the standard unless otherwise marked.

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

- `PMIX_SCOPE_UNDEF` Undefined scope.
- `PMIX_LOCAL` The data is intended only for other application processes on the same node. Data marked in this way will not be included in data packages sent to remote requesters - i.e., it is only available to processes on the local node.
- `PMIX_REMOTE` The data is intended solely for applications processes on remote nodes. Data marked in this way will not be shared with other processes on the same node - i.e., it is only available to processes on remote nodes.
- `PMIX_GLOBAL` The data is to be shared with all other requesting processes, regardless of location.

PMIx v2.0

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

7.1.2 PMIx_Store_internal

Summary

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

Format

PMIx v1.0

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

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

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

114 PMIx Standard – Version 4.0 – December 2020
Description
Store some data locally for retrieval by other areas of the process. This is data that has only internal
scope - it will never be posted externally. Typically used to cache data obtained by means outside of
PMIx so that it can be accessed by various areas of the process.

7.1.3 PMIx_Commit

Summary
Post all previously PMIx_Put values for distribution.

Format
PMIx v1.0

C

pmix_status_t PMIx_Commit(void);

C

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

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

Advice to users

Users are advised to always include the call to PMIx_Commit in case the local implementation
requires it. Note that posted data will not be circulated during PMIx_Commit. Availability of the
data by other processes upon completion of PMIx_Commit therefore still relies upon the exchange
mechanisms described at the beginning of this chapter.
7.2 Retrieval rules for non-reserved keys

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

1. If the `PMIX_GET_REFRESH_CACHE` attribute is given, then the request is first forwarded to the local PMIx server which will then update the client’s cache. Note that this may not, depending upon implementation details, result in any action.

2. Check the local PMIx client cache for the requested key - if not found and either the `PMIX_OPTIONAL` or `PMIX_GET_REFRESH_CACHE` attribute was given, the search will stop at this point and return the `PMIX_ERR_NOT_FOUND` status.

3. Request the information from the local PMIx server. The server will check its cache for the specified key within the appropriate scope as defined by the process that originally posted the key. If the value exists in a scope that contains the requesting process, then the value shall be returned. If the value exists, but in a scope that excludes the requesting process, then the server shall immediately return the `PMIX_ERR_EXISTS_OUTSIDE_SCOPE`.

If the value still isn’t found and the `PMIX_IMMEDIATE` attribute was given, then the library shall return the `PMIX_ERR_NOT_FOUND` error constant to the requester. Otherwise, the PMIx server library will take one of the following actions:

- If the target process has a rank of `PMIX_RANK_UNDEF`, then this indicates that the key being requested is globally unique and not associated with a specific process. In this case, the server shall hold the request until either the data appears at the server or, if given, the `PMIX_TIMEOUT` is reached. In the latter case, the server will return the `PMIX_ERR_TIMEOUT` status. Note that the server may, depending on PMIx implementation, never respond if the caller failed to specify a `PMIX_TIMEOUT` and the requested key fails to arrive at the server.

- If the target process is local (i.e., attached to the same PMIx server), then the server will hold the request until either the target process provides the data or, if given, the `PMIX_TIMEOUT` is reached. In the latter case, the server will return the `PMIX_ERR_TIMEOUT` status. Note that data which is posted via `PMIx_Put` but not staged with `PMIx_Commit` may, depending upon implementation, never appear at the server.

- If the target process is remote (i.e., not attached to the same PMIx server), the server will either:
  
  - If the host has provided the `pmix_server_dmodex_req_fn_t` module function interface, then the server shall pass the request to its host for servicing. The host is responsible for determining the location of the target process and passing the request to the PMIx server at that location.

  When the remote data request is received, the target PMIx server will check its cache for the specified key. If the key is not present, the request shall be held until either the target process provides the data or, if given, the `PMIX_TIMEOUT` is reached. In the latter case,
the server will return the `PMIX_ERR_TIMEOUT` status. The host shall convey the result
back to the originating PMIx server, which will reply to the requesting client with the result
of the request when the host provides it.

Note that the target server may, depending on PMIx implementation, never respond if the
caller failed to specify a `PMIX_TIMEOUT` and the target process fails to post the requested
key.

- if the host does not support the `pmix_server_dmodex_req_fn_t` interface, then the
  server will immediately respond to the client with the `PMIX_ERR_NOT_FOUND` status

--- Advice to PMIx library implementers

While there is no requirement that all PMIx implementations follow the client-server paradigm
used in the above description, implementers are required to provide behaviors consistent with the
described search pattern.

--- Advice to users

Users are advised to always specify the `PMIX_TIMEOUT` value when retrieving non-reserved keys
to avoid potential deadlocks should the specified key not become available.

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

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

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

### 8.1 PMIx_Publish

**Summary**

Publish data for later access via PMIx_Lookup.

**Format**

<table>
<thead>
<tr>
<th>PMIx v1.0</th>
<th>C</th>
</tr>
</thead>
</table>

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

**IN info**

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

**IN ninfo**

Number of elements in the info array (integer)

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
Required Attributes

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

Optional Attributes

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

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

- **PMIX_RANGE** "pmix.range" (pmix_data_range_t)
  
  Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

- **PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)
  
  Declare how long the datastore shall retain the provided data. The datastore is to delete the data upon reaching the persistence criterion.

- **PMIX_ACCESS_PERMISSIONS** "pmix.apperms" (pmix_data_array_t)
  
  Define access permissions for the published data. The value shall contain an array of `pmix_info_t` structs containing the specified permissions.

Description

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

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

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

**Summary**
Nonblocking `PMIx_Publish` routine.

**Format**

```c
PMIx v1.0

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

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

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

**IN cbfunc**
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will *not* be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will *not* be called.

**Required Attributes**

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

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

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

**PMIX_RANGE** "pmix.range" (pmix_data_range_t)
Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

**PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)
Declare how long the datastore shall retain the provided data. The datastore is to delete the data upon reaching the persistence criterion.

**PMIX_ACCESS_PERMISSIONS** "pmix.aperms" (pmix_data_array_t)
Define access permissions for the published data. The value shall contain an array of **pmix_info_t** structs containing the specified permissions.

Description
Nonblocking **PMIX_Publish** routine.

### 8.3 Publish-specific constants

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

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

### 8.4 Publish-specific attributes

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

**PMIX_RANGE** "pmix.range" (pmix_data_range_t)
Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

**PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)
Declare how long the datastore shall retain the provided data. The datastore is to delete the data upon reaching the persistence criterion.

**PMIX_ACCESS_PERMISSIONS** "pmix.aperms" (pmix_data_array_t)
Define access permissions for the published data. The value shall contain an array of **pmix_info_t** structs containing the specified permissions.
PMIX_ACCESS_USERIDS "pmix.aulids" (pmix_data_array_t)
Array of effective User IDs (UIDs) that are allowed to access the published data.

PMIX_ACCESS_GRPIDS "pmix.agids" (pmix_data_array_t)
Array of effective Group IDs (GIDs) that are allowed to access the published data.

8.5 Publish-Lookup Datatypes

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

8.5.1 Range of Published Data

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

PMIX_RANGE_UNDEF Undefined range.
PMIX_RANGE_RM Data is intended for the host environment, or lookup is restricted to data
published by the host environment.
PMIX_RANGE_LOCAL Data is only available to processes on the local node, or lookup is
restricted to data published by processes on the local node of the requester.
PMIX_RANGE_NAMESPACE Data is only available to processes in the same namespace, or
lookup is restricted to data published by processes in the same namespace as the requester.
PMIX_RANGE_SESSION Data is only available to all processes in the session, or lookup is
restricted to data published by other processes in the same session as the requester.
PMIX_RANGE_GLOBAL Data is available to all processes, or lookup is open to data published
by anyone.
PMIX_RANGE_CUSTOM Data is available only to processes as specified in the
pmix_info_t associated with this call, or lookup is restricted to data published by
processes as specified in the pmix_info_t.
PMIX_RANGE_PROC_LOCAL Data is only available to this process, or lookup is restricted to
data published by this process.
PMIX_RANGE_INVALID Invalid value - typically used to indicate that a range has not yet
been set.

8.5.2 Data Persistence Structure

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

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 - typically used to indicate that a persistence has
not yet been set.
8.6 PMIx_Lookup

Summary

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

Format

```
PMIx v1.0

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

<table>
<thead>
<tr>
<th>INOUT data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array of publishable data structures (array of pmix_pdata_t)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN ndata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of elements in the data array (integer)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array of info structures (array of pmix_info_t)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN ninfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of elements in the info array (integer)</td>
</tr>
</tbody>
</table>

Returns one of the following:

- **PMIX_SUCCESS** All data was found and has been returned.
- **PMIX_ERR_NOT_FOUND** None of the requested data could be found within the requester’s range.
- **PMIX_ERR_PARTIAL_SUCCESS** Some of the requested data was found. Any key that cannot be found will return with a data type of PMIX_UNDEF in the associated value struct. Note that the specific reason for a particular piece of missing information (e.g., lack of permissions) cannot be communicated back to the requester in this situation.
- **PMIX_ERR_NOT_SUPPORTED** There is no available datastore (either at the host environment or PMIx implementation level) on this system that supports this function.
- **PMIX_ERR_NO_PERMISSIONS** All of the requested data was found and range restrictions were met for each specified key, but none of the matching data could be returned due to lack of access permissions.
- a non-zero PMIx error constant indicating a reason for the request’s failure.
Required Attributes

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

Optional Attributes

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

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

- **PMIX_RANGE** "pmix.range" (pmix_data_range_t)
  
  Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

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

Description

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

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

Advice to users

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

Summary
Nonblocking version of PMIx_Lookup.

Format

PMIx v1.0

```c
#include <pmix/pmix_api.h>

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

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

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

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

**IN** cbfunc
Callback function (handle)

**IN** cbdata
Callback data to be provided to the callback function (pointer)

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.

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

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

- **PMIX_ERR_NOT_FOUND** None of the requested data was available within the requester’s range. The pdata array in the callback function shall be NULL and the npdata parameter set to zero.

- **PMIX_ERR_PARTIAL_SUCCESS** Some of the requested data was found. Only found data will be included in the returned pdata array. Note that the specific reason for a particular piece of missing information (e.g., lack of permissions) cannot be communicated back to the requester in this situation.

- **PMIX_ERR_NOT_SUPPORTED** There is no available datastore (either at the host environment or PMIx implementation level) on this system that supports this function.
• **PMIX_ERR_NO_PERMISSIONS** All of the requested data was found and range restrictions were met for each specified key, but none of the matching data could be returned due to lack of access permissions.

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

--- Required Attributes ---

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

--- Optional Attributes ---

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

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

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

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

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

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

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

--- Description ---

Non-blocking form of the `PMIx_Lookup` function.
8.7.1 Lookup Returned Data Structure

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

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

where:

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

8.7.1.1 Lookup data structure support macros

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

**Initialize the pdata structure**

Initialize the `pmix_pdata_t` fields

```c
PMIX_PDATA_CONSTRUCT(m)
```

**IN**

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

**Destruct the pdata structure**

Destruct the `pmix_pdata_t` fields

```c
PMIX_PDATA_DESTRUCT(m)
```

**IN**

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

CHAPTER 8. PUBLISH/LOOKUP OPERATIONS 127
Create a pdata array
Allocate and initialize an array of pmix_pdata_t structures

PMIx v1.0
PMIX_PDATA_CREATE(m, n)

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

IN n
Number of structures to be allocated (size_t)

Free a pdata structure
Release a pmix_pdata_t structure

PMIx v4.0
PMIX_PDATA_RELEASE(m)

IN m
Pointer to a pmix_pdata_t structure (handle)

Free a pdata array
Release an array of pmix_pdata_t structures

PMIx v1.0
PMIX_PDATA_FREE(m, n)

IN m
Pointer to the array of pmix_pdata_t structures (handle)

IN n
Number of structures in the array (size_t)

Load a lookup data structure
This macro simplifies the loading of key, process identifier, and data into a pmix_pdata_t by correctly assigning values to the structure’s fields.

PMIx v1.0
PMIX_PDATA_LOAD(m, p, k, d, t);

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

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

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

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

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

--- Advice to users ---

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

--- Transfer a lookup data structure ---

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

PMIX v2.0

PMIX_PDATA_XFER(d, s);

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

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

--- Advice to users ---

Key, process identifier, and data will all be copied into the destination `pmix_pdata_t` - thus, the  
source `pmix_pdata_t` may free’d without affecting the copied data once the macro has  
completed.
### 8.7.2 Lookup Callback Function

#### Summary

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

PMIx v1.0

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

| IN status |
| Status associated with the operation (handle) |
| IN data |
| Array of data returned (`pmix_pdata_t`) |
| IN ndata |
| Number of elements in the `data` array (`size_t`) |
| IN cbdata |
| Callback data passed to original API call (memory reference) |

#### Description

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

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

### 8.8 Retrieval rules for published data

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

1. If the requester specified the range, then the search shall be constrained to data where the publishing process falls within the specified range.
2. If the key of the stored information does not match the specified key, then the search will continue.
3. If the requester’s identifier does not fall within the range specified by the publisher, then the search will continue.
4. If the publisher specified access permissions, the effective UID and GID of the requester shall be checked against those permissions, with the datastore rejecting the match if the requester fails to meet the requirements.

5. If all of the above checks pass, then the value is added to the information that is to be returned. The status returned by the datastore shall be set to:

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

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

---

**Advice to users**

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

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

---

**8.9 PMIx_Unpublish**

**Summary**

Unpublish data posted by this process using the given keys.
PMIx Standard – Version 4.0 – December 2020

Format

PMIx v1.0

C

pmix_status_t

PMIx_Unpublish(char **keys,

const pmix_info_t info[], size_t ninfo);

C

IN  keys

NULL-terminated array of keys (array of strings)

IN  info

Array of info structures (array of handles)

IN  ninfo

Number of elements in the info array (integer)

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 environment for processing, and the PMIx library is
required to add the PMIX_USERID and the PMIX_GRPID attributes of the client process that is
requesting the operation.

Optional Attributes

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

PMIX_TIMEOUT  "pmix.timeout" (int)

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

PMIX_RANGE  "pmix.range" (pmix_data_range_t)

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

Description

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

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

Summary
Nonblocking version of PMIx_Unpublish.

Format

PMIx v1.0

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

| IN | keys               | NULL-terminated array of keys (array of strings) |
| IN | info               | Array of info structures (array of handles)      |
| IN | ninfo              | Number of elements in the info array (integer)   |
| IN | cbfunc             | Callback function pmix_op_cbfunc_t (function reference) |
| IN | cbdata             | Data to be passed to the callback function (memory reference) |

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called.

Required Attributes

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

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

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

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

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

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

---

**Description**

Non-blocking form of the **PMIx_Unpublish** function. The callback function will be executed once the server confirms removal of the specified data. The *info* array must be maintained until the callback is provided.
This chapter defines the PMIx event notification system. These interfaces are designed to support the reporting of events to/from clients and servers, and between library layers within a single process.

9.1 Notification and Management

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

- **Job-specific events** directly relate to a job executing within the session, such as a debugger attachment, process failure within a related job, or events generated by an application process. Events in this category are to be immediately delivered to the PMIx server library for relay to the related local processes.

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

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

Advice to PMIx library implementers

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

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.

9.1.1 Events versus status constants

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

9.1.2 PMIx_Register_event_handler

Summary

Register an event handler.
PMIx v2.0

```c
pmix_status_t PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
        pmix_info_t info[], size_t ninfo,
        pmix_notification_fn_t evhdlr,
        pmix_hdlr_reg_cbfunc_t cbfunc,
        void *cbdata);
```

IN codes
Array of status codes (array of `pmix_status_t`)

IN ncodes
Number of elements in the codes array (size_t)

IN info
Array of info structures (array of handles)

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

IN evhdlr
Event handler to be called `pmix_notification_fn_t` (function reference)

IN cbfunc
Callback function `pmix_hdlr_reg_cbfunc_t` (function reference)

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

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

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

- **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 result of the registration
  operation shall be returned in the provided callback function along with the assigned event
  handler identifier.

- **PMIX_ERR_EVENT_REGISTRATION** indicating that the registration has failed for an
  undetermined reason.

- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this
  case, the provided callback function will not be executed.
The callback function must not be executed prior to returning from the API, and no events corresponding to this registration may be delivered prior to the completion of the registration callback function (cbfunc).

Required Attributes

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

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

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

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

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

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

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

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

- **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (bool)
  - Prepend this handler to the precedence list within its category.

- **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (bool)
  - Append this handler to the precedence list within its category.

- **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrange" (pmix_data_array_t*)
  - Array of pmix_proc_t defining range of event notification.

- **PMIX_RANGE** "pmix.range" (pmix_data_range_t)
  - Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

- **PMIX_EVENT_RETURN_OBJECT** "pmix.evobject" (void *)
  - Object to be returned whenever the registered callback function cbfunc is invoked. The object will only be returned to the process that registered it.
Host environments that implement support for PMIx event notification are required to support the following attributes when registering handlers - these attributes are used to direct that the handler should be invoked only when the event affects the indicated process(es):

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

The single process that was affected.

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

Array of `pmix_proc_t` defining affected processes.

**Description**

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.

**9.1.3 Event registration constants**

**PMIX_ERR_EVENT_REGISTRATION** Error in event registration.
### 9.1.4 System events

- **PMIX_EVENT_SYS_BASE**  
  Mark the beginning of a dedicated range of constants for system event reporting.

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

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

- **PMIX_EVENT_SYS_OTHER**  
  Mark the end of a dedicated range of constants for system event reporting.

#### Detect system event constant

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

```c
#include <pmix/libpmix2.h>

PMIx v2.2

PMIX_SYSTEM_EVENT(a)

IN a

Error constant to be checked (pmix_status_t)

Returns true if the provided value falls within the dedicated range of events for system event reporting.
```

### 9.1.5 Event handler registration and notification attributes

Attributes to support event registration and notification.

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

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

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

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

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

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

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

- **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (bool)  
  Prepend this handler to the precedence list within its category.

- **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (bool)  
  Append this handler to the precedence list within its category.
Append this handler to the precedence list within its category.

PMIX_EVENT_CUSTOM_RANGE "pmix.evrange" (pmix_data_array_t*)
Array of pmix_proc_t defining range of event notification.

PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)
The single process that was affected.

PMIX_EVENT_AFFECTED_PROCS "pmix.evaffected" (pmix_data_array_t*)
Array of pmix_proc_t defining affected processes.

PMIX_EVENT_NON_DEFAULT "pmix.evnondef" (bool)
Event is not to be delivered to default event handlers.

PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
Object to be returned whenever the registered callback function cbfunc is invoked. The object will only be returned to the process that registered it.

PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)
Instruct the PMIx server not to cache the event.

PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)
PMIx server that sourced the event.

PMIX_EVENT_TEXT_MESSAGE "pmix.evtext" (char*)
Text message suitable for output by recipient - e.g., describing the cause of the event.

PMIX_EVENT_TIMESTAMP "pmix.evtstamp" (time_t)
System time when the associated event occurred.

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

PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)
The RM intends to terminate this session.

PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)
The RM intends to terminate this job.

PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)
The RM intends to terminate all processes on this node.

PMIX_EVENT_TERMINATE_PROC "pmix.evterm.proc" (bool)
The RM intends to terminate just this process.

PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int)
The time in seconds before the RM will execute the indicated operation.

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

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

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

Summary

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

Advice to users

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

PMIx v2.0

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

- **evhdlr_registration_id**
  Registration number of the handler being called (`size_t`)

- **status**
  Status associated with the operation (`pmix_status_t`)

- **source**
  Identifier of the process that generated the event (`pmix_proc_t`). If the source is the SMS, then the nspace will be empty and the rank will be PMIX_RANK_UNDEF

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

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

- **results**
  Aggregated results from prior event handlers servicing this event (`pmix_info_t`). This argument will be NULL if this is the first handler servicing the event, or if no prior handlers provided results.

- **nresults**
  Number of elements in the results array (`size_t`)

- **cbfunc**
  `pmix_event_notification_cbfunc_fn_t` callback function to be executed upon completion of the handler’s operation and prior to handler return (function reference).
IN `cbdata`
   Callback data to be passed to `cbfunc` (memory reference)

**Description**

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

---

**Advice to users**

Possible uses of the `info` array include:

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

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

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

---

**Advice to PMIx server hosts**

On the server side, the notification function is used to inform the PMIx server library’s host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the `pmix_server_notify_event_fn_t` function pointer, if provided.

---

### 9.1.7 PMIx_Deregister_event_handler

**Summary**

Deregister an event handler.
C

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

IN  evhdlr_ref
    Event handler ID returned by registration (size_t)

IN  cbfunc
    Callback function to be executed upon completion of operation pmix_op_cbfunc_t
    (function reference)

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

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

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

- **PMIX_SUCCESS**, indicating that the request is being processed - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

The returned status code will be one of the following:

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

**Description**

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

**9.1.8 PMIx_Notify_event**

**Summary**

Report an event for notification via any registered event handler.
Format

```
PMIx v2.0

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

- **IN status**
  Status code of the event (`pmix_status_t`)

- **IN source**
  Pointer to a `pmix_proc_t` identifying the original reporter of the event (handle)

- **IN range**
  Range across which this notification shall be delivered (`pmix_data_range_t`)

- **IN info**
  Array of `pmix_info_t` structures containing any further info provided by the originator of the event (array of handles)

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

- **IN cbfunc**
  Callback function to be executed upon completion of operation `pmix_op_cbfunc_t` (function reference)

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

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

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

- **PMIX_SUCCESS** The notification request is valid and is being processed. The callback function will be called when the process-local operation is complete and will provide the resulting status of that operation. Note that this does *not* reflect the success or failure of delivering the event to any recipients. The callback function must not be executed prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the `cbfunc` will *not* be called

- **PMIX_ERR_BAD_PARAM** The request contains at least one incorrect entry that prevents it from being processed. The callback function will *not* be called.
The PMIx implementation does not support event notification, or in the case of a PMIx server calling the API, the range extended beyond the local node and the host SMS environment does not support event notification. The callback function will not be called.

--- Required Attributes ---

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

- **PMIX_EVENT_NON_DEFAULT** "pmix.evnondef" (bool)
  - Event is not to be delivered to default event handlers.
- **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrange" (pmix_data_array_t*)
  - Array of pmix_proc_t defining range of event notification.
- **PMIX_EVENT_DO_NOT_CACHE** "pmix.evnocache" (bool)
  - Instruct the PMIx server not to cache the event.
- **PMIX_EVENT_PROXY** "pmix.evproxy" (pmix_proc_t*)
  - PMIx server that sourced the event.
- **PMIX_EVENT_TEXT_MESSAGE** "pmix.evtext" (char*)
  - Text message suitable for output by recipient - e.g., describing the cause of the event.

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

- **PMIX_EVENT_AFFECTED_PROC** "pmix.evproc" (pmix_proc_t)
  - The single process that was affected.
- **PMIX_EVENT_AFFECTED_PROCS** "pmix.evaffected" (pmix_data_array_t*)
  - Array of pmix_proc_t defining affected processes.

--- Optional Attributes ---

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

- **PMIX_EVENT_TERMINATE_SESSION** "pmix.evterm.sess" (bool)
  - The RM intends to terminate this session.
- **PMIX_EVENT_TERMINATE_JOB** "pmix.evterm.job" (bool)
  - The RM intends to terminate this job.
- **PMIX_EVENT_TERMINATE_NODE** "pmix.evterm.node" (bool)
  - The RM intends to terminate all processes on this node.
- **PMIX_EVENT_TERMINATE_PROC** "pmix.evterm.proc" (bool)
The RM intends to terminate just this process.

PMIX_EVENT_ACTION_TIMEOUT "pmix.evtimeout" (int)

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

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

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

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

Advice to users
The callback function will be called upon completion of the notify_event function’s actions. At that time, any messages required for executing the operation (e.g., to send the notification to the local PMIx server) will have been queued, but may not yet have been transmitted. The caller is required to maintain the input data until the callback function has been executed — the sole purpose of the callback function is to indicate when the input data is no longer required.
9.1.9 Notification Handler Completion Callback Function

Summary

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

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

- **IN status**
  Status returned by the event handler’s operation (`pmix_status_t`)

- **IN results**
  Results from this event handler’s operation on the event (`pmix_info_t`)

- **IN nresults**
  Number of elements in the results array (`size_t`)

- **IN cbfunc**
  `pmix_op_cbfunc_t` function to be executed when PMIx completes processing the callback (function reference)

- **IN thiscbdata**
  Callback data that was passed in to the handler (memory reference)

- **IN cbdata**
  Callback data to be returned when PMIx executes cbfunc (memory reference)

Description

Define a callback by which an event handler can notify the PMIx library that it has completed its response to the notification. The handler is required to execute this callback so the library can determine if additional handlers need to be called. The handler shall return `PMIX_EVENT_ACTION_COMPLETE` if no further action is required. The return status of each event handler and any returned `pmix_info_t` structures will be added to the `results` array of `pmix_info_t` passed to any subsequent event handlers to help guide their operation.

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

9.1.9.1 Completion Callback Function Status Codes

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

- **PMIX_EVENT_NO_ACTION_TAKEN**
  Event handler: No action taken.

- **PMIX_EVENT_PARTIAL_ACTION_TAKEN**
  Event handler: Partial action taken.

- **PMIX_EVENT_ACTION_DEFERRED**
  Event handler: Action deferred.

- **PMIX_EVENT_ACTION_COMPLETE**
  Event handler: Action complete.
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.

### 10.1 Data Buffer Type

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

```
typedef struct pmix_data_buffer {
    /** Start of my memory */
    char *base_ptr;
    /** Where the next data will be packed to
        (within the allocated memory starting
        at base_ptr) */
    char *pack_ptr;
    /** Where the next data will be unpacked
        from (within the allocated memory
        starting as base_ptr) */
    char *unpack_ptr;
    /** Number of bytes allocated (starting
        at base_ptr) */
    size_t bytes_allocated;
    /** Number of bytes used by the buffer
        (i.e., amount of data -- including
        overhead -- packed in the buffer) */
    size_t bytes_used;
} pmix_data_buffer_t;
```
10.2 Support Macros

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

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

```
PMIx v2.0
PMIX_DATA_BUFFER_CREATE(buffer);
```

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

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

```
PMIx v2.0
PMIX_DATA_BUFFER_RELEASE(buffer);
```

**IN buffer**
Pointer to the `pmix_data_buffer_t` to be released (handle)

PMIX_DATA_BUFFER_CONSTRUCT
Initialize a statically declared `pmix_data_buffer_t` object.

```
PMIx v2.0
PMIX_DATA_BUFFER_CONSTRUCT(buffer);
```

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

PMIX_DATA_BUFFER_DESTRUCT
Release the data contained in a `pmix_data_buffer_t` object.

```
PMIx v2.0
PMIX_DATA_BUFFER_DESTRUCT(buffer);
```

**IN buffer**
Pointer to the `pmix_data_buffer_t` whose data is to be released (handle)
**PMIX_DATABUFFER_LOAD**
Load a blob into a `pmix_data_buffer_t` object. 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.

```c
PMIX v2.0
PMIX_DATA_BUFFER_LOAD(buffer, data, size);
```

**IN** buffer
Pointer to a pre-allocated `pmix_data_buffer_t` (handle)

**IN** data
Pointer to a blob (`char*`)

**IN** size
Number of bytes in the blob `size_t`

**PMIX_DATABUFFER_UNLOAD**
Unload the data from a `pmix_data_buffer_t` object. 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.

```c
PMIX v2.0
PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);
```

**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`

10.3 General Routines

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

10.3.1 PMIx_Data_pack

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

```

pmix_status_t PMIx_Data_pack(const pmix_proc_t *target,
   pmix_data_buffer_t *buffer,
   void *src, int32_t num_vals,
   pmix_data_type_t type);
```

- **IN** target
  Pointer to a `pmix_proc_t` containing the nspace/rank of the process that will be unpacking the final buffer. A NULL value may be used to indicate that the target is based on the same PMIx version as the caller. Note that only the target’s nspace is relevant. (handle)

- **IN** buffer
  Pointer to a `pmix_data_buffer_t` where the packed data is to be stored (handle)

- **IN** src
  Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the caller must pass the address of the pointer to the string as the (void*). This allows the caller to pass multiple strings in a single call. (memory reference)

- **IN** num_vals
  Number of elements pointed to by the `src` pointer. A string value is counted as a single value regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g., string arrays) should be contiguous, although the data pointed to need not be contiguous across array entries. (`int32_t`)

- **IN** type
  The type of the data to be packed (`pmix_data_type_t`) Returns one of the following:

  - **PMIX_SUCCESS** The data has been packed as requested
  - **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.
  - **PMIX_ERR_BAD_PARAM** The provided buffer or src is NULL
  - **PMIX_ERR_UNKNOWN_DATA_TYPE** The specified data type is not known to this implementation
  - **PMIX_ERR_OUT_OF_RESOURCE** Not enough memory to support the operation
  - **PMIX_ERROR** General error

**Description**

The pack function packs one or more values of a specified type into the specified buffer. The buffer must have already been initialized via the `PMIX_DATA_BUFFER_CREATE` or `PMIX_DATA_BUFFER_CONSTRUCT` macros — otherwise, `PMIx_Data_pack` will return an error. Providing an unsupported type flag will likewise be reported as an error.

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

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

10.3.2 PMIx_Data_unpack

Summary
Unpack values from a pmix_data_buffer_t

Format

PMIx v2.0

```

C

pmix_status_t

PMIx_Data_unpack(const pmix_proc_t *source,
              pmix_data_buffer_t *buffer, void *dest,
              int32_t *max_num_values,
              pmix_data_type_t type);
```

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 that 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.

### 10.3.3 PMIx_Data_copy

**Summary**

Copy a data value from one location to another.

**Format**

```c
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.

### 10.3.4 PMIx_Data_print

**Summary**

Pretty-print a data value.
pmix_status_t

PMIx_Data_print(char **output, char *prefix,
    void *src, pmix_data_type_t type);

**IN** output
- The address of a pointer into which the address of the resulting output is to be stored.

**IN** prefix
- String to be prepended to the resulting output (char*)

**IN** src
- A pointer to the memory location of the data value to be printed (handle)

**IN** type
- The type of the data value to be printed — must be one of the PMIx defined data types.

(pmix_data_type_t)

Returns one of the following:

- PMIX_SUCCESS The data has been printed as requested
- PMIX_ERR_BAD_PARAM The provided data type is not recognized.
- PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.

**Description**
- Since registered data types can be complex structures, the system needs some way to know how to print them (i.e., convert them to a string representation). Primarily for debug purposes.

**10.3.5** PMIx_Data_copy_payload

**Summary**
- Copy a payload from one buffer to another

**Format**
PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
        pmix_data_buffer_t *src);

IN dest
    Pointer to the destination pmix_data_buffer_t (handle)

IN src
    Pointer to the source pmix_data_buffer_t (handle)

Returns one of the following:

PMIX_SUCCESS  The data has been copied as requested
PMIX_ERR_BAD_PARAM  The src and dest pmix_data_buffer_t types do not match
PMIX_ERR_NOT_SUPPORTED  The PMIx implementation does not support this function.

Description
This function will append a copy of the payload in one buffer into another buffer. Note that this is
not a destructive procedure — the source buffer’s payload will remain intact, as will any pre-existing
payload in the destination’s buffer. Only the unpacked portion of the source payload will be copied.
This chapter defines functionality processes can use to abort processes, spawn processes, and determine the relative locality of local processes.

11.1 Abort

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

11.1.1 PMIx_Abort

Summary
Abort the specified processes

Format

```
PMIx v1.0

pmix_status_t
PMIx_Abort(int status, const char msg[],
            pmix_proc_t procs[], size_t nprocs)
```

IN status
Error code to return to invoking environment (integer)

IN msg
String message to be returned to user (string)

IN procs
Array of pmix_proc_t structures (array of handles)

IN nprocs
Number of elements in the procs array (integer)

Returns one of the following:

- **PMIX_SUCCESS** if the operation was successfully completed. Note that the function shall not return in this situation if the caller’s own process was included in the request.

- **PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED** if the PMIx implementation and host environment support this API, but the request includes processes that the host environment cannot abort - e.g., if the request is to abort subsets of processes from a namespace, or processes outside of the caller’s own namespace, and the host environment does not permit such operations. In this case, none of the specified processes will be terminated.

- a PMIx error constant indicating an error in the request.
Description
Request that the host resource manager print the provided message and abort the provided array of
procs. A Unix or POSIX environment should handle the provided status as a return error code from
the main program that launched the application. A NULL for the procs array indicates that all
processes in the caller’s namespace are to be aborted, including itself - this is the equivalent of
passing a pmix_proc_t array element containing the caller’s namespace and a rank value of
PMIX_RANK_WILDCARD. While it is permitted for a caller to request abort of processes from
namespaces other than its own, not all environments will support such requests. Passing a NULL
msg parameter is allowed.

The function shall not return until the host environment has carried out the operation on the
specified processes. If the caller is included in the array of targets, then the function will not return
unless the host is unable to execute the operation.

Advice to users
The response to this request is somewhat dependent on the specific RM and its configuration (e.g.,
some resource managers will not abort the application if the provided status is zero unless
specifically configured to do so, some cannot abort subsets of processes in an application, and some
may not permit termination of processes outside of the caller’s own namespace), and thus lies
outside the control of 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.

11.2 Process Creation
The PMIx_Spawn commands spawn new processes and/or applications in the PMIx universe.
This may include requests to extend the existing resource allocation or obtain a new one, depending
upon provided and supported attributes.

11.2.1 PMIx_Spawn

Summary
Spawn a new job.
**PMIx v1.0**

```c
pmix_status_t
PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,
const pmix_app_t apps[], size_t napps,
char nspace[])
```

**IN job_info**
Array of info structures (array of handles)

**IN ninfo**
Number of elements in the `job_info` array (integer)

**IN apps**
Array of `pmix_app_t` structures (array of handles)

**IN napps**
Number of elements in the `apps` array (integer)

**OUT nspace**
Namespace of the new job (string)

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

---

**Required Attributes**

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

Host environments are required to support the following attributes when present in either the `job_info` or the `info` array of an element of the `apps` array:

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

- **PMIX_SET_SESSION_CWD** "pmix.ssncwd" (bool)
  Set the current working directory to the session working directory assigned by the RM - can be assigned to the entire job (by including attribute in the `job_info` array) or on a per-application basis in the `info` array for each `pmix_app_t`.

- **PMIX_PREFIX** "pmix.prefix" (char*)
  Prefix to use for starting spawned processes - i.e., the directory where the executables can be found.

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

- **PMIX_HOSTFILE** "pmix.hostfile" (char*)
  Hostfile to use for spawned processes.
The following attributes are optional for host environments that support this operation:

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

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

**PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)
- Preload executables onto nodes prior to executing launch procedure.

**PMIX_PRELOAD_FILES** "pmix.preloadfiles" (char*)
- Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

**PMIX_PERSONALITY** "pmix.pers" (char*)
- Name of personality corresponding to programming model used by application - supported values depend upon PMIx implementation.

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

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

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

**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. Supported values are launcher specific.

**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. Supported values are launcher specific.

**PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)
- Spawned process rank that is to receive any forwarded stdin.

**PMIX_TAG_OUTPUT** "pmix.tagout" (bool)
- Tag stdout/stderr with the identity of the source process - can be assigned to the entire job (by including attribute in the `job_info` array) or on a per-application basis in the `info` array for each `pmix_app_t`.

**PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
Timestamp output - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

**PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
Merge stdout and stderr streams - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

**PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)
Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

**PMIX_INDEX_ARGV** "pmix.indexargv" (bool)
Mark the argv with the rank of the process.

**PMIX_CPUS_PER_PROC** "pmix.cpuperproc" (uint32_t)
Number of PUs to assign to each rank - when accessed using PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the PUs/process assigned to the provided namespace.

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

**PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)
Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a node.

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

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

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

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

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

**PMIX_SET_ENVAR** "pmix.envvar.set" (pmix_envar_t*)
Set the envvar to the given value, overwriting any pre-existing one.
PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)
Unset the environment variable specified in the string.

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

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

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

PMIX_FIRST_ENVAR "pmix.envar.first" (pmix_envar_t*)
Ensure the given value appears first in the specified envar using the separator character, creating the envar if it doesn’t already exist.

PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
Name of the WLM queue to which the allocation request is to be directed, or the queue being referenced in a query.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
Total session time (in seconds) being requested in an allocation request.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
The number of nodes being requested in an allocation request.

PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
Regular expression of the specific nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
Number of PUs being requested in an allocation request.

PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
Regular expression of the number of PUs for each node being requested in an allocation request.

PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
Regular expression of the specific PUs being requested in an allocation request.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
Number of Megabytes[base2] of memory (per process) being requested in an allocation request.

PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
Fabric quality of service level for the job being requested in an allocation request.

```
PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
```
Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

```
PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
```
ID string for the fabric plane to be used for the requested allocation.

```
PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
```
Number of endpoints to allocate per process in the job.

```
PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
```
Number of endpoints to allocate per node for the job.

```
PMIX_COSPAWN_APP "pmix.cospawn" (bool)
```
Designated application is to be spawned as a disconnected job - i.e., the launcher shall not include the application in any of the job-level values (e.g., PMIX_RANK within the job) provided to any other application process generated by the same spawn request. Typically used to cospawn debugger daemons alongside an application.

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

```
PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
```
Do not generate an event when this job normally terminates.

---

**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 Section 11.3 for details). This includes that (a) the parent process will be given a copy of the new job’s information so it can query job-level info without incurring any communication penalties, (b) newly spawned child processes will receive a copy of the parent processes job-level info, and (c) both the parent process and members of the child job will receive notification of errors from processes in their combined assemblage.

---

**Advice to users**

Behavior of individual resource managers may differ, but it is expected that failure of any application process to start will result in termination/cleanup of all processes in the newly spawned job and return of an error code to the caller.
Tools may utilize `PMIx_Spawn` to start intermediate launchers as described in Section 17.2.2. For times where the tool is not attached to a PMIx server, internal support for fork/exec of the specified applications would allow the tool to maintain a single code path for both the connected and disconnected cases. Inclusion of such support is recommended, but not required.

### 11.2.2 PMIx_Spawn_nb

#### Summary

Nonblocking version of the `PMIx_Spawn` routine.

#### Format

```
PMIx v1.0

pmix_status_t
PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,
               const pmix_app_t apps[], size_t napps,
               pmix_spawn_cbfunc_t cbfunc, void *cbdata)
```

### IN job_info
- Array of info structures (array of handles)

### IN ninfo
- Number of elements in the `job_info` array (integer)

### IN apps
- Array of `pmix_app_t` structures (array of handles)

### IN cbfunc
- Callback function `pmix_spawn_cbfunc_t` (function reference)

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

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.

Host environments are required to support the following attributes when present in either the `job_info` or the `info` array of an element of the `apps` array:

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

- **PMIX_SET_SESSION_CWD** "pmix.ssncwd" (bool)
  Set the current working directory to the session working directory assigned by the RM - can be assigned to the entire job (by including attribute in the `job_info` array) or on a per-application basis in the `info` array for each `pmix_app_t`.

- **PMIX_PREFIX** "pmix.prefix" (char*)
  Prefix to use for starting spawned processes - i.e., the directory where the executables can be found.

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

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

Optional Attributes

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

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

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

- **PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)
  Preload executables onto nodes prior to executing launch procedure.

- **PMIX_PRELOAD_FILES** "pmix.preloadfiles" (char*)
  Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

- **PMIX_PERSONALITY** "pmix.pers" (char*)
  Name of personality corresponding to programming model used by application - supported values depend upon PMIx implementation.

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

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

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

**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. Supported values are launcher specific.

**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. Supported values are launcher specific.

**PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)
Spawned process rank that is to receive any forwarded stdin.

**PMIX_TAG_OUTPUT** "pmix.tagout" (bool)
Tag stdout/stderr with the identity of the source process - can be assigned to the entire
job (by including attribute in the job_info array) or on a per-application basis in the info
array for each pmix_app_t.

**PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
Timestamp output - can be assigned to the entire job (by including attribute in the job_info
array) or on a per-application basis in the info array for each pmix_app_t.

**PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
Merge stdout and stderr streams - can be assigned to the entire job (by including
attribute in the job_info array) or on a per-application basis in the info array for each
pmix_app_t.

**PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)
Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be
assigned to the entire job (by including attribute in the job_info array) or on a per-application
basis in the info array for each pmix_app_t.

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

**PMIX_CPUS_PER_PROC** "pmix.cpuperproc" (uint32_t)
Number of PUs to assign to each rank - when accessed using PMIx_Get, use the
PMIX_RANK_WILDCARD value for the rank to discover the PUs/process assigned to the
provided namespace.

**PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)
Do not place processes on the head node.
PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
node.

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

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

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

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

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

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

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

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

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

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

PMIX_FIRST_ENVAR "pmix.envar.first" (pmix_envvar_t*)
Ensure the given value appears first in the specified envvar using the separator character,
creating the envvar if it doesn’t already exist

PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
Name of the WLM queue to which the allocation request is to be directed, or the queue being
referenced in a query.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
Total session time (in seconds) being requested in an allocation request.
PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
The number of nodes being requested in an allocation request.

PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
Regular expression of the specific nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
Number of PUs being requested in an allocation request.

PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
Regular expression of the number of PUs for each node being requested in an allocation request.

PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
Regular expression of the specific PUs being requested in an allocation request.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
Number of Megabytes [base2] of memory (per process) being requested in an allocation request.

PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
Fabric bandwidth (in Megabits [base2]/sec) for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
Fabric quality of service level for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
ID string for the fabric plane to be used for the requested allocation.

PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
Number of endpoints to allocate per process in the job.

PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
Number of endpoints to allocate per node for the job.

PMIX_COSPAWN_APP "pmix.cospawn" (bool)
Designated application is to be spawned as a disconnected job - i.e., the launcher shall not include the application in any of the job-level values (e.g., PMIX_RANK within the job) provided to any other application process generated by the same spawn request. Typically used to cospawn debugger daemons alongside an application.

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

PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
Do not generate an event when this job normally terminates.
Description

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

Advice to users

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

11.2.3 Spawn-specific constants

In addition to the generic error constants, the following spawn-specific error constants may be returned by the spawn APIs:

- PMIX_ERR_JOB_ALLOC_FAILED: The job request could not be executed due to failure to obtain the specified allocation.
- PMIX_ERR_JOB_APP_NOT_EXECUTABLE: The specified application executable either could not be found, or lacks execution privileges.
- PMIX_ERR_JOB_NO_EXE_SPECIFIED: The job request did not specify an executable.
- PMIX_ERR_JOB_FAILED_TO_MAP: The launcher was unable to map the processes for the specified job request.
- PMIX_ERR_JOB_FAILED_TO_LAUNCH: One or more processes in the job request failed to launch.

11.2.4 Spawn attributes

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

- PMIX_PERSONALITY "pmix.pers" (char*): Name of personality corresponding to programming model used by application - supported values depend upon PMIx implementation.
- PMIX_HOST "pmix.host" (char*): Comma-delimited list of hosts to use for spawned processes.
- PMIX_HOSTFILE "pmix.hostfile" (char*): Hostfile to use for spawned processes.
- PMIX_ADD_HOST "pmix.addhost" (char*)
Comma-delimited list of hosts to add to the allocation.

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

**PMIX_PREFIX** "pmix.prefix" (char*)
Prefix to use for starting spawned processes - i.e., the directory where the executables can be found.

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

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

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

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

**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. Supported values are launcher specific.

**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. Supported values are launcher specific.

**PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)
Preload executables onto nodes prior to executing launch procedure.

**PMIX_PRELOAD FILES** "pmix.preloadfiles" (char*)
Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

**PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)
Spawned process rank that is to receive any forwarded stdin.

**PMIX_SET_SESSION_CWD** "pmix.ssncwd" (bool)
Set the current working directory to the session working directory assigned by the RM - can be assigned to the entire job (by including attribute in the **job_info** array) or on a per-application basis in the **info** array for each **pmix_app_t**.

**PMIX_TAG_OUTPUT** "pmix.tagout" (bool)
Tag stdout/stderr with the identity of the source process - can be assigned to the entire job (by including attribute in the **job_info** array) or on a per-application basis in the **info** array for each **pmix_app_t**.

**PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
Timestamp output - can be assigned to the entire job (by including attribute in the **job_info** array) or on a per-application basis in the **info** array for each **pmix_app_t**.

**PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
Merge `stdout` and `stderr` streams - can be assigned to the entire job (by including attribute in the `job_info` array) or on a per-application basis in the `info` array for each `pmix_app_t`.

`PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)`
Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be assigned to the entire job (by including attribute in the `job_info` array) or on a per-application basis in the `info` array for each `pmix_app_t`.

`PMIX_OUTPUT_TODIRECTORY "pmix.outdir" (char*)`
Direct output into files of form "<directory>/<jobid>/rank.<rank> stdout[err]" - can be assigned to the entire job (by including attribute in the `job_info` array) or on a per-application basis in the `info` array for each `pmix_app_t`.

`PMIX_INDEXARGV "pmix.indexargv" (bool)`
Mark the `argv` with the rank of the process.

`PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)`
Number of PUs to assign to each rank - when accessed using `PMIx_Get`, use the `PMIX_RANK_WILDCARD` value for the rank to discover the PUs/process assigned to the provided namespace.

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

`PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)`
Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a node.

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

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

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

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

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

`PMIX_SPawning_TOOL "pmix.spawn.tool" (bool)`
Indicate that the job being spawned is a tool.

`PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)`
Include process stacktraces in timeout report from a job.

`PMIX_TIMEOUT_REPORT_STATE "pmix.tim.state" (bool)`
Report process states in timeout report from a job.

`PMIX_NOTIFY_JOB_EVENTS "pmix.note.jev" (bool)`
Requests that the launcher generate the `PMIX_EVENT_JOB_START`, `PMIX_LAUNCH_COMPLETE`, and `PMIX_EVENT_JOB_END` events. Each event is to include at least the namespace of the corresponding job and a `PMIX_EVENT_TIMESTAMP` indicating the time the event occurred. Note that the requester must register for these individual events, or capture and process them by registering a default event handler instead of individual handlers and then process the events based on the returned status code. Another common method is to register one event handler for all job-related events, with a separate handler for non-job events - see `PMix_Register_event_handler` for details.

`PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)`
Requests that the launcher generate the `PMIX_EVENT_JOB_END` event for normal or abnormal termination of the spawned job. The event shall include the returned status code (`PMIX_JOB_TERM_STATUS`) for the corresponding job; the identity (`PMIX_PROCID`) and exit status (`PMIX_EXIT_CODE`) of the first failed process, if applicable; and a `PMIX_EVENT_TIMESTAMP` indicating the time the termination occurred. Note that the requester must register for the event or capture and process it within a default event handler.

`PMIX_NOTIFY_PROCP_TERMINATION "pmix.noteproc" (bool)`
Requests that the launcher generate the `PMIX_EVENT_PROC_TERMINATED` event whenever a process either normally or abnormally terminates.

`PMIX_NOTIFY_PROCP_ABNORMAL_TERMINATION "pmix.noteabproc" (bool)`
Requests that the launcher generate the `PMIX_EVENT_PROC_TERMINATED` event only when a process abnormally terminates.

`PMIX_LOG_PROCP_TERMINATION "pmix.logproc" (bool)`
Requests that the launcher log the `PMIX_EVENT_PROC_TERMINATED` event whenever a process either normally or abnormally terminates.

`PMIX_LOG_PROCP_ABNORMAL_TERMINATION "pmix.logabproc" (bool)`
Requests that the launcher log the `PMIX_EVENT_PROC_TERMINATED` event only when a process abnormally terminates.

`PMIX_LOG_JOB_EVENTS "pmix.log.jev" (bool)`
Requests that the launcher log the `PMIX_EVENT_JOB_START`, `PMIX_LAUNCH_COMPLETE`, and `PMIX_EVENT_JOB_END` events using `PMIx_Log`, subject to the logging attributes of Section 12.4.3.

`PMIX_LOG_COMPLETION "pmix.logcomp" (bool)`
Requests that the launcher log the `PMIX_EVENT_JOB_END` event for normal or abnormal termination of the spawned job using `PMIx_Log`, subject to the logging attributes of Section 12.4.3. The event shall include the returned status code (`PMIX_JOB_TERM_STATUS`) for the corresponding job; the identity (`PMIX_PROCID`) and exit status (`PMIX_EXIT_CODE`) of the first failed process, if applicable; and a `PMIX_EVENT_TIMESTAMP` indicating the time the termination occurred.

`PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)`
Do not generate an event when this job normally terminates.

Attributes used to adjust remote environment variables prior to spawning the specified application processes.
PMIX_SET_ENVAR "pmix.envvar.set" (pmix_envvar_t*)
Set the envar to the given value, overwriting any pre-existing one

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

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

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

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

PMIX_FIRST_ENVAR "pmix.envvar.first" (pmix_envvar_t*)
Ensure the given value appears first in the specified envar using the separator character, creating the envar if it doesn’t already exist

11.2.5 Application Structure

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

```c
PMIx v1.0

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

11.2.5.1 App structure support macros

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

Initialize the `pmix_app_t` fields

```c
PMIx v1.0
PMIX_APP_CONSTRUCT(m)
```

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

**Destruct the app structure**

Destruct the `pmix_app_t` fields

```c
PMIx v1.0
PMIX_APP_DESTRUCT(m)
```

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

**Create an app array**

Allocate and initialize an array of `pmix_app_t` structures

```c
PMIx v1.0
PMIX_APP_CREATE(m, n)
```

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

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

**Free an app structure**

Release a `pmix_app_t` structure

```c
PMIx v4.0
PMIX_APP_RELEASE(m)
```

IN  m
Pointer to a `pmix_app_t` structure (handle)
**Free an app array**

Release an array of `pmix_app_t` structures

**PMIx v1.0**

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

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

**Create the info array of application directives**

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

**PMIx v2.2**

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

- **IN m**
  - Pointer to the `pmix_app_t` structure (handle)
- **IN n**
  - Number of directives to be allocated (`size_t`)

### 11.2.5.2 Spawn Callback Function

**Summary**

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

**PMIx v1.0**

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

- **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 \texttt{PMIx\_Spawn\_nb}, or upon failure to launch any of them.

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

11.3 Connecting and Disconnecting Processes

This section defines functions to connect and disconnect processes in two or more separate PMIx namespaces. The PMIx definition of \textit{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.

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 \texttt{PMIx\_Query\_info\_nb}). The host environment is required to generate a \texttt{PMIX\_ERR\_PROC\_TERM\_WO\_SYNC} event should any process in the assemblage terminate or call \texttt{PMIx\_Finalize} without first \textit{disconnecting} from the assemblage. If the job including the process is terminated as a result of that action, then the host environment is required to also generate the \texttt{PMIX\_ERR\_JOB\_TERM\_WO\_SYNC} for all jobs that were terminated as a result.

\begin{center}\textbf{Advice to PMIx server hosts}\end{center}

The \textit{connect} operation does not require the exchange of job-level information nor the inclusion of information posted by participating processes via \texttt{PMIx\_Put}. Indeed, the callback function utilized in \texttt{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.

11.3.1 PMIx_Connect

Summary
Connect namespaces.

Format

PMIx v1.0

PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,
              const pmix_info_t info[], size_t ninfo)

IN procs
Array of proc structures (array of handles)

IN nprocs
Number of elements in the procs array (integer)

IN info
Array of info structures (array of handles)

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

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

Required Attributes

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

The following attributes are optional for PMIx implementations:

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

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

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

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

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

As in the case of the `PMIx_Fence` operation, the `info` array can be used to pass user-level directives regarding timeout constraints and other options available from the host RM.

Advice to users
All processes engaged in a given `PMIx_Connect` operation must provide the identical `procs` array as ordering of entries in the array and the method by which those processes are identified (e.g., use of `PMIX_RANK_WILDCARD` versus listing the individual processes) may impact the host environment’s algorithm for uniquely identifying an operation.

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

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

**Summary**
Nonblocking `PMIx_Connect_nb` routine.

**Format**

```c
PMIx v1.0

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

**IN**
- `procs`
  Array of proc structures (array of handles)

**IN**
- `nprocs`
  Number of elements in the `procs` array (integer)

**IN**
- `info`
  Array of info structures (array of handles)

**IN**
- `ninfo`
  Number of elements in the `info` array (integer)

**IN**
- `cbfunc`
  Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

**Required Attributes**

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

The following attributes are optional for PMIx implementations:

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

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

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

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

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

Description
Nonblocking version of **PMIx_Connect**. The callback function is called once all processes identified in `procs` have called either **PMIx_Connect** or its non-blocking version, and the host environment has completed any supporting operations required to meet the terms of the PMIx definition of connected processes. See the advice provided in the description for **PMIx_Connect** for more information.

11.3.3 **PMIx_Disconnect**

Summary
Disconnect a previously connected set of processes.

Format

```
PMIx v1.0
```

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

`IN` **procs**
Array of proc structures (array of handles)

`IN` **nprocs**
Number of elements in the `procs` array (integer)

`IN` **info**
Array of info structures (array of handles)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request was successfully executed
• the **PMIX_ERR_INVALID_OPERATION** error indicating that the specified set of `procs` was not previously *connected* via a call to **PMIx_Connect** or its non-blocking form.

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

---------- Required Attributes  ----------

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

---------- Optional Attributes  ----------

The following attributes are optional for PMIx implementations:

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

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

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

**Description**

Disconnect a previously connected set of processes. The function will return once all processes identified in `procs` have called either **PMix_Disconnect** or its non-blocking version, *and* the host environment has completed any required supporting operations.

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

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

**Advice to users**

All processes engaged in a given **PMix_Disconnect** operation must provide the identical `procs` array as ordering of entries in the array and the method by which those processes are identified (e.g., use of **PMIX_RANK_WILDCARD** versus listing the individual processes) may impact the host environment’s algorithm for uniquely identifying an operation.
Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

11.3.4 `PMIx_Disconnect_nb`

Summary

Nonblocking `PMIx_Disconnect` routine.

Format

```
PMIx v1.0

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

- **IN procs**
  - Array of proc structures (array of handles)
- **IN nprocs**
  - Number of elements in the procs array (integer)
- **IN info**
  - Array of info structures (array of handles)
- **IN ninfo**
  - Number of elements in the info array (integer)
- **IN cbfunc**
  - Callback function `pmix_op_cbfunc_t` (function reference)
- **IN cbdata**
  - Data to be passed to the callback function (memory reference)

Returns one of the following:
• **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

• **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

--- Required Attributes ---

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

--- Optional Attributes ---

The following attributes are optional for PMIx implementations:

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

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

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

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

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

---

**Description**

Nonblocking **PMIX_Disconnect** routine. The callback function is called either:

• to return the `PMIX_ERR_INVALID_OPERATION` error indicating that the specified set of `procs` was not previously `connected` via a call to **PMIx_Connect** or its non-blocking form;

• to return a PMIx error constant indicating that the operation failed; or

• 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.
11.4 Process Locality

The relative locality of processes is often used to optimize their interactions with the hardware and other processes. PMIx provides a means by which the host environment can communicate the locality of a given process using the \texttt{PMIx\_server\_generate\_locality\_string} to generate an abstracted representation of that value. This provides a human-readable format and allows the client to parse the locality string with a method of its choice that may differ from the one used by the server that generated it.

There are times, however, when relative locality and other PMIx-provided information doesn't include some element required by the application. In these instances, the application may need access to the full description of the local hardware topology. PMIx does not itself generate such descriptions - there are multiple third-party libraries that fulfill that role. Instead, PMIx offers an abstraction method by which users can obtain a pointer to the description. This transparently enables support for different methods of sharing the topology between the host environment (which may well have already generated it prior to local start of application processes) and the clients - e.g., through passing of a shared memory region.

11.4.1 PMIx\_Load\_topology

Summary
Load the local hardware topology description

Format

\texttt{PMIx v4.0}

\begin{verbatim}

\texttt{pmix\_status\_t}\n
\texttt{PMIx\_Load\_topology(pmix\_topology\_t *topo)};

\end{verbatim}

\texttt{INOUT topo}
Address of a \texttt{pmix\_topology\_t} structure where the topology information is to be loaded (handle)

Returns \texttt{PMIX\_SUCCESS}, indicating that the \texttt{topo} was successfully loaded, or an appropriate PMIx error constant.

Description
Obtain a pointer to the topology description of the local node. If the \texttt{source} field of the provided \texttt{pmix\_topology\_t} is set, then the PMIx library must return a description from the specified implementation or else indicate that the implementation is not available by returning the \texttt{PMIX\_ERR\_NOT\_SUPPORTED} error constant.

The returned pointer may point to a shared memory region or an actual instance of the topology description. In either case, the description shall be treated as a "read-only" object - attempts to modify the object are likely to fail and return an error. The PMIx library is responsible for performing any required cleanup when the client library finalizes.
Advice to users

It is the responsibility of the user to ensure that the `topo` argument is properly initialized prior to calling this API, and to check the returned `source` to verify that the returned topology description is compatible with the user’s code.

11.4.2 PMIx_Get_relative_locality

Summary

Get the relative locality of two local processes given their locality strings.

Format

```
PMIx v4.0

pmix_status_t
PMIx_Get_relative_locality(const char *locality1,
                           const char *locality2,
                           pmix_locality_t *locality);
```

| IN   | locality1          |
|      | String returned by the PMIx_server_generate_locality_string API (handle) |

| IN   | locality2          |
|      | String returned by the PMIx_server_generate_locality_string API (handle) |

| INOUT | locality           |
|       | Location where the relative locality bitmask is to be constructed (memory reference) |

Returns `PMIX_SUCCESS`, indicating that the `locality` was successfully loaded, or an appropriate PMIx error constant.

Description

Parse the locality strings of two processes (as returned by PMIx_Get using the PMIX_LOCALITY_STRING key) and set the appropriate pmix_locality_t locality bits in the provided memory location.

11.4.2.1 Topology description

The `pmix_topology_t` structure contains a (case-insensitive) string identifying the source of the topology (e.g., "hwloc") and a pointer to the corresponding implementation-specific topology description.

```
PMIx v4.0

typedef struct pmix_topology {
    char *source;
    void *topology;
} pmix_topology_t;
```
11.4.2.2 Topology support macros

The following macros support the `pmix_topology_t` structure.

**Initialize the topology structure**

Initialize the `pmix_topology_t` fields to NULL

```
Pmix v4.0
PMIX_TOPOLOGY_CONSTRUCT (m)
```

**Destruct the topology structure**

Destruct the `pmix_topology_t` fields

```
Pmix v4.0
PMIX_TOPOLOGY_DESTRUCT (m)
```

**Create a topology array**

Allocate and initialize a `pmix_topology_t` array.

```
Pmix v4.0
PMIX_TOPOLOGY_CREATE (m, n)
```

**Release a topology array**

Release a `pmix_topology_t` array.

```
Pmix v4.0
PMIX_TOPOLOGY_FREE (m, n)
```
11.4.2.3 Relative locality of two processes

The `pmix_locality_t` datatype is a `uint16_t` bitmask that defines the relative locality of two processes on a node. The following constants represent specific bits in the mask and can be used to test a locality value using standard bit-test methods.

- **PMIX_LOCALITY_UNKNOWN**: All bits are set to zero, indicating that the relative locality of the two processes is unknown.
- **PMIX_LOCALITY_NONLOCAL**: The two processes do not share any common locations.
- **PMIX_LOCALITY_SHARE_HWTTHREAD**: The two processes share at least one hardware thread.
- **PMIX_LOCALITY_SHARE_CORE**: The two processes share at least one core.
- **PMIX_LOCALITY_SHARE_L1CACHE**: The two processes share at least an L1 cache.
- **PMIX_LOCALITY_SHARE_L2CACHE**: The two processes share at least an L2 cache.
- **PMIX_LOCALITY_SHARE_L3CACHE**: The two processes share at least an L3 cache.
- **PMIX_LOCALITY_SHARE_PACKAGE**: The two processes share at least a package.
- **PMIX_LOCALITY_SHARE_NUMA**: The two processes share at least one Non-Uniform Memory Access (NUMA) region.
- **PMIX_LOCALITY_SHARE_NODE**: The two processes are executing on the same node.

Implementers and vendors may choose to extend these definitions as needed to describe a particular system.

11.4.2.4 Locality keys

PMIX_LOCALITY_STRING "pmix.locstr" (char*)

String describing a process’s bound location - referenced using the process’s rank. The string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon. The remainder of the string represents the corresponding locality as expressed by the underlying implementation. The entire string must be passed to `PMIx_Get_relative_locality` for processing. Note that hosts are only required to provide locality strings for local client processes - thus, a call to `PMIx_Get` for the locality string of a process that returns `PMIX_ERR_NOT_FOUND` indicates that the process is not executing on the same node.

11.4.3 PMIx_Parse_cpuset_string

Summary

Parse the PU binding bitmap from its string representation.

Format

PMIx v4.0

```c
pmix_status_t PMIx_Parse_cpuset_string(const char *cpuset_string,
                                       pmix_cpuset_t *cpuset);
```
IN  cpuset_string
String returned by the PMIx_server_generate_cpuset_string API (handle)

INOUT cpuset
Address of an object where the bitmap is to be stored (memory reference)

Returns PMIX_SUCCESS, indicating that the cpuset was successfully loaded, or an appropriate PMIx error constant.

Description
Parse the string representation of the binding bitmap (as returned by PMIx_Get using the PMIX_CPUSET key) and set the appropriate PU binding location information in the provided memory location.

11.4.4 PMIx_Get_cpuset

Summary
Get the PU binding bitmap of the current process.

Format
PMIx v4.0

pmix_status_t
PMIx_Get_cpuset(pmix_cpuset_t *cpuset, pmix_bind_envelope_t ref);

INOUT cpuset
Address of an object where the bitmap is to be stored (memory reference)

IN  ref
The binding envelope to be considered when formulating the bitmap

(pmix_bind_envelope_t)

Returns PMIX_SUCCESS, indicating that the cpuset was successfully loaded, or an appropriate PMIx error constant.

Description
Obtain and set the appropriate PU binding location information in the provided memory location based on the specified binding envelope.

11.4.4.1 Binding envelope
PMIx v4.0
The pmix_bind_envelope_t data type defines the envelope of threads within a possibly multi-threaded process that are to be considered when getting the cpuset associated with the process. Valid values include:

PMIX_CPUBIND_PROCESS Use the location of all threads in the possibly multi-threaded process.

PMIX_CPUBIND_THREAD Use only the location of the thread calling the API.
11.4.5 **PMIx_Compute_distances**

**Summary**
Compute distances from specified process location to local devices.

**Format**

```
#include <pmix/pmix.h>

pmix_status_t PMIx_Compute_distances(pmix_topology_t *topo,
                                    pmix_cpuset_t *cpuset,
                                    pmix_info_t info[], size_t ninfo[],
                                    pmix_device_distance_t *distances[],
                                    size_t *ndist);
```

- **IN topo**
  Pointer to the topology description of the node where the process is located (**NULL** indicates the local node) (**pmix_topology_t**)

- **IN cpuset**
  Pointer to the location of the process (**pmix_cpuset_t**)

- **IN info**
  Array of **pmix_info_t** describing the devices whose distance is to be computed (handle)

- **IN ninfo**
  Number of elements in **info** (integer)

- **INOUT distances**
  Pointer to an address where the array of **pmix_device_distance_t** structures containing the distances from the caller to the specified devices is to be returned (handle)

- **INOUT ndist**
  Pointer to an address where the number of elements in the **distances** array is to be returned (handle)

Returns one of the following:

- **PMIX_SUCCESS** indicating that the distances were returned.

- a non-zero PMIx error constant indicating the reason the request failed.

**Description**
Both the minimum and maximum distance fields in the elements of the array shall be filled with the respective distances between the current process location and the types of devices or specific device identified in the **info** directives. In the absence of directives, distances to all supported device types shall be returned.
Advice to users

A process whose threads are not all bound to the same location may return inconsistent results from calls to this API by different threads if the `PMIX_CPUBIND_THREAD` binding envelope was used when generating the `cpuset`.

### 11.4.6 PMIx_Compute_distances_nb

#### Summary
Compute distances from specified process location to local devices.

#### Format

```c
pmix_status_t
PMIx_Compute_distances_nb(pmix_topology_t *topo,
                          pmix_cpuset_t *cpuset,
                          pmix_info_t info[], size_t ninfo[],
                          pmix_device_dist_cbfunc_t cbfunc,
                          void *cbdata);
```

- **IN topo**
  Pointer to the topology description of the node where the process is located (`NULL` indicates the local node) (`pmix_topology_t`)

- **IN cpuset**
  Pointer to the location of the process (`pmix_cpuset_t`)

- **IN info**
  Array of `pmix_info_t` describing the devices whose distance is to be computed (handle)

- **IN ninfo**
  Number of elements in `info` (integer)

- **IN cbfunc**
  Callback function `pmix_info_cbfunc_t` (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.

- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed.
Non-blocking form of the `PMIx_Compute_distances` API.

### 11.4.7 Device Distance Callback Function

**Summary**
The `pmix_device_dist_cbfunc_t` is used to return an array of device distances.

```c
typedef void (*pmix_device_dist_cbfunc_t)(
    pmix_status_t status,
    pmix_device_distance_t *dist,
    size_t ndist,
    void *cbdata,
    pmix_release_cbfunc_t release_fn,
    void *release_cbdata);
```

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

- **IN dist**
  Array of `pmix_device_distance_t` returned by the operation (pointer)

- **IN ndist**
  Number of elements in the `dist` array (`size_t`)

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

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

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

**Description**
The `status` indicates if requested data was found or not. The array of `pmix_device_distance_t` will contain the distance information.

### 11.4.8 Device type

The `pmix_device_type_t` is a `uint64_t` bitmask for identifying the type(s) whose distances are being requested, or the type of a specific device being referenced (e.g., in a `pmix_device_distance_t` object).

```c
typedef uint16_t pmix_device_type_t;
```
The following constants can be used to set a variable of the type `pmix_device_type_t`.

- **PMIX_DEVTYPE_UNKNOWN**: The device is of an unknown type - will not be included in returned device distances.
- **PMIX_DEVTYPE_BLOCK**: Operating system block device, or non-volatile memory device (e.g., "sda" or "dax2.0" on Linux).
- **PMIX_DEVTYPE_GPU**: Operating system Graphics Processing Unit (GPU) device (e.g., "card0" for a Linux Direct Rendering Manager (DRM) device).
- **PMIX_DEVTYPE_NETWORK**: Operating system network device (e.g., the "eth0" interface on Linux).
- **PMIX_DEVTYPE_OPENFABRICS**: Operating system OpenFabrics device (e.g., an "mlx4_0" InfiniBand Host Channel Adapter (HCA), or "hfi1_0" Omni-Path interface on Linux).
- **PMIX_DEVTYPE_DMA**: Operating system Direct Memory Access (DMA) engine device (e.g., the "dma0chan0" DMA channel on Linux).
- **PMIX_DEVTYPE_COPROC**: Operating system co-processor device (e.g., "mic0" for a Xeon Phi on Linux, "openc0ld0" for an OpenCL device, or "cuda0" for a Compute Unified Device Architecture (CUDA) device).

### 11.4.9 Device Distance Structure

The `pmix_device_distance_t` structure contains the minimum and maximum relative distance from the caller to a given device.

```c
typedef struct pmix_device_distance {
    char *uuid;
    char *osname;
    pmix_device_type_t type;
    uint16_t mindist;
    uint16_t maxdist;
} pmix_device_distance_t;
```

The `uuid` is a string identifier guaranteed to be unique within the cluster and is typically assembled from discovered device attributes (e.g., the Internet Protocol (IP) address of the device). The `osname` is the local operating system name of the device and is only unique to that node.

The two distance fields provide the minimum and maximum relative distance to the device from the specified location of the process, expressed as a 16-bit integer value where a smaller number indicates that this device is closer to the process than a device with a larger distance value. Note that relative distance values are not necessarily correlated to a physical property - e.g., a device at twice the distance from another device does not necessarily have twice the latency for communication with it.
Relative distances only apply to similar devices and cannot be used to compare devices of different types. Both minimum and maximum distances are provided to support cases where the process may be bound to more than one location, and the locations are at different distances from the device.

A relative distance value of \texttt{UINT16\_MAX} indicates that the distance from the process to the device could not be provided. This may be due to lack of available information (e.g., the PMIx library not having access to device locations) or other factors.

### 11.4.10 Device distance support macros

The following macros are provided to support the \texttt{pmix\_device\_distance\_t} structure.

#### Initialize the device distance structure
Initialize the \texttt{pmix\_device\_distance\_t} fields.

```c
PMIx v4.0
PMIX\_DEVICE\_DIST\_CONSTRUCT (m)
```

\textbf{IN m}
Pointer to the structure to be initialized (pointer to \texttt{pmix\_device\_distance\_t})

#### Destruct the device distance structure
Destruct the \texttt{pmix\_device\_distance\_t} fields.

```c
PMIx v4.0
PMIX\_DEVICE\_DIST\_DESTRUCT (m)
```

\textbf{IN m}
Pointer to the structure to be destructed (pointer to \texttt{pmix\_device\_distance\_t})

#### Create an device distance array
Allocate and initialize a \texttt{pmix\_device\_distance\_t} array.

```c
PMIx v4.0
PMIX\_DEVICE\_DIST\_CREATE (m, n)
```

\textbf{INOUT m}
Address where the pointer to the array of \texttt{pmix\_device\_distance\_t} structures shall be stored (handle)

\textbf{IN n}
Number of structures to be allocated (\texttt{size\_t})
Release an device distance array
Release an array of `pmix_device_distance_t` structures.

PMIx v4.0

`PMIX_DEVICE_DIST_FREE(m, n)`

IN `m`
Pointer to the array of `pmix_device_distance_t` structures (handle)

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

11.4.11 Device distance attributes

The following attributes can be used to retrieve device distances from the PMIx data store. Note that distances stored by the host environment are based on the process location at the time of start of execution and may not reflect changes to location imposed by the process itself.

`PMIX_DEVICE_DISTANCES "pmix.dev.dist" (pmix_data_array_t)`
Return an array of `pmix_device_distance_t` containing the minimum and maximum distances of the given process location to all devices of the specified type on the local node.

`PMIX_DEVICE_TYPE "pmix.dev.type" (pmix_device_type_t)`
Bitmask specifying the type(s) of device(s) whose information is being requested. Only used as a directive/qualifier.

`PMIX_DEVICE_ID "pmix.dev.id" (string)`
System-wide Universally Unique IDentifier (UUID) or node-local Operating System (OS) name of a particular device.
CHAPTER 12

Job Management and Reporting

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

12.1 Allocation Requests

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

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

PMIx attempts to address this range of use-cases with a flexible API.

12.1.1 PMIx_Allocation_request

**Summary**

Request an allocation operation from the host resource manager.

**Format**

```c
PMIx v3.0

pmix_status_t PMIx_Allocation_request(pmix_alloc_directive_t directive,
                                        pmix_info_t info[], size_t ninfo,
                                        pmix_info_t *results[], size_t *nresults);
```
IN  directive
  Allocation directive (pmix_alloc_directive_t)

IN  info
  Array of pmix_info_t structures (array of handles)

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

INOUT  results
  Address where a pointer to an array of pmix_info_t containing the results of the request can be returned (memory reference)

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

Returns one of the following:

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

--- Required Attributes ---

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

Host environments that implement support for this operation are required to support the following attributes:

**PMIX_ALLOC_REQ_ID** "pmix.alloc.reqid" (char*)
  User-provided string identifier for this allocation request which can later be used to query status of the request.

**PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (uint64_t)
  The number of nodes being requested in an allocation request.

**PMIX_ALLOC_NUM_CPUS** "pmix.alloc.ncpus" (uint64_t)
  Number of PUs being requested in an allocation request.

**PMIX_ALLOC_TIME** "pmix.alloc.time" (uint32_t)
  Total session time (in seconds) being requested in an allocation request.
Optional Attributes

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

**PMIX_ALLOC_NODE_LIST** "pmix.alloc.nlist" (char*)
Regular expression of the specific nodes being requested in an allocation request.

**PMIX_ALLOC_NUM_CPU_LIST** "pmix.alloc.ncpulist" (char*)
Regular expression of the number of PUs for each node being requested in an allocation request.

**PMIX_ALLOC_CPU_LIST** "pmix.alloc.cpulist" (char*)
Regular expression of the specific PUs being requested in an allocation request.

**PMIX_ALLOC_MEM_SIZE** "pmix.alloc.msize" (float)
Number of Megabytes\[\text{base2}\] of memory (per process) being requested in an allocation request.

**PMIX_ALLOC_FABRIC** "pmix.alloc.net" (array)
Array of pmix_info_t describing requested fabric resources. This must include at least:
PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.

**PMIX_ALLOC_FABRIC_ID** "pmix.alloc.netid" (char*)
The key to be used when accessing this requested fabric allocation. The fabric allocation will be returned/stored as a pmix_data_array_t of pmix_info_t whose first element is composed of this key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a Transmission Control Protocol (TCP) allocation might consist of a comma-delimited string of socket ranges such as "32000-32100,33005,38123-38146". Additional array entries will consist of any provided resource request directives, along with their assigned values. Examples include: PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned from;
PMIX_ALLOC_FABRIC_QOS - the assigned QoS;
PMIX_ALLOC_BANDWIDTH - the allocated bandwidth;
PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the requested fabric allocation. NOTE: the array contents may differ from those requested, especially if PMIX_INFO_REQD was not set in the request.

**PMIX_ALLOC_BANDWIDTH** "pmix.alloc.bw" (float)
Fabric bandwidth (in Megabits\[\text{base2}\]/sec) for the job being requested in an allocation request.

**PMIX_ALLOC_FABRIC_QOS** "pmix.alloc.netqos" (char*)
Fabric quality of service level for the job being requested in an allocation request.

**PMIX_ALLOC_FABRIC_TYPE** "pmix.alloc.nettype" (char*)
Type of desired transport (e.g., “tcp”, “udp”) being requested in an allocation request.

**PMIX_ALLOC_FABRIC_PLANE** "pmix.alloc.netplane" (char*)
ID string for the fabric plane to be used for the requested allocation.

```
PMIX_ALLOC_FABRIC_ENDPTS  "pmix.alloc.endpts" (size_t)
Number of endpoints to allocate per process in the job.
```

```
PMIX_ALLOC_FABRIC_ENDPTS_NODE  "pmix.alloc.endpts.nd" (size_t)
Number of endpoints to allocate per node for the job.
```

```
PMIX_ALLOC_FABRIC_SEC_KEY  "pmix.alloc.nsec" (pmix_byte_object_t)
Request that the allocation include a fabric security key for the spawned job.
```

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

- Request allocation of additional resources, including memory, bandwidth, and compute. This should be accomplished in a non-blocking manner so that the application can continue to progress while waiting for resources to become available. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities. This includes extending the time limit on current resources, and/or requesting additional resources be allocated to the requesting job. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Return no-longer-required resources to the scheduler. This includes the “loan” of resources back to the scheduler with a promise to return them upon subsequent request.

If successful, the returned results for a request for additional resources must include the host resource manager’s identifier (PMIX_ALLOC_ID) that the requester can use to specify the resources in, for example, a call to PMIx_Spawn.

12.1.2 PMIx_Allocation_request_nb

Summary
Request an allocation operation from the host resource manager.
Format

PMIx Allocation request_nb(pmix_alloc_directive_t directive,
    pmix_info_t info[], size_t ninfo,
    pmix_info_cbfunc_t cbfunc, void *cbdata);

IN    directive
    Allocation directive (pmix_alloc_directive_t)

IN    info
    Array of pmix_info_t structures (array of handles)

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

IN    cbfunc
    Callback function pmix_info_cbfunc_t (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

**PMIX_ALLOC_REQ_ID** "pmix.alloc.reqid" (char*)
    User-provided string identifier for this allocation request which can later be used to query status of the request.

**PMIX_ALLOC_NUM_NODES** "pmix.alloc.nnodes" (uint64_t)
    The number of nodes being requested in an allocation request.
PMIX_ALLOC_NUM_CPUS  "pmix.alloc.ncpus" (uint64_t)
   Number of PUs being requested in an allocation request.

PMIX_ALLOC_TIME  "pmix.alloc.time" (uint32_t)
   Total session time (in seconds) being requested in an allocation request.

--- Optional Attributes ---

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

PMIX_ALLOC_NODE_LIST  "pmix.alloc.nlist" (char*)
   Regular expression of the specific nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPU_LIST  "pmix.alloc.ncpulist" (char*)
   Regular expression of the number of PUs for each node being requested in an allocation request.

PMIX_ALLOC_CPU_LIST  "pmix.alloc.cpulist" (char*)
   Regular expression of the specific PUs being requested in an allocation request.

PMIX_ALLOC_MEM_SIZE  "pmix.alloc.msize" (float)
   Number of Megabytes[base2] of memory (per process) being requested in an allocation request.

PMIX_ALLOC_FABRIC  "pmix.alloc.net" (array)
   Array of pmix_info_t describing requested fabric resources. This must include at least:
   PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
   PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.

PMIX_ALLOC_FABRIC_ID  "pmix.alloc.netid" (char*)
   The key to be used when accessing this requested fabric allocation. The fabric allocation
   will be returned/stored as a pmix_data_array_t of pmix_info_t whose first
   element is composed of this key and the allocated resource description. The type of the
   included value depends upon the fabric support. For example, a TCP allocation might
   consist of a comma-delimited string of socket ranges such as "32000–32100,
   33005, 38123–38146". Additional array entries will consist of any provided resource
   request directives, along with their assigned values. Examples include:
   PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
   PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
   from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
   the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
   requested fabric allocation. NOTE: the array contents may differ from those requested,
   especially if PMIX_INFO_REQD was not set in the request.

PMIX_ALLOC_BANDWIDTH  "pmix.alloc.bw" (float)
   Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.
PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
Fabric quality of service level for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
ID string for the fabric plane to be used for the requested allocation.

PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
Number of endpoints to allocate per process in the job.

PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
Number of endpoints to allocate per node for the job.

PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
Request that the allocation include a fabric security key for the spawned job.

Description
Non-blocking form of the PMIx_Allocation_request API.

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

PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
User-provided string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_ID "pmix.alloc.id" (char*)
A string identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to PMIx_Spawn.

PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
Name of the WLM queue to which the allocation request is to be directed, or the queue being referenced in a query.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
The number of nodes being requested in an allocation request.

PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
Regular expression of the specific nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
Number of PUs being requested in an allocation request.

PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
Regular expression of the number of PUs for each node being requested in an allocation request.
PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
   Regular expression of the specific PUs being requested in an allocation request.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
   Number of Megabytes[base2] of memory (per process) being requested in an allocation request.

PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
   Array of pmix_info_t describing requested fabric resources. This must include at least:
      PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
      PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.

PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
   The key to be used when accessing this requested fabric allocation. The fabric allocation
   will be returned/stored as a pmix_data_array_t of pmix_info_t whose first
   element is composed of this key and the allocated resource description. The type of the
   included value depends upon the fabric support. For example, a TCP allocation might
   consist of a comma-delimited string of socket ranges such as "32000-32100,
   33005,38123-38146". Additional array entries will consist of any provided resource
   request directives, along with their assigned values. Examples include:
      PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
      PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
      from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
      the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
      requested fabric allocation. NOTE: the array contents may differ from those requested,
      especially if PMIX_INFO_REQD was not set in the request.

PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
   Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
   Fabric quality of service level for the job being requested in an allocation request.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
   Total session time (in seconds) being requested in an allocation request.

PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
   Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
   ID string for the fabric plane to be used for the requested allocation.

PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
   Number of endpoints to allocate per process in the job.

PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
   Number of endpoints to allocate per node for the job.

PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
   Request that the allocation include a fabric security key for the spawned job.
12.1.4 Job Allocation Directives

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

- **PMIX_ALLOC_NEW** A new allocation is being requested. The resulting allocation will be disjoint (i.e., not connected in a job sense) from the requesting allocation.
- **PMIX_ALLOC_EXTEND** Extend the existing allocation, either in time or as additional resources.
- **PMIX_ALLOC_RELEASE** Release part of the existing allocation. Attributes in the accompanying `pmix_info_t` array may be used to specify permanent release of the identified resources, or “lending” of those resources for some period of time.
- **PMIX_ALLOC_REAQUIRE** Reacquire resources that were previously “lent” back to the scheduler.
- **PMIX_ALLOC_EXTERNAL** A value boundary above which implementers are free to define their own directive values.

12.2 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.
12.2.1  PMIx_Job_control

Summary
Request a job control action.

Format

PMIx v3.0

```c
pmix_status_t
PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,
                 const pmix_info_t directives[], size_t ndirs,
                 pmix_info_t *results[], size_t *nresults);
```

**IN**  targets
Array of proc structures (array of handles)

**IN**  ntargets
Number of elements in the targets array (integer)

**IN**  directives
Array of info structures (array of handles)

**IN**  ndirs
Number of elements in the directives array (integer)

**INOUT**  results
Address where a pointer to an array of pmix_info_t containing the results of the request
  can be returned (memory reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request was processed by the host environment and
  returned success. Details of the result will be returned in the results array

- a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

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

Host environments that implement support for this operation are required to support the following
attributes:

- **PMIX_JOB_CTRL_ID**  "pmix.jctrl.id" (char*)


Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

**PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
Pause the specified processes.

**PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
Resume ("un-pause") the specified processes.

**PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)
Forcibly terminate the specified processes and cleanup.

**PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)
Send given signal to specified processes.

**PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)
Politely terminate the specified processes.

**PMIX_REGISTER_CLEANUP** "pmix.reg.cleanup" (char*)
Comma-delimited list of files to be removed upon process termination.

**PMIX_REGISTER_CLEANUP_DIR** "pmix.reg.cleanupdir" (char*)
Comma-delimited list of directories to be removed upon process termination.

**PMIX_CLEANUP_RECURSIVE** "pmix.clnup.recurse" (bool)
Recursively cleanup all subdirectories under the specified one(s).

**PMIX_CLEANUP_EMPTY** "pmix.clnup.empty" (bool)
Only remove empty subdirectories.

**PMIX_CLEANUP_IGNORE** "pmix.clnup.ignore" (char*)
Comma-delimited list of filenames that are not to be removed.

**PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)
When recursively cleaning subdirectories, do not remove the top-level directory (the one given in the cleanup request).

---

**Optional Attributes**

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

**PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
Cancel the specified request - the provided request ID must match the **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control**. An ID of NULL implies cancel all requests from this requestor.

**PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)
Restart the specified processes using the given checkpoint ID.

---

CHAPTER 12. JOB MANAGEMENT AND REPORTING 207
PMIX_JOB_CTRL_CHECKPOINT  "pmix.jctrl.ckpt" (char*)
    Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT  "pmix.jctrl.ckptev" (bool)
    Use event notification to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_SIGNAL  "pmix.jctrl.ckptsig" (int)
    Use the given signal to trigger a process checkpoint.

PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT  "pmix.jctrl.ckptsig" (int)
    Time in seconds to wait for a checkpoint to complete.

PMIX_JOB_CTRL_CHECKPOINT_METHOD
    "pmix.jctrl.ckmethod" (pmix_data_array_t)
    Array of pmix_info_t declaring each method and value supported by this application.

PMIX_JOB_CTRL_PROVISION  "pmix.jctrl.pvn" (char*)
    Regular expression identifying nodes that are to be provisioned.

PMIX_JOB_CTRL_PROVISION_IMAGE  "pmix.jctrl.pvnimg" (char*)
    Name of the image that is to be provisioned.

PMIX_JOB_CTRL_PREEMPTIBLE  "pmix.jctrl.preempt" (bool)
    Indicate that the job can be pre-empted.

Description
Request a job control action. The targets array identifies the processes to which the requested job
control action is to be applied. All clones of an identified process are to have the requested action
applied to them. 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 returned
status indicates whether or not the request was granted, and information as to the reason for any
denial of the request shall be returned in the results array.

12.2.2  PMIx_Job_control_nb
Summary
Request a job control action.
pmix_status_t

PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,
                     const pmix_info_t directives[], size_t ndirs,
                     pmix_info_cbfunc_t cbfunc, void *cbdata);

IN  targets
    Array of proc structures (array of handles)

IN  ntargets
    Number of elements in the targets array (integer)

IN  directives
    Array of info structures (array of handles)

IN  ndirs
    Number of elements in the directives array (integer)

IN  cbfunc
    Callback function pmix_info_cbfunc_t (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will *not* be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will *not* be called

---

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

**PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)

Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.
PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
    Pause the specified processes.

PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
    Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
    Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
    Send given signal to specified processes.

PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
    Politely terminate the specified processes.

PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
    Comma-delimited list of files to be removed upon process termination.

PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)
    Comma-delimited list of directories to be removed upon process termination.

PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)
    Recursively cleanup all subdirectories under the specified one(s).

PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)
    Only remove empty subdirectories.

PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)
    Comma-delimited list of filenames that are not to be removed.

PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
    When recursively cleaning subdirectories, do not remove the top-level directory (the one
given in the cleanup request).

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

PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
    Cancel the specified request - the provided request ID must match the
    PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control. An ID of
    NULL implies cancel all requests from this requestor.

PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
    Restart the specified processes using the given checkpoint ID.

PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
    Checkpoint the specified processes and assign the given ID to it.

PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
Use event notification to trigger a process checkpoint.

```
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
```

Use the given signal to trigger a process checkpoint.

```
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
```

Time in seconds to wait for a checkpoint to complete.

```
PMIX_JOB_CTRL_CHECKPOINT_METHOD
"pmix.jctrl.ckmethod" (pmix_data_array_t)
```

Array of `pmix_info_t` declaring each method and value supported by this application.

```
PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
```

Regular expression identifying nodes that are to be provisioned.

```
PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
```

Name of the image that is to be provisioned.

```
PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
```

Indicate that the job can be pre-empted.

---

**Description**

Non-blocking form of the `PMIx_Job_control` API. The `targets` array identifies the processes to which the requested job control action is to be applied. All `clones` of an identified process are to have the requested action applied to them. 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.

### 12.2.3 Job control constants

The following constants are specifically defined for return by the job control APIs:

```
PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES
```

Conflicting directives given for job/process cleanup.
12.2.4 Job control events

The following job control events may be available for registration, depending upon implementation and host environment support:

- **PMIX_JCTRL_CHECKPOINT** monitored by PMIx client to trigger a checkpoint operation.
- **PMIX_JCTRL_CHECKPOINT_COMPLETE** sent by a PMIx client and monitored by a PMIx server to notify that requested checkpoint operation has completed.
- **PMIX_JCTRL_PREEMPT_ALERT** monitored by a PMIx client to detect that an RM intends to preempt the job.
- **PMIX_ERR_PROC_RESTART** error in process restart.
- **PMIX_ERR_PROC_CHECKPOINT** error in process checkpoint.
- **PMIX_ERR_PROC_MIGRATE** error in process migration.

12.2.5 Job control attributes

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

- **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
  Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

- **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
  Pause the specified processes.

- **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
  Resume ("un-pause") the specified processes.

- **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
  Cancel the specified request - the provided request ID must match the **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control**. An ID of NULL implies cancel all requests from this requestor.

- **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)
  Forcibly terminate the specified processes and cleanup.

- **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)
  Restart the specified processes using the given checkpoint ID.

- **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)
  Checkpoint the specified processes and assign the given ID to it.

- **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)
  Use event notification to trigger a process checkpoint.

- **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)
  Use the given signal to trigger a process checkpoint.

- **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)
  Time in seconds to wait for a checkpoint to complete.

- **PMIX_JOB_CTRL_CHECKPOINT_METHOD** "pmix.jctrl.ckmethod" (pmix_data_array_t)
Array of `pmix_info_t` declaring each method and value supported by this application.

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

### 12.3 Process and Job Monitoring

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

The following APIs allow applications to request monitoring, directing what is to be monitored, the frequency of the associated check, whether or not the application is to be notified (via the event notification subsystem) of stall detection, and other characteristics of the operation.

#### 12.3.1 PMIx_Process_monitor

**Summary**

Request that application processes be monitored.
Format

PMIx v3.0

```
pmix_status_t
PMIx_Process_monitor(const pmix_info_t *monitor,
    pmix_status_t error,
    const pmix_info_t directives[], size_t ndirs,
    pmix_info_t *results[], size_t *nresults);
```

**IN** monitor
  info (handle)

**IN** error
  status (integer)

**IN** directives
  Array of info structures (array of handles)

**IN** ndirs
  Number of elements in the directives array (integer)

**INOUT** results
  Address where a pointer to an array of pmix_info_t containing the results of the request can be returned (memory reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request was processed and returned success. Details of the result will be returned in the results array
- a PMIx error constant indicating either an error in the input or that the request was refused

**Optional Attributes**

The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is required to add the 

**PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

- **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
  Provide a string identifier for this request.

- **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
  Identifier to be canceled (NULL means cancel all monitoring for this process).

- **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
The application desires to control the response to a monitoring event - i.e., the application is requesting that the host environment not take immediate action in response to the event (e.g., terminating the job).

\[
\begin{align*}
\text{PMIX_MONITOR_HEARTBEAT} & \ "pmix.monitor.mbeat" \ (\text{void}) \\
& \text{Register to have the PMIx server monitor the requestor for heartbeats.} \\
\text{PMIX_MONITOR_HEARTBEAT_TIME} & \ "pmix.monitor.btime" \ (\text{uint32_t}) \\
& \text{Time in seconds before declaring heartbeat missed.} \\
\text{PMIX_MONITOR_HEARTBEAT_DROPS} & \ "pmix.monitor.bdrop" \ (\text{uint32_t}) \\
& \text{Number of heartbeats that can be missed before generating the event.} \\
\text{PMIX_MONITOR_FILE} & \ "pmix.monitor.fmon" \ (\text{char*)} \\
& \text{Register to monitor file for signs of life.} \\
\text{PMIX_MONITOR_FILE_SIZE} & \ "pmix.monitor.fsize" \ (\text{bool}) \\
& \text{Monitor size of given file is growing to determine if the application is running.} \\
\text{PMIX_MONITOR_FILE_ACCESS} & \ "pmix.monitor.faccess" \ (\text{char*)} \\
& \text{Monitor time since last access of given file to determine if the application is running.} \\
\text{PMIX_MONITOR_FILE_MODIFY} & \ "pmix.monitor.fmod" \ (\text{char*)} \\
& \text{Monitor time since last modified of given file to determine if the application is running.} \\
\text{PMIX_MONITOR_FILE_CHECK_TIME} & \ "pmix.monitor.ftime" \ (\text{uint32_t}) \\
& \text{Time in seconds between checking the file.} \\
\text{PMIX_MONITOR_FILE_DROPS} & \ "pmix.monitor.fdrop" \ (\text{uint32_t}) \\
& \text{Number of file checks that can be missed before generating the event.} \\
\text{PMIX_SEND_HEARTBEAT} & \ "pmix.monitor.beat" \ (\text{void}) \\
& \text{Send heartbeat to local PMIx server.}
\end{align*}
\]

**Description**

Request that application processes be monitored via several possible methods. For example, that the server monitor this process for periodic heartbeats as an indication that the process has not become “wedged”. When a monitor detects the specified alarm condition, it will generate an event notification using the provided error code and passing along any available relevant information. It is up to the caller to register a corresponding event handler.

The *monitor* argument is an attribute indicating the type of monitor being requested. For example, `PMIX_MONITOR_FILE` to indicate that the requestor is asking that a file be monitored.

The *error* argument is the status code to be used when generating an event notification alerting that the monitor has been triggered. The range of the notification defaults to `PMIX_RANGE_NAMESPACE`. This can be changed by providing a `PMIX_RANGE` directive.
The directives argument characterizes the monitoring request (e.g., monitor file size) and frequency of checking to be done.

The returned status indicates whether or not the request was granted, and information as to the reason for any denial of the request shall be returned in the results array.

### 12.3.2 PMIx_Process_monitor_nb

**Summary**

Request that application processes be monitored.

**Format**

```c
PMIx v2.0

```pmix_status_t PMIx_Process_monitor_nb(const pmix_info_t *monitor,
                        pmix_status_t error,
                        const pmix_info_t directives[],
                        size_t ndirs,
                        pmix_info_cbfunc_t cbfunc, void *cbdata);
```pmix_status_t

**IN** monitor
info (handle)

**IN** error
status (integer)

**IN** directives
Array of info structures (array of handles)

**IN** ndirs
Number of elements in the directives array (integer)

**IN** cbfunc
Callback function pmix_info_cbfunc_t (function reference)

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

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called.
Optional Attributes

The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is required to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the requesting process:

- **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
  - Provide a string identifier for this request.

- **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
  - Identifier to be canceled (NULL means cancel all monitoring for this process).

- **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
  - The application desires to control the response to a monitoring event - i.e., the application is requesting that the host environment not take immediate action in response to the event (e.g., terminating the job).

- **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)
  - Register to have the PMIx server monitor the requestor for heartbeats.

- **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)
  - Time in seconds before declaring heartbeat missed.

- **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)
  - Number of heartbeats that can be missed before generating the event.

- **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)
  - Register to monitor file for signs of life.

- **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)
  - Monitor size of given file is growing to determine if the application is running.

- **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)
  - Monitor time since last access of given file to determine if the application is running.

- **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)
  - Monitor time since last modified of given file to determine if the application is running.

- **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)
  - Time in seconds between checking the file.

- **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)
  - Number of file checks that can be missed before generating the event.

- **PMIX_SEND_HEARTBEAT** "pmix.monitor.beat" (void)
  - Send heartbeat to local PMIx server.
**Description**

Non-blocking form of the `PMIx_Process_monitor` API. The `cbfunc` function provides a `status` to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of `pmix_info_t` structures.

12.3.3 `PMIx_Heartbeat`

**Summary**

Send a heartbeat to the PMIx server library

**Format**

```
PMIx v2.0
```

```
PMIx_Heartbeat();
```

**Description**

A simplified macro wrapping `PMIx_Process_monitor_nb` that sends a heartbeat to the PMIx server library.

12.3.4 Monitoring events

The following monitoring events may be available for registration, depending upon implementation and host environment support:

- `PMIX_MONITOR_HEARTBEAT_ALERT` Heartbeat failed to arrive within specified window. The process that triggered this alert will be identified in the event.
- `PMIX_MONITOR_FILE_ALERT` File failed its monitoring detection criteria. The file that triggered this alert will be identified in the event.

12.3.5 Monitoring attributes

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

```
PMIX_MONITOR_ID "pmix.monitor.id" (char*)
```

Provide a string identifier for this request.

```
PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)
```

Identifier to be canceled (NULL means cancel all monitoring for this process).

```
PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)
```

The application desires to control the response to a monitoring event - i.e., the application is requesting that the host environment not take immediate action in response to the event (e.g., terminating the job).

```
PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)
```

Register to have the PMIx server monitor the requestor for heartbeats.
PMIX_SEND_HEARTBEAT "pmix.monitor.beat" (void)
Send heartbeat to local PMIx server.

PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)
Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)
Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)
Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)
Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)
Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)
Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)
Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)
Number of file checks that can be missed before generating the event.

12.4 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.

12.4.1 PMIx_Log
Summary
Log data to a data service.

Format

PMIx v3.0

pmix_status_t
PMIx_Log(const pmix_info_t data[], size_t ndata,
const pmix_info_t directives[], size_t ndirs);

IN data
Array of info structures (array of handles)

IN ndata
Number of elements in the data array (size_t)
IN directives
    Array of info structures (array of handles)
IN ndirs
    Number of elements in the directives array (size_t)

Return codes are one of the following:

- **PMIX_SUCCESS** The logging request was successful.
- **PMIX_ERR_BAD_PARAM** The logging request contains at least one incorrect entry.
- **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation or host environment does not support this function.
- other appropriate PMIx error code

---

### Required Attributes
---

If the PMIx library does not itself perform this operation, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed info array:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  - Effective user ID of the connecting process.
- **PMIX_GRPID** "pmix.egid" (uint32_t)
  - Effective group ID of the connecting process.

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

- **PMIX_LOG_STDERR** "pmix.log.stderr" (char*)
  - Log string to stderr.
- **PMIX_LOG_STDOUT** "pmix.log.stdout" (char*)
  - Log string to stdout.
- **PMIX_LOG_SYSLOG** "pmix.log.syslog" (char*)
  - Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available, otherwise to local syslog.
- **PMIX_LOG_LOCAL_SYSLOG** "pmix.log.lsys" (char*)
  - Log data to local syslog. Defaults to ERROR priority.
- **PMIX_LOG_GLOBAL_SYSLOG** "pmix.log.gsys" (char*)
  - Forward data to system “gateway” and log msg to that syslog Defaults to ERROR priority.
- **PMIX_LOG_SYSLOG_PRI** "pmix.log.syspri" (int)
  - Syslog priority level.
- **PMIX_LOG_ONCE** "pmix.log.once" (bool)
  - Only log this once with whichever channel can first support it, taking the channels in priority order.
Optional Attributes

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

- **PMIX_LOG_SOURCE** "pmix.log.source" (pmix_proc_t*)
  ID of source of the log request.

- **PMIX_LOG_TIMESTAMP** "pmix.log.tstmp" (time_t)
  Timestamp for log report.

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

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

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

- **PMIX_LOG_XML_OUTPUT** "pmix.log.xml" (bool)
  Print the output stream in eXtensible Markup Language (XML) format.

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

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

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

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

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

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

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

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

- **PMIX_LOG_GLOBAL_DATASTORE** "pmix.log.gstore" (bool)
  Store the log data in a global data store (e.g., database).
Description
Log data subject to the services offered by the host environment. The data to be logged is provided in the data array. The (optional) directives can be used to direct the choice of logging channel.

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

12.4.2 PMIx_Log_nb

Summary
Log data to a data service.

Format
PMIx v2.0

```
#include <pmix/pmix_api.h>

pmix_status_t PMIx_Log_nb(const pmix_info_t data[], size_t ndata,
                          const pmix_info_t directives[], size_t ndirs,
                          pmix_op_cbfunc_t cbfunc, void *cbdata);
```

IN data
Array of info structures (array of handles)

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

IN directives
Array of info structures (array of handles)

IN ndirs
Number of elements in the directives array (size_t)

IN cbfunc
Callback function pmix_op_cbfunc_t (function reference)

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

Return codes are one of the following:

PMIX_SUCCESS The logging request is valid and is being processed. The resulting status from the operation will be provided in the callback function. Note that the library must not invoke the callback function prior to returning from the API.
PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
returned success - the cbfunc will not be called

PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry that prevents
it from being processed. The callback function will not be called.

PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function. The
callback function will not be called.

other appropriate PMIx error code - the callback function will not be called.

Required Attributes

If the PMIx library does not itself perform this operation, then it is required to pass any attributes
provided by the client to the host environment for processing. In addition, it must include the
following attributes in the passed info array:

PMIX_USERID "pmix.euid" (uint32_t)
    Effective user ID of the connecting process.

PMIX_GRPID "pmix.egid" (uint32_t)
    Effective group ID of the connecting process.

Host environments or PMIx libraries that implement support for this operation are required to
support the following attributes:

PMIX_LOG_STDERR "pmix.log.stderr" (char*)
    Log string to stderr.

PMIX_LOGSTDOUT "pmix.log.stdout" (char*)
    Log string to stdout.

PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
    Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
    otherwise to local syslog.

PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)
    Log data to local syslog. Defaults to ERROR priority.

PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
    Forward data to system “gateway” and log msg to that syslog Defaults to ERROR priority.

PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)
    Syslog priority level.

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

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

**PMIX_LOG_SOURCE** "pmix.log.source" (pmix_proc_t*)
   ID of source of the log request.

**PMIX_LOG_TIMESTAMP** "pmix.log.tstmp" (time_t)
   Timestamp for log report.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Advice to users

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

### 12.4.3 Log attributes

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

- `PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)`
  ID of source of the log request.

- `PMIX_LOG_STDERR "pmix.log.stderr" (char*)`
  Log string to stderr.

- `PMIX_LOG_STDOUT "pmix.log.stdout" (char*)`
  Log string to stdout.

- `PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)`
  Log data to syslog. Defaults to `ERROR` priority. Will log to global syslog if available, otherwise to local syslog.

- `PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)`
  Log data to local syslog. Defaults to `ERROR` priority.

- `PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)`
  Forward data to system “gateway” and log msg to that syslog Defaults to `ERROR` priority.

- `PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)`
  Syslog priority level.

- `PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)`
  Timestamp for log report.

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

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

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

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

PMIX_LOG_ONCE "pmix.log.once" (bool)
    Only log this once with whichever channel can first support it, taking the channels in priority order.
PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
    Message blob to be sent somewhere.
PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
    Log via email based on pmix_info_t containing directives.
PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
    Comma-delimited list of email addresses that are to receive the message.
PMIX_LOG_EMAIL_SENDER_ADDR "pmix.log.emfaddr" (char*)
    Return email address of sender.
PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
    Subject line for email.
PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
    Message to be included in email.
PMIX_LOG_EMAIL_SERVER "pmix.log.esrvr" (char*)
    Hostname (or IP address) of SMTP server.
PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (int32_t)
    Port the email server is listening to.
PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
    Store the log data in a global data store (e.g., database).
PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
    Log the provided information to the host environment’s job record.
CHAPTER 13

Process Sets and Groups

PMIx supports two slightly related, but functionally different concepts known as process sets and process groups. This chapter defines 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 or host environment assigned label associated with a given set of application processes. Processes can belong to multiple process sets at a time. Users may define a PMIx process set at time of application execution. For example, if using the command line parallel launcher "prun", one could specify process sets as follows:

```
$ 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_NAMES attribute with a value 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 process using PMIx_Get. Alternatively, other executing applications could utilize the PMIx_Query_info APIs 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.

Host environments can create or delete process sets at any time through the PMIx_server_define_process_set and PMIx_server_delete_process_set APIs. PMIx servers shall notify all local clients of process set operations via the PMIX_PROCESS_SET_DEFINE or PMIX_PROCESS_SET_DELETE events.

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.

- Process set identifiers are set by the host environment or by the user at time of application submission for execution - there are no PMIx APIs provided by which an application can define a process set or change a process set membership. In contrast, PMIx process groups can only be defined dynamically by the application.
Process sets are immutable - members cannot be added or removed once the set has been defined. In contrast, PMIx process groups can dynamically change their membership using the appropriate APIs.

Process groups can be used in calls to PMIx operations. Members of process groups that are involved in an operation are translated by their PMIx server into their native identifier prior to the operation being passed to the host environment. For example, an application can define a process group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the group id of foo. If the application subsequently calls the PMIx_Fence API with a process identifier of \{foo, PMIX_RANK_WILDCARD\}, the PMIx server will replace that identifier with an array consisting of \{210456, 0\} and \{210456, 1\} - the host-assigned identifiers of the participating processes - prior to processing the request.

Process groups can request that the host environment assign a unique size_t Process Group Context IDentifier (PGCID) to the group at time of group construction. An Message Passing Interface (MPI) library may, for example, use the PGCID as the MPI communicator identifier for the group.

The two concepts do, however, overlap in that they both involve collections of processes. Users desiring to create a process group based on a process set could, for example, obtain the membership array of the process set and use that as input to PMIx_Group_construct, perhaps including the process set name as the group identifier for clarity. Note that no linkage between the set and group of the same name is implied nor maintained - e.g., changes in process group membership cannot be reflected in the process set using the same identifier.

--- Advice to PMIx server hosts ---

The host environment is responsible for ensuring:

- consistent knowledge of process set membership across all involved PMIx servers; and

- that process set names do not conflict with system-assigned namespaces within the scope of the set.

---

13.1.1 Process Set Constants

PMIx v4.0

The PMIx server is required to send a notification to all local clients upon creation or deletion of process sets. Client processes wishing to receive such notifications must register for the corresponding event:

- **PMIX_PROCESS_SET_DEFINE** - The host environment has defined a new process set - the event will include the process set name (PMIX_PSET_NAME) and the membership (PMIX_PSET_MEMBERS).
- **PMIX_PROCESS_SET_DELETE** - The host environment has deleted a process set - the event will include the process set name (PMIX_PSET_NAME).
13.1.2 Process Set Attributes

PMIx v4.0

Several attributes are provided for querying the system regarding process sets using the
PMIx_Query_info APIs.

PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)

PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)

PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t*)

The PMIX_PROCESS_SET_DEFINE event shall include the name of the newly defined process
set and its members:

PMIX_PSET_NAME "pmix.pset.nm" (char*)

The name of the newly defined process set.

PMIX_PSET_MEMBERS "pmix.pset.mems" (pmix_data_array_t*)

An array of pmix_proc_t containing the members of the newly defined process set.

In addition, a process can request (via PMIx_Get) the process sets to which a given process
(including itself) belongs:

PMIX_PSET_NAMES "pmix.pset.nms" (pmix_data_array_t*)

Returns an array of char* string names of the process sets in which the given process is a
member.

13.2 Process Groups

PMIx Groups are defined as a collection of processes desiring a common, unique identifier for
operational purposes such as passing events or participating in PMIx fence operations. As with
processes that assemble via PMIx_Connect, each member of the group is provided with both the
job-level information of any other namespace represented in the group, and the contact information
for all group members.

However, members of PMIx Groups are loosely coupled as opposed to tightly connected when
constructed via PMIx_Connect. Thus, groups differ from PMIx_Connect assemblages in
several key areas, as detailed in the following sections.

13.2.1 Relation to the host environment

Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed, the
server creates a tracker that associates the specified processes with the user-provided group
identifier, and assigns a new group rank based on their relative position in the array of processes
provided in the call to PMIx_Group_construct. Members of the group can subsequently
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.

In contrast, 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.

Advice to users

User-provided group identifiers must be distinct from both other group identifiers within the system
and namespaces 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”

13.2.2 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.

In contrast, PMIx Groups are designed to be more flexible in their construction procedure by
relaxing these constraints. While a standard blocking form of constructing groups is provided, the
event notification system is utilized to provide a designated group leader with the ability to replace
participants that fail to participate within a given timeout period. This provides a mechanism by
which the application can, if desired, replace members on-the-fly or allow the group to proceed
with partial membership. In such cases, the final group membership is returned to all participants
upon completion of the operation.

Additionally, PMIx supports dynamic definition of group membership based on an invite/join
model. A process can asynchronously initiate construction of a group of any processes via the
PMIx_Group_invite function call. Invitations are delivered via a PMIx event (using the
PMIX_GROUP_INVITED event) to the invited processes which can then either accept or decline
the invitation using the PMIx_Group_join API. The initiating process tracks responses by
registering for the events generated by the call to PMIx_Group_join, timeouts, or process
terminations, optionally replacing processes that decline the invitation, fail to respond in time, or terminate without responding. Upon completion of the operation, the final list of participants is communicated to each member of the new group.

### 13.2.3 Destruct procedure

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.

In contrast, 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`.

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.

#### 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.

### 13.2.4 Process Group Events

**PMIx v4.0** Asynchronous process group operations rely heavily on PMIx events. The following events have been defined for that purpose.

- **PMIX_GROUP_INVITED** The process has been invited to join a PMIx Group - the identifier of the group and the ID’s of other invited (or already joined) members will be included in the notification.
- **PMIX_GROUP_LEFT** A process has asynchronously left a PMIx Group - the process identifier of the departing process will in included in the notification.
- **PMIX_GROUP_MEMBER_FAILED** A member of a PMIx Group has abnormally terminated (i.e., without formally leaving the group prior to termination) - the process identifier of the failed process will be included in the notification.
PMIX_GROUP_INVITE_ACCEPTED  A process has accepted an invitation to join a PMIx Group - the identifier of the group being joined will be included in the notification.

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

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

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

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

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

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

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

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

13.2.5 Process Group Attributes

Attributes for querying the system regarding process groups include:

PMIX_QUERY_NUM_GROUPS "pmix.qry.pgrpnum" (size_t)
Return the number of process groups defined in the specified range (defaults to session). OPTIONAL QUALIFIERS: PMIX_RANGE.

PMIX_QUERY_GROUP_NAMES "pmix.qry.pgrp" (pmix_data_array_t*)
Return a pmix_data_array_t containing an array of string names of the process groups defined in the specified range (defaults to session). OPTIONAL QUALIFIERS:
PMIX_RANGE.

PMIX_QUERY_GROUP_MEMBERSHIP
"pmix.qry.pgrpmems" (pmix_data_array_t*)
Return a pmix_data_array_t of pmix_proc_t containing the members of the specified process group. REQUIRED QUALIFIERS: PMIX_GROUP_ID.

The following attributes are used as directives in PMIx Group operations:

PMIX_GROUP_ID "pmix.grp.id" (char*)
User-provided group identifier - as the group identifier may be used in PMIx operations, the user is required to ensure that the provided ID is unique within the scope of the host environment (e.g., by including some user-specific or application-specific prefix or suffix to the string).

**PMIX_GROUP_LEADER**  "pmix.grp.ldr"  (bool)
This process is the leader of the group.

**PMIX_GROUP_OPTIONAL**  "pmix.grp.opt"  (bool)
Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false.

**PMIX_GROUP_NOTIFY_TERMINATION**  "pmix.grp.notterm"  (bool)
Notify remaining members when another member terminates without first leaving the group.

**PMIX_GROUP_FT_COLLECTIVE**  "pmix.grp.ftcoll"  (bool)
Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

**PMIX_GROUP_MEMBERSHIP**  "pmix.grp.mbrs"  (pmix_data_array_t*)
Array pmix_proc_t identifiers identifying the members of the specified group.

**PMIX_GROUP_ASSIGN_CONTEXT_ID**  "pmix.grp.actxid"  (bool)
Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

**PMIX_GROUP_LOCAL_ONLY**  "pmix.grp.lcl"  (bool)
Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

The following attributes are used to return information at the conclusion of a PMIx Group operation and/or in event notifications:

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

**PMIX_GROUP_ENDPT_DATA**  "pmix.grp.endpt"  (pmix_byte_object_t)
Data collected during group construction to ensure communication between group members is supported upon completion of the operation.

In addition, a process can request (via **PMIx_Get**) the process groups to which a given process (including itself) belongs:

**PMIX_GROUP_NAMES**  "pmix.pgrp.nm"  (pmix_data_array_t*)
Returns an array of char* string names of the process groups in which the given process is a member.

13.2.6 PMIx_Group_construct

Summary
Construct a PMIx process group.

Format

PMIx v4.0

```
pmix_status_t
PMIx_Group_construct(const char grp[],
    const pmix_proc_t procs[], size_t nprocs,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_info_t **results,
    size_t *nresults);
```

IN grp
NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the group identifier (string)

IN procs
Array of pmix_proc_t structures containing the PMIx identifiers of the member processes (array of handles)

IN nprocs
Number of elements in the procs array (size_t)

IN directives
Array of pmix_info_t structures (array of handles)

IN ndirs
Number of elements in the directives array (size_t)

INOUT results
Pointer to a location where the array of pmix_info_t describing the results of the operation is to be returned (pointer to handle)

INOUT nresults
Pointer to a size_t location where the number of elements in results is to be returned (memory reference)

Returns one of the following:

- PMIX_SUCCESS, indicating that the request has been successfully completed
- PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this operation
- a PMIx error constant indicating either an error in the input or that the request failed to be completed
Required Attributes

The following attributes are required to be supported by all PMIx libraries that support this operation:

**PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)
This process is the leader of the group.

**PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)
Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false.

**PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)
Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

**PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)
Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are required to support the following attributes:

**PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)
Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.

**PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)
Notify remaining members when another member terminates without first leaving the group.

Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)
Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.
Description
Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, NULL-terminated character array of length less than or equal to PMIX_MAX_NSLEN. Only characters accepted by standard string comparison functions (e.g., strncmp) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with a unique group ID. The directives array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

If the PMIX_GROUP_NOTIFY_TERMINATION attribute is provided and has a value of true, then either the construct leader (if PMIX_GROUP_LEADER is provided) or all participants who register for the PMIX_GROUP_MEMBER_FAILED event will receive events whenever a process fails or terminates prior to calling PMIx_Group_construct – i.e. if a group leader is declared, only that process will receive the event. In the absence of a declared leader, all specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the PMIX_GROUP_MEMBERSHIP attribute) as part of the results array returned by this API.

Failure of the declared leader at any time will cause a PMIX_GROUP_LEADER_FAILED event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the PMIX_GROUP_LEADER attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a PMIX_GROUP_LEADER_SELECTED event identifying the new leader. If no leader was selected, then the pmix_info_t provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns PMIX_GROUP_CONSTRUCT_ABORT from either the PMIX_GROUP_MEMBER_FAILED or the PMIX_GROUP_LEADER_FAILED event handler will cause the construct process to abort, returning from the call with a PMIX_GROUP_CONSTRUCT_ABORT status.

If the PMIX_GROUP_NOTIFY_TERMINATION attribute is not provided or has a value of false, then the PMIx_Group_construct operation will simply return an error whenever a proposed group member fails or terminates prior to calling PMIx_Group_construct.

Providing the PMIX_GROUP_OPTIONAL attribute with a value of true directs the PMIx library to consider participation by any specified group member as non-required - thus, the operation will return PMIX_SUCCESS if all members participate, or PMIX_ERR_PARTIAL_SUCCESS if some
members fail to participate. The results array will contain the final group membership in the latter case. Note that this use-case can cause the operation to hang if the PMIx_TIMEOUT attribute is not specified and one or more group members fail to call PMIx_Group_construct while continuing to execute. Also, note that no leader or member failed events will be generated during the operation.

Processes in a group under construction are not allowed to leave the group until group construction is complete. Upon completion of the construct procedure, each group member will have access to the job-level information of all namespaces represented in the group plus any information posted via PMIx_Put (subject to the usual scoping directives) for every group member.

--- Advice to PMIx library implementers ---

At the conclusion of the construct operation, the PMIx library is required to ensure that job-related information from each participating namespace plus any information posted by group members via PMIx_Put (subject to scoping directives) is available to each member via calls to PMIx_Get.

--- Advice to PMIx server hosts ---

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a signature for such operations may experience potential conflicts should both a PMIx_Group_construct and a PMIx_Fence operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

13.2.7 PMIx_Group_construct_nb

Summary

Non-blocking form of PMIx_Group_construct.
Format

```c
pmix_status_t
PMIx_Group_construct_nb(const char grp[],
                         const pmix_proc_t procs[], size_t nprocs,
                         const pmix_info_t directives[],
                         size_t ndirs,
                         pmix_info_cbfunc_t cbfunc, void *cbdata);
```

**IN** `grp`
- NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

**IN** `procs`
- Array of `pmix_proc_t` structures containing the PMIx identifiers of the member processes (array of handles)

**IN** `nprocs`
- Number of elements in the `procs` array (`size_t`)

**IN** `directives`
- Array of `pmix_info_t` structures (array of handles)

**IN** `ndirs`
- Number of elements in the `directives` array (`size_t`)

**IN** `cbfunc`
- Callback function `pmix_info_cbfunc_t` (function reference)

**IN** `cbdata`
- Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the `cbfunc` will not be called.

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the `cbfunc` will not be called.

- a non-zero PMIx error constant indicating a reason for the request to have been rejected - the `cbfunc` will not be called.

If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** The operation succeeded and all specified members participated.
• **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function.

• **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.

• a non-zero PMIx error constant indicating a reason for the request’s failure.

### Required Attributes

PMIx libraries that choose not to support this operation must return **PMIX_ERR_NOT_SUPPORTED** when the function is called.

The following attributes are required to be supported by all PMIx libraries that support this operation:

- **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)
  This process is the leader of the group.

- **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)
  Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false.

- **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)
  Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

- **PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)
  Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are required to provide the following attributes:

- **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)
  Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

- **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)
  Notify remaining members when another member terminates without first leaving the group.
Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  - Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

Non-blocking version of the PMIx_Group_construct operation. The callback function will be called once all group members have called either PMIx_Group_construct or PMIx_Group_construct_nb.

### 13.2.8 PMIx_Group_destruct

**Summary**

Destruct a PMIx process group.

**Format**

```c
pmix_status_t PMIx_Group_destruct(const char grp[],
                                  const pmix_info_t directives[],
                                  size_t ndirs);
```

- **IN grp**
  - NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the identifier of the group to be destructed (string)

- **IN directives**
  - Array of pmix_info_t structures (array of handles)

- **IN ndirs**
  - Number of elements in the directives array (size_t)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been successfully completed

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation

- a PMIx error constant indicating either an error in the input or that the request failed to be completed
Required Attributes

For implementations and host environments that support the operation, there are no identified required attributes for this API.

Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The directives array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling **PMIx_Group_destruct** or its non-blocking version unless the **PMIX_GROUP_NOTIFY_TERMINATION** attribute was provided (with a value of **false**) at time of group construction. If notification was requested, then the **PMIX_GROUP_MEMBER_FAILED** event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The **PMIx_Group_destruct** operation will subsequently return **PMIX_SUCCESS** when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

**Advice to PMIx server hosts**

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a signature for such operations may experience potential conflicts should both a **PMIx_Group_destruct** and a **PMIx_Fence** operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

13.2.9 **PMIx_Group_destruct_nb**

**Summary**

Non-blocking form of **PMIx_Group_destruct**.
Format

PMIx v4.0

```c
pmix_status_t PMIx_Group_destruct_nb(const char grp[],
          const pmix_info_t directives[],
          size_t ndirs,
          pmix_op_cbfunc_t cbfunc, void *cbdata);
```

- **IN** `grp` - NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the identifier of the group to be destructed (string)
- **IN** `directives` - Array of `pmix_info_t` structures (array of handles)
- **IN** `ndirs` - Number of elements in the `directives` array (`size_t`)
- **IN** `cbfunc` - Callback function `pmix_op_cbfunc_t` (function reference)
- **IN** `cbdata` - Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed - result will be returned in the provided `cbfunc`. Note that the library *must not* invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the `cbfunc` will *not* be called
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the `cbfunc` will *not* be called.
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will *not* be called.

If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** The operation was successfully completed.
- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request’s failure.
PMIx libraries that choose not to support this operation must return `PMIX_ERR_NOT_SUPPORTED` when the function is called. For implementations and host environments that support the operation, there are no identified required attributes for this API.

The following attributes are optional for host environments that support this operation:

```
PMIX_TIMEOUT "pmix.timeout" (int)
```

Time in seconds before the specified operation should time out (zero indicating infinite) and return the `PMIX_ERR_TIMEOUT` error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**

Non-blocking version of the `PMIx_Group_destruct` operation. The callback function will be called once all members of the group have executed either `PMIx_Group_destruct` or `PMIx_Group_destruct_nb`.

**13.2.10 PMIx_Group_invite**

**Summary**

Asynchronously construct a PMIx process group.
Format

pmix_status_t
PMIx_Group_invite(const char grp[],
    const pmix_proc_t procs[], size_t nprocs,
    const pmix_info_t directives[], size_t ndirs,
    pmix_info_t **results, size_t *nresult);

IN  grp
    NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the group
    identifier (string)

IN  procs
    Array of pmix_proc_t structures containing the PMIx identifiers of the processes to be
    invited (array of handles)

IN  nprocs
    Number of elements in the procs array (size_t)

IN  directives
    Array of pmix_info_t structures (array of handles)

IN  ndirs
    Number of elements in the directives array (size_t)

INOUT results
    Pointer to a location where the array of pmix_info_t describing the results of the
    operation is to be returned (pointer to handle)

INOUT nresults
    Pointer to a size_t location where the number of elements in results is to be returned
    (memory reference)

Returns one of the following:

• PMIX_SUCCESS, indicating that the request has been successfully completed.

• PMIX_ERR_NOT_SUPPORTED The PMIx library and/or the host RM does not support this
  operation.

• a PMIx error constant indicating either an error in the input or that the request failed to be
  completed.

Required Attributes

The following attributes are required to be supported by all PMIx libraries that support this
operation:

PMIX_GROUP_OPTIONAL  "pmix.grp.opt" (bool)
    Participation is optional - do not return an error if any of the specified processes terminate
    without having joined. The default is false.
PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)
Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are required to provide the following attributes:

PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)
Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.

PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)
Notify remaining members when another member terminates without first leaving the group.

Optional Attributes
The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT "pmix.timeout" (int)
Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description
Explicitly invite the specified processes to join a group. The process making the PMIx_Group_invite call is automatically declared to be the group leader. Each invited process will be notified of the invitation via the PMIX_GROUP_INVITED event - the processes being invited must therefore register for the PMIX_GROUP_INVITED event in order to be notified of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of invite versus event registration is required.

The invitation event will include the identity of the inviting process plus the name of the group. When ready to respond, each invited process provides a response using either the blocking or non-blocking form of PMIx_Group_join. This will notify the inviting process that the invitation was either accepted (via the PMIX_GROUP_INVITE_ACCEPTED event) or declined (via the PMIX_GROUP_INVITE_DECLINED event). The PMIX_GROUP_INVITE_ACCEPTED event is captured by the PMIx client library of the inviting process - i.e., the application itself does not need to register for this event. The library will track the number of accepting processes and alert the inviting process (by returning from the blocking form of PMIx_Group_invite or calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the PMIX_GROUP_INVITE_DECLINED if the application allows invited processes to decline the invitation. This provides an opportunity for the
application to either invite a replacement, declare “abort”, or choose to remove the declining
process from the final group. The inviting process should also register to receive

**PMIX_GROUP_INVITE_FAILED** events whenever a process fails or terminates prior to
responding to the invitation. Actions taken by the inviting process in response to these events must
be communicated at the end of the event handler by returning the corresponding result so that the
PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level
information of each other’s namespaces plus any information posted via **PMIx_Put** by the other
members.

The inviting process is automatically considered the leader of the asynchronous group construction
procedure and will receive all failure or termination events for invited members prior to completion.
The inviting process is required to provide a **PMIX_GROUP_CONSTRUCT_COMPLETE** event once
the group has been fully assembled – this event is used by the PMIx library as a trigger to release
participants from their call to **PMIx_Group_join** and provides information (e.g., the final group
membership) to be returned in the **results** array.

Failure of the inviting process at any time will cause a **PMIX_GROUP_LEADER_FAILED** event to
be delivered to all participants so they can optionally declare a new leader. A new leader is
identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of
the event handler. Only one process is allowed to return that attribute, declaring itself as the new
leader. Results of the leader selection will be communicated to all participants via a
**PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected,
then the status code provided in the event handler will provide an error value so the participants can
take appropriate action.

---

**Advice to users**

Applications are not allowed to use the group in any operations until group construction is
complete. This is required in order to ensure consistent knowledge of group membership across all
participants.

---

**13.2.11 PMIx_Group_invite_nb**

**Summary**

Non-blocking form of **PMIx_Group_invite**.
pmix_status_t
PMIx_Group_invite_nb(const char grp[],
const pmix_proc_t procs[], size_t nprocs,
const pmix_info_t directives[], size_t ndirs,
pmix_info_cbfunc_t cbfunc, void *cbdata);

IN grp
NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the group
identifier (string)

IN procs
Array of pmix_proc_t structures containing the PMIx identifiers of the processes to be
invited (array of handles)

IN nprocs
Number of elements in the procs array (size_t)

IN directives
Array of pmix_info_t structures (array of handles)

IN ndirs
Number of elements in the directives array (size_t)

IN cbfunc
Callback function pmix_info_cbfunc_t (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

- PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the
  provided cbfunc. Note that the library must not invoke the callback function prior to returning
  from the API.

- PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called.

- PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the cbfunc
  will not be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called.

If executed, the status returned in the provided callback function will be one of the following
constants:

- PMIX_SUCCESS The operation succeeded and all specified members participated.
The operation succeeded but not all specified members participated - the final group membership is included in the callback function.

- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.

- a non-zero PMIx error constant indicating a reason for the request’s failure.

### Required Attributes

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

- **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)
  Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is *false*.

- **PMIX_GROUP_FT_COLLECTIVE** "pmix.grp.ftcoll" (bool)
  Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

Host environments that support this operation are *required* to provide the following attributes:

- **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)
  Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, `size_t` value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

- **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)
  Notify remaining members when another member terminates without first leaving the group.

### Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

### Description

Non-blocking version of the **PMIx_Group_invite** operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either **PMIx_Group_join** or **PMIx_Group_join_nb**.
13.2.12 PMIx_Group_join

Summary
Accept an invitation to join a PMIx process group.

Format

```
PMIx v4.0

pmix_status_t
PMIx_Group_join(const char grp[],
const pmix_proc_t *leader,
const pmix_group_opt_t opt,
const pmix_info_t directives[], size_t ndirs,
pmix_info_t **results, size_t *nresult);
```

\[\text{IN} \quad \text{grp}\]
NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

\[\text{IN} \quad \text{leader}\]
Process that generated the invitation (handle)

\[\text{IN} \quad \text{opt}\]
Accept or decline flag (`pmix_group_opt_t`)

\[\text{IN} \quad \text{directives}\]
Array of `pmix_info_t` structures (array of handles)

\[\text{IN} \quad \text{ndirs}\]
Number of elements in the `directives` array (`size_t`)

\[\text{INOUT} \quad \text{results}\]
Pointer to a location where the array of `pmix_info_t` describing the results of the operation is to be returned (pointer to handle)

\[\text{INOUT} \quad \text{nresults}\]
Pointer to a `size_t` location where the number of elements in `results` is to be returned (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been successfully completed.
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation.
- a PMIx error constant indicating either an error in the input or that the request failed to be completed.

---

Required Attributes

There are no identified required attributes for implementers.
Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

Respond to an invitation to join a group that is being asynchronously constructed. The process must have registered for the **PMIX_GROUP_INVITED** event in order to be notified of the invitation. When called, the event information will include the **pmix_proc_t** identifier of the process that generated the invitation along with the identifier of the group being constructed. When ready to respond, the process provides a response using either form of **PMIx_Group_join**.

Advice to users

Since the process is alerted to the invitation in a PMIx event handler, the process **must not** use the blocking form of this call unless it first “thread shifts” out of the handler and into its own thread context. Likewise, while it is safe to call the non-blocking form of the API from the event handler, the process **must not** block in the handler while waiting for the callback function to be called.

Calling this function causes the inviting process (aka the group leader) to be notified that the process has either accepted or declined the request. The blocking form of the API will return once the group has been completely constructed or the group’s construction has failed (as described below) – likewise, the callback function of the non-blocking form will be executed upon the same conditions.

Failure of the leader during the call to **PMIx_Group_join** will cause a **PMIX_GROUP_LEADER_FAILED** event to be delivered to all invited participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX_GROUP_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX_GROUP_LEADER_SELECTED** event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

Any participant that returns **PMIX_GROUP_CONSTRUCT_ABORT** from the leader failed event handler will cause all participants to receive an event notifying them of that status. Similarly, the leader may elect to abort the procedure by either returning **PMIX_GROUP_CONSTRUCT_ABORT** from the handler assigned to the **PMIX_GROUP_INVITE_ACCEPTED** or **PMIX_GROUP_INVITE_DECLINED** codes, or by generating an event for the abort code. Abort events will be sent to all invited participants.
13.2.13  PMIx_Group_join_nb

Summary
Non-blocking form of PMIx_Group_join

Format

PMIx v4.0

```c
pmix_status_t
PMIx_Group_join_nb(const char grp[],
    const pmix_proc_t *leader,
    pmix_group_opt_t opt,
    const pmix_info_t directives[], size_t ndirs,
    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

IN  grp
    NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the group
identifier (string)

IN  leader
    Process that generated the invitation (handle)

IN  opt
    Accept or decline flag (pmix_group_opt_t)

IN  directives
    Array of pmix_info_t structures (array of handles)

IN  ndirs
    Number of elements in the directives array (size_t)

IN  cbfunc
    Callback function pmix_info_cbfucnt (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed - result will be returned in the
  provided cbfunc. Note that the library must not invoke the callback function prior to returning
  from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called.

- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the cbfunc
  will not be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called.
If executed, the status returned in the provided callback function will be one of the following constants:

- **PMIX_SUCCESS** The operation succeeded and group membership is in the callback function parameters.
- **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request’s failure.

### Required Attributes

There are no identified required attributes for implementers.

### Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  
  Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

### Description

Non-blocking version of the **PMIx_Group_join** operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either **PMIx_Group_join** or **PMIx_Group_join_nb**.

#### 13.2.13.1 Group accept/decline directives

**PMIx v4.0**

The **pmix_group_opt_t** type is a `uint8_t` value used with the **PMIx_Group_join** API to indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

- **PMIX_GROUP_DECLINE** Decline the invitation.
- **PMIX_GROUP_ACCEPT** Accept the invitation.

#### 13.2.14 **PMIx_Group_leave**

**Summary**

Leave a PMIx process group.
PMIx v4.0

Format

```c
pmix_status_t PMIx_Group_leave(const char grp[],
    const pmix_info_t directives[],
    size_t ndirs);
```

IN  `grp`
    NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

IN  `directives`
    Array of `pmix_info_t` structures (array of handles)

IN  `ndirs`
    Number of elements in the `directives` array (`size_t`)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server.
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation.
- a PMIx error constant indicating either an error in the input or that the request is unsupported.

**Required Attributes**

There are no identified required attributes for implementers.

**Description**

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.
13.2.15 PMIx_Group_leave_nb

Summary
Non-blocking form of PMIx_Group_leave.

Format

PMIx v4.0

C

pmix_status_t PMIx_Group_leave_nb(const char grp[],
const pmix_info_t directives[],
size_t ndirs,
pmix_op_cbfunc_t cbfunc,
void *cbdata);

IN grp
NULL-terminated character array of maximum size PMIX_MAX_NSLEN containing the group identifier (string)

IN directives
Array of pmix_info_t structures (array of handles)

IN ndirs
Number of elements in the directives array (size_t)

IN cbfunc
Callback function pmix_op_cbfunc_t (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

• PMIX_SUCCESS, indicating that the request is being processed - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and returned success - the cbfunc will not be called.

• PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the cbfunc will not be called.

• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called.

If executed, the status returned in the provided callback function will be one of the following constants:

• PMIX_SUCCESS The operation succeeded - i.e., the PMIX_GROUP_LEFT event was generated.
• **PMIX_ERR_NOT_SUPPORTED** While the PMIx library supports this operation, the host RM does not.

• a non-zero PMIx error constant indicating a reason for the request’s failure.

**Required Attributes**

There are no identified required attributes for implementers.

**Description**

Non-blocking version of the **PMIx_Group_leave** operation. The callback function will be called once the event has been locally generated and is not indicative of remote receipt.
CHAPTER 14

Fabric Support Definitions

As the drive for performance continues, interest has grown in scheduling algorithms that take into account network locality of the allocated resources and in optimizing collective communication patterns by structuring them to follow fabric topology. In addition, concerns over the time required to initiate execution of parallel applications and enable communication across them have grown as the size of those applications extends into the hundreds of thousands of individual processes spanning tens of thousands of nodes.

PMIx supports the communication part of these efforts by defining data types and attributes by which fabric endpoints and coordinates for processes and devices can be obtained from the host environment. When used in conjunction with other PMIx methods described in Chapter 16, this results in the ability of a process to obtain the fabric endpoint and coordinate of all other processes without incurring additional overhead associated with a global exchange of that information. This includes:

- Defining several interfaces specifically intended to support WLMs by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric.

- Supporting hierarchical collective operations by providing the fabric coordinates for all devices on participating nodes as well as a list of the peers sharing each fabric switch. This enables one, for example, to aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches based on detailed knowledge of the fabric location of each participant.

- Enabling the "instant on" paradigm to mitigate the scalable launch problem by providing each process with a rich set of information about the environment and the application, including everything required for communication between peers within the application, at time of process start of execution.

Meeting these needs in the case where only a single fabric device exists on each node is relatively straightforward - PMIx and the host environment provide a single endpoint for each process plus a coordinate for the device on each node, and there is no uncertainty regarding the endpoint each process will use. Extending this to the multiple device per node case is more difficult as the choice of endpoint by any given process cannot be known in advance, and questions arise regarding reachability between devices on different nodes. Resolving these ambiguities without requiring a global operation requires that PMIx provide both (a) an endpoint for each application process on each of its local devices; and (b) the fabric coordinates of all remote and local devices on participating nodes. It also requires that each process open all of its assigned endpoints as the endpoint selected for contact by a remote peer cannot be known in advance.
While these steps ensure the ability of a process to connect to a remote peer, it leaves unanswered the question of selecting the preferred device for that communication. If multiple devices are present on a node, then the application can benefit from having each process utilize its "closest" fabric device (i.e., the device that minimizes the communication distance between the process’ location and that device) for messaging operations. In some cases, messaging libraries prefer to also retain the ability to use non-nearest devices, prioritizing the devices based on distance to support multi-device operations (e.g., for large message transmission in parallel).

PMIx supports this requirement by providing the array of process-to-device distance information for each process and local fabric device at start of execution. Both minimum and maximum distances are provided since a single process can occupy multiple processor locations. In addition, since processes can relocate themselves by changing their processor bindings, PMIx provides an API that allows the process to dynamically request an update to its distance array.

However, while these measures assist a process in selecting its own best endpoint, they do not resolve the uncertainty over the choice of preferred device by a remote peer. There are two methods by which this ambiguity can be resolved:

a) A process can select a remote endpoint to use based on its own preferred device and reachability of the peer’s remote devices. Once the initial connection has been made, the two processes can exchange information and mutually determine their desired communication path going forward.

b) The application can use knowledge of both the local and remote distance arrays to compute the best communication path and establish that connection. In some instances (e.g., a homogeneous system), a PMIx server may provide distance information for both local and remote devices. Alternatively, when this isn’t available, an application can opt to collect the information using the `PMIX_COLLECT_GENERATED_JOB_INFO` with the `PMIx_Fence` API, or can obtain it on a one peer-at-a-time basis using the `PMIx_Get` API on systems where the host environment supports the Direct Modex operation.

Information on fabric coordinates, endpoints, and device distances are provided as reserved keys as detailed in Chapter 6 - i.e., they are to be available at client start of execution and are subject to the retrieval rules of Section 6.2. Examples for retrieving fabric-related information include retrieval of:

- An array of information on fabric devices for a node by passing `PMIX_FABRIC_DEVICES` as the key to `PMIx_Get` along with the `PMIX_HOSTNAME` of the node as a directive
- An array of information on a specific fabric device by passing `PMIX_FABRIC_DEVICE` as the key to `PMIx_Get` along with the `PMIX_DEVICE_ID` of the device as a directive
- An array of information on a specific fabric device by passing `PMIX_FABRIC_DEVICE` as the key to `PMIx_Get` along with both `PMIX_FABRIC_DEVICE_NAME` of the device and the `PMIX_HOSTNAME` of the node as directives

When requesting data on a device, returned data must include at least the following attributes:

- `PMIX_HOSTNAME "pmix.hname"` (char*)
Name of the host, as returned by the `gethostname` utility or its equivalent. The PMIX_NODEID may be returned in its place, or in addition to the hostname.

- **PMIX_DEVICE_ID** "pmix.dev.id" (string)
  System-wide UUID or node-local OS name of a particular device.

- **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 "en01") or an absolute filename.

- **PMIX_FABRIC_DEVICE_VENDOR** "pmix.fabdev.vnrd" (string)
  Indicates the name of the vendor that distributes the device.

- **PMIX_FABRIC_DEVICE_BUS_TYPE** "pmix.fabdev.btyp" (string)
  The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

- **PMIX_FABRIC_DEVICE_PCI_DEVID** "pmix.fabdev.pcidevid" (string)
  A node-level unique identifier for a Peripheral Component Interconnect (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 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The combination of node identifier (PMIX_HOSTNAME or PMIX_NODEID) and PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the overall system. This item should be included if the device bus type is PCI - the equivalent should be provided for any other bus type.

The returned array may optionally contain one or more of the following in addition to the above list:

- **PMIX_FABRIC_DEVICE_INDEX** "pmix.fabdev.idx" (uint32_t)
  Index of the device within an associated communication cost matrix.

- **PMIX_FABRIC_DEVICE_VENDORID** "pmix.fabdev.vendid" (string)
  This is a vendor-provided identifier for the device or product.

- **PMIX_FABRIC_DEVICE_DRIVER** "pmix.fabdev.driver" (string)
  The name of the driver associated with the device.

- **PMIX_FABRIC_DEVICE_FIRMWARE** "pmix.fabdev.fmwr" (string)
  The device’s firmware version.

- **PMIX_FABRIC_DEVICE_ADDRESS** "pmix.fabdev.addr" (string)
  The primary link-level address associated with the device, such as a Media Access Control (MAC) address. If multiple addresses are available, only one will be reported.

- **PMIX_FABRIC_DEVICE_COORDINATES** "pmix.fab.coord" (pmix_geometry_t)
  The `pmix_geometry_t` fabric coordinates for the device, including values for all supported coordinate views.
• **PMIX_FABRIC_DEVICE_MTU** "pmix.fabdev.mtu" (size_t)
  The maximum transfer unit of link level frames or packets, in bytes.

• **PMIX_FABRIC_DEVICE_SPEED** "pmix.fabdev.speed" (size_t)
  The active link data rate, given in bits per second.

• **PMIX_FABRIC_DEVICE_STATE** "pmix.fabdev.state" (pmix_link_state_t)
  The last available physical port state for the specified device. Possible values are
  PMIX_LINK_STATE_UNKNOWN, PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

• **PMIX_FABRICDEVICE_TYPE** "pmix.fabdev.type" (string)
  Specifies the type of fabric interface currently active on the device, such as Ethernet or InfiniBand.

The remainder of this chapter details the events, data types, attributes, and APIs associated with fabric-related operations.

## 14.1 Fabric Support Events

The following events are defined for use in fabric-related operations.

- **PMIX_FABRIC_UPDATE_PENDING** The PMIx server library has been alerted to a change in the fabric that requires updating of one or more registered pmix_fabric_t objects.
- **PMIX_FABRIC_UPDATED** The PMIx server library has completed updating the entries of all affected pmix_fabric_t objects registered with the library. Access to the entries of those objects may now resume.
- **PMIX_FABRIC_UPDATE_ENDPOINTS** Endpoint assignments have been updated, usually in response to migration or restart of a process. Clients should use PMIx_Get to update any internally cached connections.

## 14.2 Fabric Support Datatypes

Several datatype definitions have been created to support fabric-related operations and information.

### 14.2.1 Fabric Endpoint Structure

The pmix_endpoint_t structure contains an assigned endpoint for a given fabric device.
typedef struct pmix_endpoint {
    char *uuid;
    char *osname;
    pmix_byte_object_t endpt;
} pmix_endpoint_t;

The uuid field contains the UUID of the fabric device, the osname is the local operating system’s name for the device, and the endpt field contains a fabric vendor-specific object identifying the communication endpoint assigned to the process.

14.2.2 Fabric endpoint support macros

The following macros are provided to support the pmix_endpoint_t structure.

**Initialize the endpoint structure**
Initialize the pmix_endpoint_t fields.

**PMIx v4.0**

```
PMIX_ENDPOINT_CONSTRUCT(m)
```

**IN m**
Pointer to the structure to be initialized (pointer to pmix_endpoint_t)

**Destruct the endpoint structure**
Destruct the pmix_endpoint_t fields.

**PMIx v4.0**

```
PMIX_ENDPOINT_DESTRUCT(m)
```

**IN m**
Pointer to the structure to be destructed (pointer to pmix_endpoint_t)

**Create an endpoint array**
Allocate and initialize a pmix_endpoint_t array.

**PMIx v4.0**

```
PMIX_ENDPOINT_CREATE(m, n)
```

**INOUT m**
Address where the pointer to the array of pmix_endpoint_t structures shall be stored (handle)

**IN n**
Number of structures to be allocated (size_t)
Release an endpoint array

Release an array of `pmix_endpoint_t` structures.

```c
PMIX v4.0
PMIX_ENDPOINT_FREE(m, n)
```

**IN** `m`
- Pointer to the array of `pmix_endpoint_t` structures (handle)

**IN** `n`
- Number of structures in the array (`size_t`)

### 14.2.3 Fabric Coordinate Structure

The `pmix_coord_t` structure describes the fabric coordinates of a specified device in a given view.

```c
typedef struct pmix_coord {
    pmix_coord_view_t view;
    uint32_t *coord;
    size_t dims;
} pmix_coord_t;
```

All coordinate values shall be expressed as unsigned integers due to their units being defined in fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

**Advice to PMIx library implementers**

Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the coordinate data is to be stored within the PMIx library. Implementers are free to store the coordinate in whatever format they choose.

A fabric coordinate is associated with a given fabric device and must be unique within a given view. Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates are logically grouped within the `node` realm (as described in Section 6.1) and can be retrieved per the rules detailed in Section 6.1.5.

### 14.2.4 Fabric coordinate support macros

The following macros are provided to support the `pmix_coord_t` structure.
**Initialize the coord structure**
Initialize the `pmix_coord_t` fields.

```c
PMIx v4.0
PMIX_COORD_CONSTRUCT(m)
```

**IN m**
Pointer to the structure to be initialized (pointer to `pmix_coord_t`)

**Destruct the coord structure**
Destruct the `pmix_coord_t` fields.

```c
PMIx v4.0
PMIX_COORD_DESTRUCT(m)
```

**IN m**
Pointer to the structure to be destructed (pointer to `pmix_coord_t`)

**Create a coord array**
Allocate and initialize a `pmix_coord_t` array.

```c
PMIx v4.0
PMIX_COORD_CREATE(m, n)
```

**INOUT m**
Address where the pointer to the array of `pmix_coord_t` structures shall be stored (handle)

**IN n**
Number of structures to be allocated (`size_t`)

**Release a coord array**
Release an array of `pmix_coord_t` structures.

```c
PMIx v4.0
PMIX_COORD_FREE(m, n)
```

**IN m**
Pointer to the array of `pmix_coord_t` structures (handle)

**IN n**
Number of structures in the array (`size_t`)
14.2.5 Fabric Geometry Structure

The `pmix_geometry_t` structure describes the fabric coordinates of a specified device.

```
typedef struct pmix_geometry {
    size_t fabric;
    char *uuid;
    char *osname;
    pmix_coord_t *coordinates;
    size_t ncoords;
} pmix_geometry_t;
```

All coordinate values shall be expressed as unsigned integers due to their units being defined in fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity and not relative communication distance.

Advice to PMIx library implementers

Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the coordinate data is to be stored within the PMIx library. Implementers are free to store the coordinate in whatever format they choose.

A fabric coordinate is associated with a given fabric device and must be unique within a given view. Fabric devices are associated with the operating system which hosts them - thus, fabric coordinates are logically grouped within the node realm (as described in Section 6.1) and can be retrieved per the rules detailed in Section 6.1.5.

14.2.6 Fabric geometry support macros

The following macros are provided to support the `pmix_geometry_t` structure.

**Initialize the geometry structure**

Initialize the `pmix_geometry_t` fields.

```
PMIx v4.0

PMIX_GEOMETRY_CONSTRUCT(m)
```

`IN m`

Pointer to the structure to be initialized (pointer to `pmix_geometry_t`)
Destruct the geometry structure
Destruct the `pmix_geometry_t` fields.

```
PMIx v4.0
```
```
PMIX_GEOMETRY_DESTRUCT(m)
```

**IN** `m`
Pointer to the structure to be destructed (pointer to `pmix_geometry_t`)

Create a geometry array
Allocate and initialize a `pmix_geometry_t` array.

```
PMIx v4.0
```
```
PMIX_GEOMETRY_CREATE(m, n)
```

**INOUT** `m`
Address where the pointer to the array of `pmix_geometry_t` structures shall be stored (handle)

**IN** `n`
Number of structures to be allocated (`size_t`)

Release a geometry array
Release an array of `pmix_geometry_t` structures.

```
PMIx v4.0
```
```
PMIX_GEOMETRY_FREE(m, n)
```

**IN** `m`
Pointer to the array of `pmix_geometry_t` structures (handle)

**IN** `n`
Number of structures in the array (`size_t`)
14.2.7 Fabric Coordinate Views

Fabric coordinates can be reported based on different views according to user preference at the time of request. The following views have been defined:

- **PMIX_COORD_VIEW_UNDEF**  The coordinate view has not been defined.
- **PMIX_COORD_LOGICAL_VIEW**  The coordinates are provided in a logical view, typically given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains, and other similar factors employed by that fabric.
- **PMIX_COORD_PHYSICAL_VIEW**  The coordinates are provided in a physical view based on the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative position of that interface on the specific fabric cabling.

If the requester does not specify a view, coordinates shall default to the logical view.

14.2.8 Fabric Link State

The `pmix_link_state_t` is a `uint32_t` type for fabric link states.

The following constants can be used to set a variable of the type `pmix_link_state_t`. All definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state values start at zero.

- **PMIX_LINK_STATE_UNKNOWN**  The port state is unknown or not applicable.
- **PMIX_LINK_DOWN**  The port is inactive.
- **PMIX_LINK_UP**  The port is active.
14.2.9 Fabric Operation Constants

PMIx v4.0 The `pmix_fabric_operation_t` data type is an enumerated type for specifying fabric operations used in the PMIx server module’s `pmix_server_fabric_fn_t` API.

- **PMIX_FABRIC_REQUEST_INFO** Request information on a specific fabric - if the fabric isn’t specified as per `PMIx_Fabric_register`, then return information on the default fabric of the overall system. Information to be returned is described in `pmix_fabric_t`.
- **PMIX_FABRIC_UPDATE_INFO** Update information on a specific fabric - the index of the fabric (`PMIX_FABRIC_INDEX`) to be updated must be provided.

14.2.10 Fabric registration structure

PMIx v4.0 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.

```c
typedef struct pmix_fabric_s {
    char *name;
    size_t index;
    pmix_info_t *info;
    size_t ninfo;
    void *module;
} pmix_fabric_t;
```

Note that in this structure:

- **name** is an optional user-supplied string name identifying the fabric being referenced by this struct. If provided, the field must be a NULL-terminated string composed of standard alphanumeric values supported by common utilities such as `strcmp`;
- **index** is a PMIx-provided number identifying this object;
- **info** is an array of `pmix_info_t` containing information (provided by the PMIx library) about the fabric;
- **ninfo** is the number of elements in the `info` array;
- **module** points to an opaque object reserved for use by the PMIx server library.

Note that only the `name` field is provided by the user - all other fields are provided by the PMIx library and must not be modified by the user. The `info` 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

- **PMIX_FABRIC_VENDOR** "pmix.fab.vndr" (string)
  
  Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

- **PMIX_FABRIC_IDENTIFIER** "pmix.fab.id" (string)
  
  An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

- **PMIX_FABRIC_NUM_DEVICES** "pmix.fab.nverts" (size_t)
  
  Total number of fabric devices in the overall system - corresponds to the number of rows or columns in the cost matrix.

and may optionally contain one or more of the following:

Optional Attributes

- **PMIX_FABRIC_COST_MATRIX** "pmix.fab.cm" (pointer)
  
  Pointer to a two-dimensional square array of point-to-point relative communication costs expressed as \texttt{uint16\_t} values.

- **PMIX_FABRIC_GROUPS** "pmix.fab.grps" (string)
  
  A string delineating the group membership of nodes in the overall 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.,
  
  \texttt{0:node000,node002,node004,node006;1:node001,node003,node005,node007})

- **PMIX_FABRIC_DIMS** "pmix.fab.dims" (uint32\_t)
  
  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 overall system will be returned as a \texttt{pmix\_data\_array\_t} containing an array of \texttt{uint32\_t} values. Default is to provide dimensions in logical view.

- **PMIX_FABRIC_PLANE** "pmix.fab.plane" (string)
  
  ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly as a key in a request, returns a \texttt{pmix\_data\_array\_t} of string identifiers for all fabric planes in the overall system.

- **PMIX_FABRIC_SHAPE** "pmix.fab.shape" (pmix\_data\_array\_t*)
  
  The size of each dimension in the specified fabric plane/view, returned in a \texttt{pmix\_data\_array\_t} containing an array of \texttt{uint32\_t} values. The size is defined as the number of elements present in that dimension - e.g., the number of devices in one dimension of a physical view of a fabric plane. If no plane is specified, then the shape of each plane in the overall system will be returned in a \texttt{pmix\_data\_array\_t} array where each element is itself a two-element array containing the \texttt{PMIX_FABRIC_PLANE} followed by that plane’s fabric shape. Default is to provide the shape in logical view.
Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the shape of each plane in the overall system will be returned in a `pmix_data_array_t` array where each element is itself a two-element array containing the `PMIX_FABRIC_PLANE` followed by that plane’s fabric shape string. Default is to provide the shape in logical view.

While unusual due to scaling issues, implementations may include an array of `PMIX_FABRIC_DEVICE` elements describing the device information for each device in the overall system. Each element shall contain a `pmix_data_array_t` of `pmix_info_t` values describing the device. Each array may contain one or more of the following (ordering is arbitrary):

- **PMIX_FABRIC_DEVICE_NAME** "pmix.fabdev.nm" (string)
  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.

- **PMIX_FABRIC_DEVICE_VENDOR** "pmix.fabdev.vndr" (string)
  Indicates the name of the vendor that distributes the device.

- **PMIX_DEVICE_ID** "pmix.dev.id" (string)
  System-wide UUID or node-local OS name of a particular device.

- **PMIX_HOSTNAME** "pmix.hname" (char*)
  Name of the host, as returned by the `gethostname` utility or its equivalent.

- **PMIX_FABRIC_DEVICE_DRIVER** "pmix.fabdev.driver" (string)
  The name of the driver associated with the device.

- **PMIX_FABRIC_DEVICE_FIRMWARE** "pmix.fabdev.fmwr" (string)
  The device’s firmware version.

- **PMIX_FABRIC_DEVICE_ADDRESS** "pmix.fabdev.addr" (string)
  The primary link-level address associated with the device, such as a MAC address. If multiple addresses are available, only one will be reported.

- **PMIX_FABRIC_DEVICE_MTU** "pmix.fabdev.mtu" (size_t)
  The maximum transfer unit of link level frames or packets, in bytes.

- **PMIX_FABRIC_DEVICE_SPEED** "pmix.fabdev.speed" (size_t)
  The active link data rate, given in bits per second.

- **PMIX_FABRIC_DEVICE_STATE** "pmix.fabdev.state" (pmix_link_state_t)
  The last available physical port state for the specified device. Possible values are `PMIX_LINK_STATE_UNKNOWN`, `PMIX_LINK_DOWN`, and `PMIX_LINK_UP`, to indicate if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

- **PMIX_FABRIC_DEVICE_TYPE** "pmix.fabdev.type" (string)
  Specifies the type of fabric interface currently active on the device, such as Ethernet or InfiniBand.

- **PMIX_FABRIC_DEVICE_BUS_TYPE** "pmix.fabdev.btyp" (string)
The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

**PMIX_FABRIC_DEVICE_PCI_DEVID** "pmix.fabdev.pcidevid" (string)
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 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The combination of node identifier (**PMIX_HOSTNAME** or **PMIX_NODEID**) and **PMIX_FABRIC_DEVICE_PCI_DEVID** shall be unique within the overall system.

### 14.2.10.1 Initialize the fabric structure

Initialize the **pmix_fabric_t** fields.

```c
#include <pmix/pmix.h>

void PMIxadastro:0f23:01

/* Initialize the fabric structure */

void PMIxadastro:0f23:01

PMIx v4.0

PMIX_FABRIC_CONSTRUCT(m) IN m

Pointer to the structure to be initialized (pointer to **pmix_fabric_t**)
```

### 14.3 Fabric Support Attributes

The following attribute is used by the PMIx server library supporting the system’s WLM to indicate that it wants access to the fabric support functions:

**PMIX_SERVER_SCHEDULER** "pmix.srv.sched" (bool)
Server is supporting system scheduler and desires access to appropriate WLM-supporting features. Indicates that the library is to be initialized for scheduler support.

The following attributes may be returned in response to fabric-specific APIs or queries (e.g., **PMIx_Get** or **PMIx_Query_info**). These attributes are not related to a specific data realm (as described in Section 6.1) - the **PMIx_Get** function shall therefore ignore the value in its proc process identifier argument when retrieving these values:

**PMIX_FABRIC_COST_MATRIX** "pmix.fab.cm" (pointer)
Pointer to a two-dimensional square 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 overall 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)
```
**PMIX_FABRIC_PLANE** "pmix.fab.plane" (string)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly as a key in a request, returns a **pmix_data_array_t** of string identifiers for all fabric planes in the overall system.

**PMIX_FABRIC_SWITCH** "pmix.fab.switch" (string)

ID string of a fabric switch. When used as a modifier in a request for information, specifies the switch whose information is to be returned. When used directly as a key in a request, returns a **pmix_data_array_t** of string identifiers for all fabric switches in the overall system.

The following attributes may be returned in response to queries (e.g., **PMIx_Get** or **PMix_Query_info**). A qualifier (e.g., **PMIX_FABRIC_INDEX**) identifying the fabric whose value is being referenced must be provided for queries on systems supporting more than one fabric when values for the non-default fabric are requested. These attributes are not related to a specific data realm (as described in Section 6.1) - the PMIx_Get function shall therefore ignore the value in its proc process identifier argument when retrieving these values.

**PMIX_FABRIC_VENDOR** "pmix.fab.vndr" (string)

Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

**PMIX_FABRIC_IDENTIFIER** "pmix.fab.id" (string)

An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

**PMIX_FABRIC_INDEX** "pmix.fab.idx" (size_t)

The index of the fabric as returned in **pmix_fabric_t**.

**PMIX_FABRIC_NUM_DEVICES** "pmix.fab.nverts" (size_t)

Total number of fabric devices in the overall system - corresponds to the number of rows or columns in the cost matrix.

**PMIX_FABRIC_DIMS** "pmix.fab.dims" (uint32_t)

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 overall system will be returned as a **pmix_data_array_t** containing an array of uint32_t values. Default is to provide dimensions in logical view.

**PMIX_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 devices in one dimension of a physical view of a fabric plane. If no plane is specified, then the shape of each plane in the overall system will be returned in a **pmix_data_array_t** array where each element is itself a two-element array containing the **PMIX_FABRIC_PLANE** followed by that plane’s fabric shape. Default is to provide the shape in logical view.

**PMIX_FABRIC_SHAPE_STRING** "pmix.fab.shapestr" (string)

Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the shape of each plane in the overall system will be returned in a **pmix_data_array_t** array...
where each element is itself a two-element array containing the PMIX_FABRIC_PLANE followed by that plane’s fabric shape string. Default is to provide the shape in logical view.

The following attributes are related to the node realm (as described in Section 6.1.5) and are retrieved according to those rules.

**PMIX_FABRIC_DEVICES** "pmix.fab.devs" (pmix_data_array_t)
Array of pmix_info_t containing information for all devices on the specified node. Each element of the array will contain a PMIX_FABRIC_DEVICE entry, which in turn will contain an array of information on a given device.

**PMIX_FABRIC_COORDINATES** "pmix.fab.coords" (pmix_data_array_t)
Array of pmix_geometry_t fabric coordinates for devices on the specified node. The array will contain the coordinates of all devices on the node, including values for all supported coordinate views. The information for devices on the local node shall be provided if the node is not specified in the request.

**PMIX_FABRIC_DEVICE** "pmix.fabdev" (pmix_data_array_t)
An array of pmix_info_t describing a particular fabric device using one or more of the attributes defined below. The first element in the array shall be the PMIX_DEVICE_ID of the device.

**PMIX_FABRIC_DEVICE_INDEX** "pmix.fabdev.idx" (uint32_t)
Index of the device within an associated communication cost matrix.

**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.

**PMIX_FABRIC_DEVICE_VENDOR** "pmix.fabdev.vndr" (string)
Indicates the name of the vendor that distributes the device.

**PMIX_FABRIC_DEVICE_BUS_TYPE** "pmix.fabdev.btyp" (string)
The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

**PMIX_FABRIC_DEVICE_VENDORID** "pmix.fabdev.vendid" (string)
This is a vendor-provided identifier for the device or product.

**PMIX_FABRIC_DEVICE_DRIVER** "pmix.fabdev.driver" (string)
The name of the driver associated with the device.

**PMIX_FABRIC_DEVICE_FIRMWARE** "pmix.fabdev.fmwr" (string)
The device’s firmware version.

**PMIX_FABRIC_DEVICE_ADDRESS** "pmix.fabdev.addr" (string)
The primary link-level address associated with the device, such as a MAC address. If multiple addresses are available, only one will be reported.

**PMIX_FABRIC_DEVICE_COORDINATES** "pmix.fab.coord" (pmix_geometry_t)
The pmix_geometry_t fabric coordinates for the device, including values for all supported coordinate views.

**PMIX_FABRIC_DEVICE_MTU** "pmix.fabdev.mtu" (size_t)
The maximum transfer unit of link level frames or packets, in bytes.

**PMIX_FABRIC_DEVICE_SPEED** "pmix.fabdev.speed" (size_t)
The active link data rate, given in bits per second.

**PMIX_FABRIC_DEVICE_STATE** "pmix.fabdev.state" (pmix_link_state_t)
The last available physical port state for the specified device. Possible values are **PMIX_LINK_STATE_UNKNOWN**, **PMIX_LINK_DOWN**, and **PMIX_LINK_UP**, to indicate if the port state is unknown or not applicable (unknown), inactive (down), or active (up).

**PMIX_FABRIC_DEVICE_TYPE** "pmix.fabdev.type" (string)
Specifies the type of fabric interface currently active on the device, such as Ethernet or InfiniBand.

**PMIX_FABRIC_DEVICE_PCI_DEVID** "pmix.fabdev.pcidevid" (string)
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 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The combination of node identifier (**PMIX_HOSTNAME** or **PMIX_NODEID**) and **PMIX_FABRIC_DEVICE_PCI_DEVID** shall be unique within the overall system.

The following attributes are related to the **process realm** (as described in Section 6.1.4) and are retrieved according to those rules.

**PMIX_FABRIC_ENDPT** "pmix.fab.endpt" (pmix_data_array_t)
Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given process (e.g., in the case where multiple devices are associated with a package to which the process is bound), the returned values will be provided in a **pmix_data_array_t** of **pmix_endpoint_t** elements.

The following attributes are related to the **job realm** (as described in Section 6.1.2) and are retrieved according to those rules. Note that distances to fabric devices are retrieved using the **PMIX_DEVICE_DISTANCES** key with the appropriate **pmix_device_type_t** qualifier.

**PMIX_SWITCH_PEERS** "pmix.speers" (pmix_data_array_t)
Peer ranks that share the same switch as the process specified in the call to PMIx_Get. Returns a **pmix_data_array_t** array of **pmix_info_t** results, each element containing the **PMIX_SWITCH_PEERS** key with a three-element **pmix_data_array_t** array of **pmix_info_t** containing the **PMIX_DEVICE_ID** of the local fabric device, the **PMIX_FABRIC_SWITCH** identifying the switch to which it is connected, and a comma-delimited string of peer ranks sharing the switch to which that device is connected.

### 14.4 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 scheduler). The `PMIX_SERVER_SCHEDULER` 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 `pmix_server_fabric_fn_t` server module function, as needed.

14.4.1 PMIx_Fabric_register

Summary

Register for access to fabric-related information.

Format

```c
PMIx v4.0

pmix_status_t
PMIx_Fabric_register(pmix_fabric_t *fabric,
    const pmix_info_t directives[],
    size_t ndirs);
```

**INOUT fabric**

address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

**IN directives**

an optional array of values indicating desired behaviors and/or fabric to be accessed. If `NULL`, then the highest priority available fabric will be used (array of handles)

**IN ndirs**

Number of elements in the `directives` array (integer)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.
The following directives are required to be supported by all PMIx libraries to aid users in identifying the fabric whose data is being sought:

**PMIX_FABRIC_PLANE** "pmix.fab.plane" (string)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in the overall system.

**PMIX_FABRIC_IDENTIFIER** "pmix.fab.id" (string)

An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

**PMIX_FABRIC_VENDOR** "pmix.fab.vndr" (string)

Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

Description

Register for access to fabric-related information, including the communication cost matrix. This call must be made prior to requesting information from a fabric. The caller may request access to a particular fabric using the vendor, type, or identifier, or to a specific fabric plane via the `PMIX_FABRIC_PLANE` attribute - otherwise, information for the default fabric will be returned.

Upon successful completion of the call, information will have been filled into the fields of the provided fabric structure.

For performance reasons, the PMIx library does not provide thread protection for accessing the information in the `pmix_fabric_t` structure. Instead, the PMIx implementation shall provide two methods for coordinating updates to the provided fabric information:

- Users may periodically poll for updates using the `PMIx_Fabric_update` API
- Users may register for `PMIX_FABRIC_UPDATE_PENDING` events indicating that an update to the cost matrix is pending. When received, users are required to terminate or pause any actions involving access to the cost matrix before returning from the event. Completion of the `PMIX_FABRIC_UPDATE_PENDING` event handler indicates to the PMIx library that the fabric object’s entries are available for updating. This may include releasing and re-allocating memory as the number of vertices may have changed (e.g., due to addition or removal of one or more devices). When the update has been completed, the PMIx library will generate a `PMIX_FABRIC_UPDATED` event indicating that it is safe to begin using the updated fabric object(s).

There is no requirement that the caller exclusively use either one of these options. For example, the user may choose to both register for fabric update events, but poll for an update prior to some critical operation.
14.4.2 PMIx_Fabric_register_nb

Summary
Register for access to fabric-related information.

Format

PMIx v4.0

```c
pmix_status_t PMIx_Fabric_register_nb(pmix_fabric_t *fabric,
  const pmix_info_t directives[],
  size_t ndirs,
  pmix_op_cbfunc_t cbfunc, void *cbdata);
```

INOUT fabric
address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

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)

IN cbfunc
Callback function `pmix_op_cbfunc_t` (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.

- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

Description
Non-blocking form of `PMIx_Fabric_register`. The caller is not allowed to access the provided `pmix_fabric_t` until the callback function has been executed, at which time the fabric information will have been loaded into the provided structure.

14.4.3 PMIx_Fabric_update

Summary
Update fabric-related information.
PMIx v4.0

```c
pmix_status_t
PMIx_Fabric_update(pmix_fabric_t *fabric);
```

**INOUT** `fabric`

address of a `pmix_fabric_t` (backed by storage) (handle)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant.

**Description**

Update fabric-related information. This call can be made at any time to request an update of the fabric information contained in the provided `pmix_fabric_t` object. The caller is not allowed to access the provided `pmix_fabric_t` until the call has returned. Upon successful return, the information fields in the `fabric` structure will have been updated.

**14.4.4 PMIx_Fabric_update_nb**

**Summary**

Update fabric-related information.

```c
PMIx v4.0

pmix_status_t
PMIx_Fabric_update_nb(pmix_fabric_t *fabric,
                     pmix_op_cbfunc_t cbfunc, void *cbdata);
```

**INOUT** `fabric`

address of a `pmix_fabric_t` (handle)

**IN** `cbfunc`

Callback function `pmix_op_cbfunc_t` (function reference)

**IN** `cbdata`

Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.
- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed.
Non-blocking form of `PMIx_Fabric_update`. The caller is not allowed to access the provided `pmix_fabric_t` until the callback function has been executed, at which time the fields in the provided `fabric` structure will have been updated.

### 14.4.5 PMIx_Fabric_deregister

**Summary**
Deregister a fabric object.

**Format**

```c
pmix_status_t
PMIx_Fabric_deregister(pmix_fabric_t *fabric);
```

**IN fabric**
address of a `pmix_fabric_t` (handle)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

**Description**
Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information (e.g., cost matrix) associated with it. Contents of the provided `pmix_fabric_t` will be invalidated upon function return.

### 14.4.6 PMIx_Fabric_deregister_nb

**Summary**
Deregister a fabric object.

**Format**

```c
```

PMIx v4.0

CHAPTER 14. FABRIC SUPPORT DEFINITIONS 277
pmix_status_t PMIx_Fabric_deregister_nb(pmix_fabric_t *fabric,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);

IN   fabric
    address of a pmix_fabric_t (handle)

IN   cbfunc
    Callback function pmix_op_cbfunc_t (function reference)

IN   cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided callback function will be executed upon completion of the operation. Note that the library must not invoke the callback function prior to returning from the API.

- a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this case, the provided callback function will not be executed

**Description**
Non-blocking form of PMIx_Fabric_deregister. Provided fabric must not be accessed until after callback function has been executed.
CHAPTER 15

Security

PMIx utilizes a multi-layered approach toward security that differs for client versus tool processes. By definition, client processes must be preregistered with the PMIx server library via the `PMIx_server_register_client` API before they are spawned. This API requires that the host pass the expected effective UID/GID of the client process.

When the client attempts to connect to the PMIx server, the server shall use available standard 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. The effective UID/GID reported by the OS is compared to the values provided by the host during registration - if the values fail to 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_connected2_fn_t` interface, if provided. The host may 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 of the tool and report them upwards as part of invoking the `pmix_server_tool_connection_fn_t` interface, deferring initial security screening to the host. Host environments willing to accept tool connections must therefore both explicitly enable them via the `PMIX_SERVER_TOOL_SUPPORT` attribute, thereby confirming acceptance of the authentication and authorization burden, and provide the `pmix_server_tool_connection_fn_t` server module function pointer.

15.1 Obtaining Credentials

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.

### 15.1.1 PMIx_Get_credential

**Summary**

Request a credential from the PMIx server library or the host environment.

**Format**

```
PMIx v3.0

pmix_status_t
PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,
                    pmix_byte_object_t *credential);
```

**IN info**

Array of `pmix_info_t` structures (array of handles)

**IN ninfo**

Number of elements in the `info` array (`size_t`)

**IN credential**

Address of a `pmix_byte_object_t` within which to return credential (handle)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the credential has been returned in the provided `pmix_byte_object_t`
- a PMIx error constant indicating either an error in the input or that the request is unsupported

**Required Attributes**

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client’s request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the `info` array passed from the PMIx library to the host environment:

- **PMIX_USERID** "pmix.euid" (`uint32_t`)
  
  Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (`uint32_t`)
  
  Effective group ID of the connecting process.
The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

  Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**

Request a credential from the PMIx server library or the host environment. The credential is returned as a **pmix_byte_object_t** to support potential binary formats - it is therefore opaque to the caller. No information as to the source of the credential is provided.

**15.1.2 PMIx_Get_credential_nb**

**Summary**

Request a credential from the PMIx server library or the host environment.

**Format**

```
PMIx v3.0
```

```
pmix_status_t
PMIx_Get_credential_nb(const pmix_info_t info[], size_t ninfo,  
                          pmix_credential_cbfunc_t cbfunc,  
                          void *cbdata);
```

**IN**  
info
  Array of **pmix_info_t** structures (array of handles)

**IN**  
ninfo
  Number of elements in the **info** array (**size_t**)

**IN**  
cbfunc
  Callback function to return credential (**pmix_credential_cbfunc_t** function reference)

**IN**  
cbdata
  Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided **cbfunc**

- a PMIx error constant indicating either an error in the input or that the request is unsupported - the **cbfunc** will *not* be called
There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a munge server).

Implementations that support the operation but cannot directly process the client’s request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the info array passed from the PMIx library to the host environment:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  
  Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (uint32_t)
  
  Effective group ID of the connecting process.

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  
  Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**

Request a credential from the PMIx server library or the host environment. This version of the API is generally preferred in scenarios where the host environment may have to contact a remote credential service. 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.

**15.1.3 Credential Attributes**

The following attributes are defined to support credential operations:

- **PMIX_CRED_TYPE** "pmix.sec.ctype" (char*)
  
  When passed in PMIx_Get_credential, a prioritized, comma-delimited list of desired credential types for use in environments where multiple authentication mechanisms may be available. When returned in a callback function, a string identifier of the credential type.

- **PMIX_CRYPTO_KEY** "pmix.sec.key" (pmix_byte_object_t)
  
  Blob containing crypto key.
15.2 Validating Credentials

Given a credential, PMIx provides two methods by which a caller can request that the system validate it, returning any additional information (e.g., authorizations) conveyed within the credential.

15.2.1 PMIx_Validate_credential

Summary

Request validation of a credential by the PMIx server library or the host environment.

Format

```
PMIx v3.0

pmix_status_t
PMIx_Validate_credential(const pmix_byte_object_t *cred,
                         const pmix_info_t info[], size_t ninfo,
                         pmix_info_t **results, size_t *nresults);
```

IN cred
Pointer to `pmix_byte_object_t` containing the credential (handle)

IN info
Array of `pmix_info_t` structures (array of handles)

IN ninfo
Number of elements in the `info` array (`size_t`)

INOUT results
Address where a pointer to an array of `pmix_info_t` containing the results of the request can be returned (memory reference)

INOUT nresults
Address where the number of elements in `results` can be returned (handle)

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request was processed and returned `success` (i.e., the credential was both valid and any information it contained was successfully processed). Details of the result will be returned in the `results` array
- a PMIx error constant indicating either an error in the parsing of the credential or that the request was refused
There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a munge server).

Implementations that support the operation but cannot directly process the client’s request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the info array passed from the PMIx library to the host environment:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  - Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (uint32_t)
  - Effective group ID of the connecting process.

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  - Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**

Request validation of a credential by the PMIx server library or the host environment.

**15.2.2 PMIx.Validate_credential_nb**

**Summary**

Request validation of a credential by the PMIx server library or the host environment. Provision is made for the system to return additional information regarding possible authorization limitations beyond simple authentication.
pmix_status_t
PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
const pmix_info_t info[], size_t ninfo,
pmix_validation_cbfunc_t cbfunc,
void *cbdata);

IN  cred
  Pointer to pmix_byte_object_t containing the credential (handle)
IN  info
  Array of pmix_info_t structures (array of handles)
IN  ninfo
  Number of elements in the info array (size_t)
IN  cbfunc
  Callback function to return result (pmix_validation_cbfunc_t function reference)
IN  cbdata
  Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided cbfunc
- a PMIx error constant indicating either an error in the input or that the request is unsupported - the cbfunc will not be called

Upon completion of processing the callback function will be executed. Note that the callback function must not be executed prior to return from the API.

**Required Attributes**

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a munge server).

Implementations that support the operation but cannot directly process the client’s request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the info array passed from the PMIx library to the host environment:

**PMIX_USERID** "pmix.euid" (uint32_t)
  Effective user ID of the connecting process.

**PMIX_GRPID** "pmix.egid" (uint32_t)
  Effective group ID of the connecting process.
Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)
Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description
Request validation of a credential by the PMIx server library or the host environment. This version of the API is generally preferred in scenarios where the host environment may have to contact a remote credential service. Provision is made for the system to return additional information (e.g., possible authorization limitations) beyond simple authentication.
The process that hosts the PMIx server library interacts with that library in two distinct manners. First, PMIx provides a set of APIs by which the host can request specific services from its library. This includes:

- collecting inventory to support scheduling algorithms,
- providing subsystems with an opportunity to precondition their resources for optimized application support,
- generating regular expressions,
- registering information to be passed to client processes, and
- requesting information on behalf of a remote process.

Note that the host always has access to all PMIx client APIs - the functions listed below are in addition to those available to a PMIx client.

Second, the host can provide a set of callback functions by which the PMIx server library can pass requests upward for servicing by the host. These include notifications of client connection and finalize, as well as requests by clients for information and/or services that the PMIx server library does not itself provide.

### 16.1 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers.

#### 16.1.1 PMIx_server_init

**Summary**

Initialize the PMIx server.

**Format**

```c
pmix_status_t
PMIx_server_init(pmix_server_module_t *module,
                 pmix_info_t info[], size_t ninfo);
```

---

287
INOUT module

    pmix_server_module_t structure (handle)

IN info

    Array of pmix_info_t structures (array of handles)

IN ninfo

    Number of elements in the info array (size_t)

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)

    Name of the namespace to use for this PMIx server.

PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)

    Rank of this PMIx server.

PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)

    Top-level temporary directory for all client processes connected to this server, and where the
    PMIx server will place its tool rendezvous point and contact information.

PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)

    Temporary directory for this system, and where a PMIx server that declares itself to be a
    system-level server will place a tool rendezvous point and contact information.

PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)

    The host RM wants to declare itself as willing to accept tool connection requests.

PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)

    The host RM wants to declare itself as being the local system server for PMIx connection
    requests.

PMIX_SERVER_SESSION_SUPPORT "pmix.srvr.sess" (bool)

    The host RM wants to declare itself as being the local session server for PMIx connection
    requests.

PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)

    Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
    (e.g., logging to email).

PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)

    Server is supporting system scheduler and desires access to appropriate WLM-supporting
    features. Indicates that the library is to be initialized for scheduler support.
Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

- **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)
  Disable legacy UNIX socket (usock) support. If the library supports Unix socket connections, this attribute may be supported for disabling it.

- **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)
  POSIX mode_t (9 bits valid). If the library supports socket connections, this attribute may be supported for setting the socket mode.

- **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)
  Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

- **PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)
  If provided, directs that the TCP URI be reported and indicates the desired method of reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

- **PMIX_TCP_IF_INCLUDE** "pmix.tcp.ifinclude" (char*)
  Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

- **PMIX_TCP_IF_EXCLUDE** "pmix.tcp.ifexclude" (char*)
  Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are not to be used.

- **PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)
  The IPv4 port to be used. If the library supports IPv4 connections, this attribute may be supported for specifying the port to be used.

- **PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)
  The IPv6 port to be used. If the library supports IPv6 connections, this attribute may be supported for specifying the port to be used.

- **PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)
  Set to true to disable IPv4 family of addresses. If the library supports IPv4 connections, this attribute may be supported for disabling it.

- **PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)
  Set to true to disable IPv6 family of addresses. If the library supports IPv6 connections, this attribute may be supported for disabling it.

- **PMIX_SERVER_REMOTE_CONNECTIONS** "pmix.srvr.remote" (bool)
Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device. If the library supports connections from remote tools, this attribute may be supported for enabling or disabling it.

**PMIX_EXTERNAL_PROGRESS** "pmix.evext" (bool)
The host shall progress the PMIx library via calls to **PMIx_Progress**

**PMIX_EVENT_BASE** "pmix.evbase" (void*)
Pointer to an **event_base** to use in place of the internal progress thread. All PMIx library events are to be assigned to the provided event base. The event base must be compatible with the event library used by the PMIx implementation - e.g., either both the host and PMIx library must use libevent, or both must use libev. Cross-matches are unlikely to work and should be avoided - it is the responsibility of the host to ensure that the PMIx implementation supports (and was built with) the appropriate event library.

**PMIX_TOPOLOGY2** "pmix.topo2" (pmix_topology_t)
Provide a pointer to an implementation-specific description of the local node topology.

**PMIX_SERVER_SHARE_TOPOLOGY** "pmix.srvr.share" (bool)
The PMIx server is to share its copy of the local node topology (whether given to it or self-discovered) with any clients. The PMIx server will perform the necessary actions to scalably expose the description to the local clients. This includes creating any required shared memory backing stores and/or XML representations, plus ensuring that all necessary key-value pairs for clients to access the description are included in the job-level information provided to each client. All required files are to be installed under the effective **PMIX_SERVER_TMPDIR** directory. The PMIx server library is responsible for cleaning up any artifacts (e.g., shared memory backing files or cached key-value pairs) at library finalize.

**PMIX_SERVER_ENABLE_MONITORING** "pmix.srv.monitor" (bool)
Enable PMIx internal monitoring by the PMIx server.

**PMIX_HOMOGENEOUS_SYSTEM** "pmix.homo" (bool)
The nodes comprising the session are homogeneous - i.e., they each contain the same number of identical packages, fabric interfaces, GPUs, and other devices.

---

**Description**
Initialize the PMIx server support library, and provide a pointer to a **pmix_server_module_t** structure containing the caller’s callback functions. The array of **pmix_info_t** structs is used to pass additional info that may be required by the server when initializing. For example, it may include the **PMIX_SERVER_TOOL_SUPPORT** attribute, thereby indicating that the daemon is willing to accept connection requests from tools.
Advice to PMIx server hosts

Providing a value of NULL for the module argument is permitted, as is passing an empty module structure. Doing so indicates that the host environment will not provide support for multi-node operations such as PMIx_Fence, but does intend to support local clients access to information.

16.1.2 PMIx_server_finalize

Summary
Finalize the PMIx server library.

Format

PMIx v1.0

C

pmix_status_t
PMIx_server_finalize(void);

C

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

Description
Finalize the PMIx server support library, terminating all connections to attached tools and any local clients. All memory usage is released.

16.1.3 Server Initialization Attributes

These attributes are used to direct the configuration and operation of the PMIx server library by passing them into PMIx_server_init.

PMIX_TOPOLOGY2 "pmix.topo2" (pmix_topology_t)
Provide a pointer to an implementation-specific description of the local node topology.

PMIX_SERVER_SHARE_TOPOLOGY "pmix.srvr.share" (bool)
The PMIx server is to share its copy of the local node topology (whether given to it or self-discovered) with any clients.

PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)
Disable legacy UNIX socket (usock) support.

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)
POSIX mode_t (9 bits valid).

PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)
Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport.

PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (bool)
The host RM wants to declare itself as willing to accept tool connection requests.

PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)
Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device.

PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (bool)
The host RM wants to declare itself as being the local system server for PMIx connection requests.

PMIX_SERVER_SESSION_SUPPORT "pmix.srvr.sess" (bool)
The host RM wants to declare itself as being the local session server for PMIx connection requests.

PMIX_SERVER_START_TIME "pmix.srvr.strtime" (char*)
Time when the server started - i.e., when the server created it’s rendezvous file (given in ctime string format).

PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (char*)
Top-level temporary directory for all client processes connected to this server, and where the PMIx server will place its tool rendezvous point and contact information.

PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (char*)
Temporary directory for this system, and where a PMIx server that declares itself to be a system-level server will place a tool rendezvous point and contact information.

PMIX_SERVER_ENABLE_MONITORING "pmix.srv.monitor" (bool)
Enable PMIx internal monitoring by the PMIx server.

PMIX_SERVER_NSPACE "pmix.srv.nspace" (char*)
Name of the namespace to use for this PMIx server.

PMIX_SERVER_RANK "pmix.srv.rank" (pmix_rank_t)
Rank of this PMIx server.

PMIX_SERVER_GATEWAY "pmix.srv.gway" (bool)
Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes (e.g., logging to email).

PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
Server is supporting system scheduler and desires access to appropriate WLM-supporting features. Indicates that the library is to be initialized for scheduler support.

PMIX_EXTERNAL_PROGRESS "pmix.evext" (bool)
The host shall progress the PMIx library via calls to PMIx_Progress

PMIX_HOMOGENEOUS_SYSTEM "pmix.homo" (bool)
The nodes comprising the session are homogeneous - i.e., they each contain the same number of identical packages, fabric interfaces, GPUs, and other devices.

16.2 Server Support Functions

The following APIs allow the RM daemon that hosts the PMIx server library to request specific services from the PMIx library.

16.2.1 PMIx_generate_regex

Summary
Generate a compressed representation of the input string.
Format

PMIx v1.0

pmix_status_t

PMIx_generate_regex(const char *input, char **output);

IN  input
    String to process (string)

OUT output
    Compressed representation of input (array of bytes)

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

Description

Given a comma-separated list of input values, generate a reduced size representation of the input
that can be passed down to the PMIx server library’s PMIx_server_register_nspace API
for parsing. The order of the individual values in the input string is preserved across the operation.
The caller is responsible for releasing the returned data.

The precise compressed representations will be implementation specific. The regular expression
itself is not required to be a printable string nor to obey typical string constraints (e.g., include a
NULL terminator byte). However, all PMIx implementations are required to include a
colon-delimited NULL-terminated string at the beginning of the output representation that can be
printed for diagnostic purposes and identifies the method used to generate the representation. The
following identifiers are reserved by the PMIx Standard:

• "raw:\0" - indicates that the expression following the identifier is simply the
  comma-delimited input string (no processing was performed).

• "pmix:\0" - a PMIx-unique regular expression represented as a NULL-terminated string
  following the identifier.

• "blob:\0" - a PMIx-unique regular expression that is not represented as a NULL-terminated
  string following the identifier. Additional implementation-specific metadata may follow the
  identifier along with the data itself. For example, a compressed binary array format based on the
  zlib compression package, with the size encoded in the space immediately following the
  identifier.

Communicating the resulting output should be done by first packing the returned expression using
the PMIx_Data_pack, declaring the input to be of type PMIX_REGEX, and then obtaining the
resulting blob to be communicated using the PMIX_DATA_BUFFER_UNLOAD macro. The
reciprocal method can be used on the remote end prior to passing the regex into
PMIx_server_register_nspace. The pack/unpack routines will ensure proper handling of
the data based on the regex prefix.
16.2.2 PMIx_generate_ppn

**Summary**
Generate a compressed representation of the input identifying the processes on each node.

**Format**

```c
pmix_status_t
PMIx_generate_ppn(const char *input, char **ppn);
```

**IN**  
String to process (string)

**OUT**  
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.

Rules governing the format of the returned regular expression are the same as those specified for **PMIx_generate_regex**, as detailed here.

16.2.3 PMIx_server_register_nspace

**Summary**
Setup the data about a particular namespace.

**Format**

```c
pmix_status_t
PMIx_server_register_nspace(const pmix_nspace_t nspace, 
int nlocalprocs,  
const pmix_info_t info[], size_t ninfo, 
const pmix_op_cbfunc_t cbfunc, 
void *cbdata);
```
C

IN nspace
Character array of maximum size PMIX_MAX_NSLEN containing the namespace identifier (string)

IN nlocalprocs
number of local processes (integer)

IN info
Array of info structures (array of handles)

IN ninfo
Number of elements in the info array (integer)

IN cbfunc
Callback function pmix_op_cbfunc_t to be executed upon completion of the operation.
A NULL function reference indicates that the function is to be executed as a blocking operation (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- a PMix error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

The following attributes are required to be supported by all PMix libraries:

**PMIX_REGISTER_NODATA** "pmix.reg.nodata" (bool)
Registration is for this namespace only, do not copy job data.

**PMIX_SESSION_INFO_ARRAY** "pmix.ssn.arr" (pmix_data_array_t)
Provide an array of pmix_info_t containing session-realm information. The PMIX_SESSION_ID attribute is required to be included in the array.

**PMIX_JOB_INFO_ARRAY** "pmix.job.arr" (pmix_data_array_t)
Provide an array of pmix_info_t containing job-realm information. The PMIX_SESSION_ID attribute of the session containing the job is required to be included in the array whenever the PMix server library may host multiple sessions (e.g., when executing with a host RM daemon). As information is registered one job (aka namespace) at a time via the PMix_server_register_nspace API, there is no requirement that the array contain either the PMIX_NSPACE or PMIX_JOBID attributes when used in that context.
though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is ambiguous.

**PMIX_APP_INFO_ARRAY** "pmix.app.arr" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing application-realm information. The `PMIX_NSPACE` or `PMIX_JOBID` attributes of the job containing the application, plus its `PMIX_APPNUM` attribute, must to be included in the array when the array is not included as part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

**PMIX_PROC_INFO_ARRAY** "pmix.pdata" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing process-realm information. The `PMIX_RANK` and `PMIX_NSPACE` attributes, or the `PMIX_PROCID` attribute, are required to be included in the array when the array is not included as part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the process is ambiguous. All three may be included if desired. When the array is included in some broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID` attribute must be included (the others are optional).

**PMIX_NODE_INFO_ARRAY** "pmix.node.arr" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing node-realm information. At a minimum, either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the array, though both may be included.

Host environments are required to provide a wide range of session-, job-, application-, node-, and process-realm information, and may choose to provide a similarly wide range of optional information. The information is broadly separated into categories based on the *data realm* definitions explained in Section 6.1, and retrieved according to the rules detailed in Section 6.2.

Session-realm information may be passed as individual `pmix_info_t` entries, or as part of a `pmix_data_array_t` using the `PMIX_SESSION_INFO_ARRAY` attribute. The list of data referenced in this way shall include:

- **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)
  Maximum number of process that can be simultaneously executing in a session. Note that this attribute is equivalent to the `PMIX_MAX_PROCS` attribute for the *session* realm - it is included in the PMIx Standard for historical reasons.

- **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
  Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Defaults to the *job* realm. Must be provided if `PMIX_UNIV_SIZE` is not given. Requires use of the `PMIX_SESSION_INFO` attribute to avoid ambiguity when retrieving it.

- **PMIX_SESSION_ID** "pmix.session.id" (uint32_t)
Session identifier assigned by the scheduler.

plus the following optional information:

- **PMIX_CLUSTER_ID** "pmix.clid" (char*)
  A string name for the cluster this allocation is on. As this information is not related to the namespace, it is best passed using the `PMIx_server_register_resources` API.

- **PMIX_ALLOCATED_NODELIST** "pmix.alist" (char*)
  Comma-delimited list or regular expression of all nodes in the specified realm regardless of whether or not they currently host processes. Defaults to the `job` realm.

- **PMIX_RM_NAME** "pmix.rm.name" (char*)
  String name of the RM. As this information is not related to the namespace, it is best passed using the `PMIx_server_register_resources` API.

- **PMIX_RM_VERSION** "pmix.rm.version" (char*)
  RM version string. As this information is not related to the namespace, it is best passed using the `PMIx_server_register_resources` API.

- **PMIX_SERVER_HOSTNAME** "pmix.srvr.host" (char*)
  Host where target PMIx server is located. As this information is not related to the namespace, it is best passed using the `PMIx_server_register_resources` API.

Job-realm information may be passed as individual `pmix_info_t` entries, or as part of a `pmix_data_array_t` using the `PMIX_JOB_INFO_ARRAY` attribute. The list of data referenced in this way shall include:

- **PMIX_SERVER_NSPACE** "pmix.srv.nspace" (char*)
  Name of the namespace to use for this PMIx server. Identifies the namespace of the PMIx server itself.

- **PMIX_SERVER_RANK** "pmix.srv.rank" (pmix_rank_t)
  Rank of this PMIx server. Identifies the rank of the PMIx server itself.

- **PMIX_NSNAME** "pmix.nspace" (char*)
  Namespace of the job - may be a numerical value expressed as a string, but is often an alphanumeric string carrying information solely of use to the system. Required to be unique within the scope of the host environment. Identifies the namespace of the job being registered.

- **PMIX_JOBID** "pmix.jobid" (char*)
  Job identifier assigned by the scheduler to the specified job - may be identical to the namespace, but is often a numerical value expressed as a string (e.g., "12345.3").

- **PMIX_JOB_SIZE** "pmix.job.size" (uint32_t)
  Total number of processes in the specified job across all contained applications. Note that this value can be different from `PMIX_MAX_PROCS`. For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic
programming models may support adding and removing processes from a running job on-the-fly. In the latter case, PMIx events may be used to notify processes within the job that the job size has changed.

- **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
  Maximum number of processes that can be executed in the specified realm. Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Defaults to the job realm. Retrieval of this attribute defaults to the job level unless an appropriate specification is given (e.g., PMIX_SESSION_INFO).

- **PMIX_NODE_MAP** "pmix.nmap" (char*)
  Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the job realm.

- **PMIX_PROC_MAP** "pmix.pmap" (char*)
  Regular expression describing processes on each node in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the job realm.

plus the following optional information:

- **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
  Starting global rank of the specified job.

- **PMIX_JOB_NUM_APPS** "pmix.job.napps" (uint32_t)
  Number of applications in the specified job. This is a required attribute if more than one application is included in the job.

- **PMIX_MAPBY** "pmix.mapby" (char*)
  Process mapping policy - when accessed using PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace. Supported values are launcher specific.

- **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. Supported values are launcher specific.

- **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. Supported values are launcher specific.

- **PMIX_HOSTNAME_KEEP_FQDN** "pmix.fqdn" (bool)
  FQDNs are being retained by the PMIx library.

- **PMIX_ANL_MAP** "pmix.anlmap" (char*)
  Process map equivalent to PMIX_PROC_MAP expressed in Argonne National Laboratory’s PMI-1/PMI-2 notation. Defaults to the job realm.
• **PMIX_TDIR_RMCLEAN** "pmix.tdir.rmclean" (bool)
  
  Resource Manager will cleanup assigned temporary directory trees.

• **PMIX_CRYPTO_KEY** "pmix.sec.key" (pmix_byte_object_t)
  
  Blob containing crypto key.

If more than one application is included in the namespace, then the host environment is also
required to supply data consisting of the following items for each application in the job, passed as a
`pmix_data_array_t` using the `PMIX_APP_INFO_ARRAY` attribute:

• **PMIX_APPNUM** "pmix.appnum" (uint32_t)
  
  The application number within the job in which the specified process is a member. This
  attribute must appear at the beginning of the array.

• **PMIX_APP_SIZE** "pmix.app.size" (uint32_t)
  
  Number of processes in the specified application, regardless of their execution state - i.e.,
  this number may include processes that either failed to start or have already terminated.

• **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
  
  Maximum number of processes that can be executed in the specified realm. Typically, this
  is a constraint imposed by a scheduler or by user settings in a hostfile or other resource
description. Defaults to the job realm. Requires use of the `PMIX_APP_INFO` attribute
to avoid ambiguity when retrieving it.

• **PMIX_APPLDR** "pmix.aldr" (pmix_rank_t)
  
  Lowest rank in the specified application.

• **PMIX_WDIR** "pmix.wdir" (char*)
  
  Working directory for spawned processes. This attribute is required for all registrations,
  but may be provided as an individual `pmix_info_t` entry if only one application is
  included in the namespace.

• **PMIX_APP_ARGV** "pmix.app.argv" (char*)
  
  Consolidated argv passed to the spawn command for the given application (e.g., "/myapp
  arg1 arg2 arg3"). This attribute is required for all registrations, but may be provided as an
  individual `pmix_info_t` entry if only one application is included in the namespace.

plus the following optional information:

• **PMIX_PSET_NAMES** "pmix.pset.nms" (pmix_data_array_t*)
  
  Returns an array of `char*` string names of the process sets in which the given process is
  a member.

• **PMIX_APP_MAP_TYPE** "pmix.apmap.type" (char*)
  
  Type of mapping used to layout the application (e.g., `cyclic`). This attribute may be
  provided as an individual `pmix_info_t` entry if only one application is included in the
  namespace.

• **PMIX_APP_MAP_REGEX** "pmix.apmap.regex" (char*)
Regular expression describing the result of the process mapping. This attribute may be provided as an individual `pmix_info_t` entry if only one application is included in the namespace.

The data may also include attributes provided by the host environment that identify the programming model (as specified by the user) being executed within the application. 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”).

Node-realm information may be passed as individual `pmix_info_t` entries if only one node will host processes from the job being registered, or as part of a `pmix_data_array_t` using the `PMIX_NODE_INFO_ARRAY` attribute when multiple nodes are involved in the job. The list of data referenced in this way shall include:

- `PMIX_NODEID "pmix.nodeid" (uint32_t)`
  Node identifier expressed as the node’s index (beginning at zero) in an array of nodes within the active session. The value must be unique and directly correlate to the `PMIX_HOSTNAME` of the node - i.e., users can interchangeably reference the same location using either the `PMIX_HOSTNAME` or corresponding `PMIX_NODEID`.

- `PMIX_HOSTNAME "pmix.hname" (char*)`
  Name of the host, as returned by the `gethostname` utility or its equivalent. As this information is not related to the namespace, it can be passed using the `PMIx_server_register_resources` API. However, either it or the `PMIX_NODEID` must be included in the array to properly identify the node.

- `PMIX_HOSTNAME_ALIASES "pmix.alias" (char*)`
  Comma-delimited list of names by which the target node is known. As this information is not related to the namespace, it is best passed using the `PMIx_server_register_resources` API.

- `PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)`
  Number of processes in the specified job or application realm on the caller’s node. Defaults to job realm unless the `PMIX_APP_INFO` and the `PMIX_APPNUM` qualifiers are given.

- `PMIX_NODE_SIZE "pmix.node.size" (uint32_t)`
  Number of processes across all jobs that are executing upon the node.

- `PMIX_LOCAL_LDR "pmix.lldr" (pmix_rank_t)`
Lowest rank within the specified job on the node (defaults to current node in absence of
PMIX_HOSTNAME or PMIX_NODEID qualifier).

- **PMIX_LOCAL_PEERS** "pmix.lpeers" (char*)
  Comma-delimited list of ranks that are executing on the local node within the specified
  namespace – shortcut for PMIx.Resolve_peers for the local node.

plus the following information for the server’s own node:

- **PMIX_TMPDIR** "pmix.tmpdir" (char*)
  Full path to the top-level temporary directory assigned to the session.

- **PMIX_NS_DIR** "pmix.nsdir" (char*)
  Full path to the temporary directory assigned to the specified job, under PMIX_TMPDIR.

- **PMIX_LOCAL_PROCS** "pmix.lprocs" (pmix_proc_t array)
  Array of pmix_proc_t of all processes executing on the local node – shortcut for
  PMIx.Resolve_peers for the local node and a NULL namespace argument. The
  process identifier is ignored for this attribute.

The data may also include the following optional information for the server’s own node:

- **PMIX_LOCAL_CPUSETS** "pmix.lcpus" (pmix_data_array_t)
  A pmix_data_array_t array of string representations of the PU binding bitmaps
  applied to each local peer on the caller’s node upon launch. Each string shall begin with
  the name of the library that generated it (e.g., “hwloc”) followed by a colon and the bitmap
  string itself. The array shall be in the same order as the processes returned by
  PMIX_LOCAL_PEERS for that namespace.

- **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)
  Total available physical memory on a node. As this information is not related to the
  namespace, it can be passed using the PMIx_server_register_resources API.

and the following optional information for other nodes:

- **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
  Maximum number of processes that can be executed in the specified realm. Typically, this
  is a constraint imposed by a scheduler or by user settings in a hostfile or other resource
  description. Defaults to the job realm. Requires use of the PMIX_NODE_INFO attribute
  to avoid ambiguity when retrieving it.

Process-realm information shall include the following data for each process in the job, passed as a
pmix_data_array_t using the PMIX_PROC_INFO_ARRAY attribute:

- **PMIX_RANK** "pmix.rank" (pmix_rank_t)
  Process rank within the job, starting from zero.

- **PMIX_APPNUM** "pmix.appnum" (uint32_t)
  The application number within the job in which the specified process is a member. This
  attribute may be omitted if only one application is present in the namespace.
• **PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)
  Rank of the specified process within its application. This attribute may be omitted if only
  one application is present in the namespace.

• **PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)
  Rank of the specified process spanning across all jobs in this session, starting with zero.
  Note that no ordering of the jobs is implied when computing this value. As jobs can start
  and end at random times, this is defined as a continually growing number - i.e., it is not
dynamically adjusted as individual jobs and processes are started or terminated.

• **PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)
  Rank of the specified process on its node - refers to the numerical location (starting from
  zero) of the process on its node when counting only those processes from the same job
  that share the node, ordered by their overall rank within that job.

• **PMIX_NODE_RANK** "pmix.nrank" (uint16_t)
  Rank of the specified process on its node spanning all jobs - refers to the numerical location
  (starting from zero) of the process on its node when counting all processes (regardless of
  job) that share the node, ordered by their overall rank within the job. The value represents
  a snapshot in time when the specified process was started on its node and is not
dynamically adjusted as processes from other jobs are started or terminated on the node.

• **PMIX_NODEID** "pmix.nodeid" (uint32_t)
  Node identifier expressed as the node’s index (beginning at zero) in an array of nodes
  within the active session. The value must be unique and directly correlate to the
  **PMIX_HOSTNAME** of the node - i.e., users can interchangeably reference the same
  location using either the **PMIX_HOSTNAME** or corresponding **PMIX_NODEID**.

• **PMIX_REINCARNATION** "pmix.reinc" (uint32_t)
  Number of times this process has been re-instantiated - i.e, a value of zero indicates that
  the process has never been restarted. 5

• **PMIX_SPAWNED** "pmix.spawned" (bool)
  true if this process resulted from a call to **PMIx_Spawn**. Lack of inclusion (i.e., a return
  status of **PMIX_ERR_NOT_FOUND**) corresponds to a value of false for this attribute.

plus the following information for processes that are local to the server:

• **PMIX_LOCALITY_STRING** "pmix.locstr" (char*)
  String describing a process’s bound location - referenced using the process’s rank. The
  string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon.
  The remainder of the string represents the corresponding locality as expressed by the
  underlying implementation. The entire string must be passed to
  **PMIx_Get_relative_locality** for processing. Note that hosts are only required to
  provide locality strings for local client processes - thus, a call to **PMIx_Get** for the
  locality string of a process that returns **PMIX_ERR_NOT_FOUND** indicates that the
  process is not executing on the same node.
• **PMIX_PROCDIR** "pmix.pdir" (char*)
  Full path to the subdirectory under **PMIX_NSDIR** assigned to the specified process.

• **PMIX_PACKAGE_RANK** "pmix.pkgrank" (uint16_t)
  Rank of the specified process on the package where this process resides - refers to the numerical location (starting from zero) of the process on its package when counting only those processes from the same job that share the package, ordered by their overall rank within that job. Note that processes that are not bound to PUs within a single specific package cannot have a package rank.

and the following optional information - note that some of this information can be derived from information already provided by other attributes, but it may be included here for ease of retrieval by users:

• **PMIX_HOSTNAME** "pmix.hname" (char*)
  Name of the host, as returned by the `gethostname` utility or its equivalent.

• **PMIX_CPUSET** "pmix.cpuset" (char*)
  A string representation of the PU binding bitmap applied to the process upon launch. The string shall begin with the name of the library that generated it (e.g., "hwloc") followed by a colon and the bitmap string itself.

• **PMIX_CPUSET_BITMAP** "pmix.bitmap" (pmix_cpuset_t*)
  Bitmap applied to the process upon launch.

• **PMIX_DEVICE_DISTANCES** "pmix.dev.dist" (pmix_data_array_t)
  Return an array of `pmix_device_distance_t` containing the minimum and maximum distances of the given process location to all devices of the specified type on the local node.

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.

---

**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.

A NULL cbfunc reference indicates that the function is to be executed as a blocking operation.

Advice to users

The number of local processes for any given namespace is generally fixed at the time of application launch. Calls to PMIx_Spawn result in processes launched in their own namespace, not that of their parent. However, it is possible for processes to migrate to another node via a call to PMIx_Job_control_nb, thus resulting in a change to the number of local processes on both the initial node and the node to which the process moved. It is therefore critical that applications not migrate processes without first ensuring that PMIx-based collective operations are not in progress, and that no such operations be initiated until process migration has completed.
16.2.3.1 Namespace registration attributes

The following attributes are defined specifically for use with the
PMIx_server_register_nspace API: PMIX_REGISTER_NODATA
"pmix.reg.nodata" (bool)
Registration is for this namespace only, do not copy job data.

The following attributes are used to assemble information according to its data realm (session, job, application, node, or process as defined in Section 6.1) for registration where ambiguity may exist - see 16.2.3.2 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-realm information. The
PMIX_SESSION_ID attribute is required to be included in the array.

PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
Provide an array of pmix_info_t containing job-realm information. The
PMIX_SESSION_ID attribute of the session containing the job is required to be included in
the array whenever the PMix server library may host multiple sessions (e.g., when executing
with a host RM daemon). As information is registered one job (aka namespace) at a time via
the PMix_server_register_nspace API, there is no requirement that the array
contain either the PMIX_NSPACE or PMIX_JOBID attributes when used in that context
(though either or both of them may be included). At least one of the job identifiers must be
provided in all other contexts where the job being referenced is ambiguous.

PMIX_APP_INFO_ARRAY "pmix.app.arr" (pmix_data_array_t)
Provide an array of pmix_info_t containing application-realm information. The
PMIX_NSPACE or PMIX_JOBID attributes of the job containing the application, plus its
PMIX_APPNUM attribute, must to be included in the array when the array is not included as
part of a call to PMix_server_register_nspace - i.e., when the job containing the
application is ambiguous. The job identification is otherwise optional.

PMIX_PROC_INFO_ARRAY "pmix.pdata" (pmix_data_array_t)
Provide an array of pmix_info_t containing process-realm information. The
PMIX_RANK and PMIX_NSPACE attributes, or the PMIX_PROCID attribute, are required
to be included in the array when the array is not included as part of a call to
PMix_server_register_nspace - i.e., when the job containing the process is
ambiguous. All three may be included if desired. When the array is included in some
broader structure that identifies the job, then only the PMIX_RANK or the PMIX_PROCID
attribute must be included (the others are optional).

PMIX_NODE_INFO_ARRAY "pmix.node.arr" (pmix_data_array_t)
Provide an array of pmix_info_t containing node-realm information. At a minimum,
either the PMIX_NODEID or PMIX_HOSTNAME attribute is required to be included in the
array, though both may be included.

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 must be accessible via both **PMIx_Get** and **PMIx_Query_info_nb** APIs, assuming appropriate directives are provided by the caller.

### 16.2.3.2 Assembling the registration information

The following description is not intended to represent the actual layout of information in a given PMIx library. Instead, it describes how information provided in the *info* parameter of the **PMIx_server_register_nspace** shall be organized for proper processing by a PMIx server library. The ordering of the various information elements is arbitrary - they are presented in a top-down hierarchical form solely for clarity in reading.

--- Advice to PMIx server hosts ---

Creating the *info* array of data requires knowing in advance the number of elements required for the array. This can be difficult to compute and somewhat fragile in practice. One method for resolving the problem is to create a linked list of objects, each containing a single **pmix_info_t** structure. Allocation and manipulation of the list can then be accomplished using existing standard methods. Upon completion, the final *info* array can be allocated based on the number of elements on the list, and then the values in the list object **pmix_info_t** structures transferred to the corresponding array element utilizing the **PMIX_INFO_XFER** macro.

A common building block used in several areas is the construction of a regular expression identifying the nodes involved in that area - e.g., the nodes in a *session* or *job*. PMIx provides several tools to facilitate this operation, beginning by constructing an argv-like array of node names. This array is then passed to the **PMIx_generate_regex** function to create a regular expression parseable by the PMIx server library, as shown below:
Changing the filter criteria allows the construction of node maps for any level of information. A description of the returned regular expression is provided here.

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++) {
    /* for each node, construct an array of ranks on that node */
    ndppn = NULL;
    for (m=0; m < node[n]->num_procs; m++) {
        /* ignore processes that are not part of the target job */
        if (!PMIX_CHECK_NSPACE(targetjob,node[n]->proc[m].nspace)) {
            continue;
        }
        snprintf(rank, 30, "%d", node[n]->proc[m].rank);
        PMIX_ARGV_APPEND(&ndppn, rank);
    }
    /* convert the array into a comma-delimited string of ranks */
    localranks = PMIX_ARGV_JOIN(ndppn, ',');
    /* release the local array */
    PMIX_ARGV_FREE(ndppn);
    /* add this node’s contribution to the overall array */
    PMIX_ARGV_APPEND(&ppnarray, localranks);
    /* release the local list */
    free(localranks);
}

/* join into a semicolon-delimited string */
ppn = PMIX_ARGV_JOIN(ppnarray, ';');

/* release the array */
PMIX_ARGV_FREE(ppnarray);

/* generate ppn regex */
rc = PMIx_generate_ppn(ppn, &regex);

/* release list */
free(ppn);

ﾉ* pass the regex as the value to the PMIX_PROC_MAP key */
PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_REGEX);
ﾉ* 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 (PMIX_SESSION_ID and PMIX_UNIV_SIZE) are included in the registration array. Since both of these values are session-specific, they can be specified independently - i.e., in their own pmix_info_t elements of the info array. Alternatively, they can be provided as a pmix_data_array_t array of pmix_info_t using the PMIX_SESSION_INFO_ARRAY attribute and identified by including the PMIX_SESSION_ID attribute in the array - this is required in cases where non-specific attributes (e.g., PMIX_NUM_NODES or PMIX_NODE_MAP) are passed to describe aspects of the session. Note that the node map can include nodes not used by the job being registered as no corresponding process map is specified.

The info array at this point might look like (where the labels identify the corresponding attribute - e.g., “Session ID” corresponds to the PMIX_SESSION_ID attribute):

```
info   Univ   Session
size   Info
|       |
|       |
|       Num
|       nodes
```

Figure 16.1.: Session-level information elements

- Job-level information includes all job-specific values such as PMIX_JOB_SIZE, PMIX_JOB_NUM_APPS, and PMIX_JOBID. Since each invocation of PMIx_server_register_nspace describes a single job, job-specific values can be specified independently - i.e., in their own pmix_info_t elements of the info array. Alternatively, they can be provided as a pmix_data_array_t array of pmix_info_t identified by the PMIX_JOB_INFO_ARRAY attribute - this is required in cases where non-specific attributes (e.g., PMIX_NODE_MAP) are passed to describe aspects of the job. Note
that since the invocation only involves a single namespace, there is no need to include the
PMIX_NSPACE attribute in the array.

Upon conclusion of this step, the info array might look like:

```
Figure 16.2.: Job-level information elements
```

Note that in this example, PMIX_NUM_NODES is not required as that information is contained in
the PMIX_NODE_MAP attribute. Similarly, PMIX_JOB_SIZE is not technically required as that
information is contained in the PMIX_PROC_MAP when combined with the corresponding node
map - however, there is no issue with including the job size as a separate entry.

The example also illustrates the hierarchical use of the PMIX_NODE_INFO_ARRAY attribute.
In this case, we have chosen to pass several job-related values for each node - since those values
are non-unique across the job, they must be passed in a node-info container. Note that the choice
of what information to pass into the PMIx server library versus what information to derive from
other values at time of request is left to the host environment. PMIx implementors in turn may, if
they choose, pre-parse registration data to create expanded views (thus enabling faster response
to requests at the expense of memory footprint) or to compress views into tighter representations
(thus trading minimized footprint for longer response times).

- Application-level information includes all application-specific values such as PMIX_APP_SIZE
and `PMIX_APPLDR`. If the job contains only a single application, then the application-specific values can be specified independently - i.e., in their own `pmix_info_t` elements of the `info` array - or as a `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_APP_INFO_ARRAY` attribute and identified by including the `PMIX_APPNUM` attribute in the array. Use of the array format is must in cases where non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the `PMIX_APP_INFO_ARRAY` format, each identified by its `PMIX_APPNUM` value.

Upon conclusion of this step, the `info` array might look like that shown in 16.3, assuming there are two applications in the job being registered:

![Diagram of info array](image)

Figure 16.3.: Application-level information elements

- Process-level information includes an entry for each process in the job being registered, each entry marked with the `PMIX_PROC_INFO_ARRAY` attribute. The rank of the process must be the first entry in the array - this provides efficiency when storing the data. Upon conclusion of this step, the `info` array might look like the diagram in 16.4:

- For purposes of this example, node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases, the values included in this level are unique to it and can be specified independently - i.e., in their own `pmix_info_t` elements of the `info` array. Alternatively, they can be provided as a
`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:
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 an Hardware Locality (HWLOC) cpuset representation (a comma-delimited string of processor IDs) of the processors assigned to each process has previously been generated and stored on the process description. Thus, the value can be constructed as shown below:

char **ndcpus = NULL;
char *localcpus;
size_t m;
pmix_info_t info;

for (m=0; m < mynode->num_procs; m++) {
    /* ignore processes that are not part of the target job */
    if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace)) {
        continue;
    }
    /* convert the array into a comma-delimited string of ranks */
    localcpus = PMIX_ARGV_JOIN(ndcpus, ',');
    /* release the local array */
    PMIX_ARGV_FREE(ndcpus);
    /* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
    PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);
    /* release the list */
    free(localcpus);
PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);

/* convert the array into a colon-delimited string */
localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
/* release the local array */
PMIX_ARGV_FREE(ndcpus);

/* pass the string as the value to the PMIX_LOCAL_CPUSETS key */
PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSETS, localcpus, PMIX_STRING);

/* release the list */
free(localcpus);

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 16.5:

![Diagram of final information array]

Figure 16.5.: Final information array
### 16.2.4 PMIx_server_deregister_nspace

**Summary**
Deregister a namespace.

**Format**

```c
void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
                                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

**IN** `nspace`
Namespace (string)

**IN** `cbfunc`
Callback function `pmix_op_cbfunc_t`. A **NULL** function reference indicates that the function is to be executed as a blocking operation. (function reference)

**IN** `cbdata`
Data to be passed to the callback function (memory reference)

**Description**
Deregister the specified `nspace` and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library must not invoke the callback function prior to returning from the API, and that a **NULL** `cbfunc` reference indicates that the function is to be executed as a blocking operation.

### 16.2.5 PMIx_server_register_resources

**Summary**
Register non-namespace related information with the local PMIx server library.

**Format**

```c
pmix_status_t
PMIx_server_register_resources(pmix_info_t info[], size_t ninfo,
                               pmix_op_cbfunc_t cbfunc,
                               void *cbdata);
```

**IN** `info`
Array of info structures (array of handles)

**IN** `ninfo`
Number of elements in the `info` array (integer)
IN cbfunc
Callback function pmix_op_cbfunc_t. A NULL function reference indicates that the function is to be executed as a blocking operation (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Description
Pass information about resources not associated with a given namespace to the PMIx server library for distribution to local client processes. This includes information on fabric devices, GPUs, and other resources. All information provided through this API shall be made available to each job as part of its job-level information. Duplicate information provided with the PMIx_server_register_nspace API shall override any information provided by this function for that namespace, but only for that specific namespace.

Advice to PMIx server hosts
Note that information passed in this manner could also have been included in a call to PMIx_server_register_nspace - e.g., as part of a PMIX_NODE_INFO_ARRAY array. This API is provided as a logical alternative for code clarity, especially where multiple jobs may be supported by a single PMIx server library instance, to avoid multiple registration of static resource information.

A NULL cbfunc reference indicates that the function is to be executed as a blocking operation.

16.2.6 PMIx_server_deregister_resources

Summary
Remove specified non-namespace related information from the local PMIx server library.

Format

PMIx v4.0

pmix_status_t
PMIx_server_deregister_resources(pmix_info_t info[], size_t ninfo,
                               pmix_op_cbfunc_t cbfunc,
                               void *cbdata);

IN info
Array of info structures (array of handles)

IN ninfo
Number of elements in the info array (integer)

IN cbfunc
Callback function pmix_op_cbfunc_t. A NULL function reference indicates that the function is to be executed as a blocking operation (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)
Description
Remove information about resources not associated with a given namespace from the PMIx server library. Only the key fields of the provided info array shall be used for the operation - the associated values shall be ignored except where they serve as qualifiers to the request. For example, to remove a specific fabric device from a given node, the info array might include a PMIX_NODE_INFO_ARRAY containing the PMIX_NODEID or PMIX_HOSTNAME identifying the node hosting the device, and the PMIX_FABRIC_DEVICE_NAME specifying the device to be removed. Alternatively, the device could be removed using only the PMIX_DEVICE_ID as this is unique across the overall system.

Advice to PMIx server hosts
As information not related to namespaces is considered static, there is no requirement that the host environment deregister resources prior to finalizing the PMIx server library. The server library shall properly cleanup as part of its normal finalize operations. Deregistration of resources is only required, therefore, when the host environment determines that client processes should no longer have access to that information.

A NULL cbfunc reference indicates that the function is to be executed as a blocking operation.

16.2.7 PMIx_server_register_client

Summary
Register a client process with the PMIx server library.

Format

PMIx v1.0

pmix_status_t PMIx_server_register_client(const pmix_proc_t *proc,
uid_t uid, gid_t gid,
void *server_object,

pmix_op_cbfunc_t cbfunc, void *cbdata);

IN proc
pmix_proc_t structure (handle)
IN uid
user id (integer)
IN gid
group id (integer)
IN server_object
(memory reference)
IN  cbfunc
    Callback function pmix_op_cbfunc_t. A NULL function reference indicates that the
    function is to be executed as a blocking operation (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
  will be returned in the provided cbfunc. Note that the library must not invoke the callback
  function prior to returning from the API.

- PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called

Description
Register a client process with the PMIx server library.

The host server can also, if it desires, provide an object it wishes to be returned when a server
function is called that relates to a specific process. For example, the host server may have an object
that tracks the specific client. Passing the object to the library allows the library to provide that
object to the host server during subsequent calls related to that client, such as a
pmix_server_client_connected2_fn_t function. This allows the host server to access
the object without performing a lookup based on the client’s namespace and rank.

Advice to PMIx server hosts
Host environments are required to execute this operation prior to starting the client process. The
expected userID and groupID of the child process allows the server library to properly authenticate
clients as they connect by requiring the two values to match. Accordingly, the detected user and
group ID’s of the connecting process are not included in the
pmix_server_client_connected2_fn_t server module function.

Advice to PMIx library implementers
For security purposes, the PMIx server library should check the user and group ID’s of a
connecting process against those provided for the declared client process identifier via the
PMIx_server_register_client prior to completing the connection.
16.2.8 PMIx_server_deregister_client

Summary
Deregister a client and purge all data relating to it.

Format

```c
void PMIx_server_deregister_client(const pmix_proc_t *proc,
                                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

IN  `proc`  
`pmix_proc_t` structure (handle)

IN  `cbfunc`  
Callback function `pmix_op_cbfunc_t`. A `NULL` function reference indicates that the function is to be executed as a blocking operation (function reference)

IN  `cbdata`  
Data to be passed to the callback function (memory reference)

Description
The `PMIx_server_deregister_nspace` API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library must not invoke the callback function prior to returning from the API.

16.2.9 PMIx_server_setup_fork

Summary
Setup the environment of a child process to be forked by the host.

Format

```c
pmix_status_t PMIx_server_setup_fork(const pmix_proc_t *proc,
                                      char ***env);
```

IN  `proc`  
`pmix_proc_t` structure (handle)

IN  `env`  
Environment array (array of strings)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.
Description

Setup the environment of a child process to be forked by the host so it can correctly interact with
the PMIx server.

The PMIx client needs some setup information so it can properly connect back to the server. This
function will set appropriate environmental variables for this purpose, and will also provide any
environmental variables that were specified in the launch command (e.g., via PMIx_Spawn) plus
other values (e.g., variables required to properly initialize the client’s fabric library).

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process.

16.2.10 PMIx_server_dmodex_request

Summary

Define a function by which the host server can request modex data from the local PMIx server.

Format

PMIx v1.0

pmix_status_t

PMIx_server_dmodex_request(const pmix_proc_t *proc,
        pmix_dmodex_response_fn_t cbfunc,
        void *cbdata);

IN proc

    pmix_proc_t structure (handle)

IN 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 significant time impact at large scale.

PMIx supports an alternative wireup method known as Direct Modex that replaces the barrier-based exchange of all process-posted information with on-demand fetch of a peer’s data. In place of the barrier operation, data posted by each process is cached on the local PMIx server. When a process requests the information posted by a particular peer, it first checks the local cache to see if the data is already available. If not, then the request is passed to the local PMIx server, which subsequently requests that its RM host request the data from the RM daemon on the node where the specified peer process is located. Upon receiving the request, the RM daemon passes the request into its PMIx server library using the `PMIx_server_dmodex_request` function, receiving the response in the provided `cbfunc` once the indicated process has posted its information. The RM daemon then returns the data to the requesting daemon, who subsequently passes the data to its PMIx server library for transfer to the requesting client.

Advice to users

While direct modex allows for faster launch times by eliminating the barrier operation, per-peer retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by returning posted information from all processes on a node upon first request - but in general direct modex remains best suited for sparsely connected applications.

16.2.10.1 Server Direct Modex Response Callback Function

The `PMIx_server_dmodex_request` callback function.

Summary

Provide a function by which the local PMIx server library can return connection and other data posted by local application processes to the host resource manager.
typedef void (*pmix_dmodex_response_fn_t)(
    pmix_status_t status,
    char *data, size_t sz,
    void *cbdata);

IN status
    Returned status of the request (pmix_status_t)

IN data
    Pointer to a data "blob" containing the requested information (handle)

IN sz
    Number of bytes in the data blob (integer)

IN cbdata
    Data passed into the initial call to PMIx_server_dmodex_request (memory reference)

Description
Define a function to be called by the PMIx server library for return of information posted by a local
application process (via PMIx_Put with subsequent PMIx_Commit) in response to a request
from the host RM. The returned data blob is owned by the PMIx server library and will be free’d
upon return from the function.

16.2.11 PMIx_server_setup_application

Summary
Provide a function by which a launcher can request application-specific setup data prior to launch of
a job.

Format

PMIx v2.0

pmix_status_t
PMIx_server_setup_application(const pmix_nspace_t nspace,
    pmix_info_t info[], size_t ninfo,
    pmix_setup_application_cbfnc_t cbfunc,
    void *cbdata);

IN nspace
    namespace (string)

IN info
    Array of info structures (array of handles)
IN  ninfo
    Number of elements in the info array (integer)
IN  cbfunc
    Callback function pmix_setup_application_cbfunc_t (function reference)
IN  cbdata
    Data to be passed to the cbfunc callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.
- a PMIx error constant indicating either an error in the input - the cbfunc will not be called

```
--- Required Attributes ---
```

PMIx libraries that support this operation are required to support the following:

**PMIX_SETUP_APP_ENVARS** "pmix.setup.env" (bool)
Harvest and include relevant environmental variables.

**PMIX_SETUP_APP_NONENVARS** ""pmix.setup.nenv" (bool)
Include all relevant data other than environmental variables.

**PMIX_SETUP_APP_ALL** "pmix.setup.all" (bool)
Include all relevant data.

**PMIX_ALLOC_FABRIC** "pmix.alloc.net" (array)
Array of pmix_info_t describing requested fabric resources. This must include at least:
**PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and PMIX_ALLOC_FABRIC_ENDPTS**, plus whatever other descriptors are desired.

**PMIX_ALLOC_FABRIC_ID** "pmix.alloc.netid" (char*)
The key to be used when accessing this requested fabric allocation. The fabric allocation will be returned/stored as a pmix_data_array_t of pmix_info_t whose first element is composed of this key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as "32000-32100, 33005, 38123-38146". Additional array entries will consist of any provided resource request directives, along with their assigned values. Examples include:
**PMIX_ALLOC_FABRIC_TYPE** - the type of resources provided;
**PMIX_ALLOC_FABRIC_PLANE** - if applicable, what plane the resources were assigned from; **PMIX_ALLOC_FABRIC_QOS** - the assigned QoS; **PMIX_ALLOC_BANDWIDTH** - the allocated bandwidth; **PMIX_ALLOC_FABRIC_SEC_KEY** - a security key for the requested fabric allocation. NOTE: the array contents may differ from those requested, especially if **PMIX_INFO_REQD** was not set in the request.

**PMIX_ALLOC_FABRIC_SEC_KEY** "pmix.alloc.nsec" (pmix_byte_object_t)
Request that the allocation include a fabric security key for the spawned job.

```
PMIX_ALLOC_FABRIC_TYPE  "pmix.alloc.nettype" (char*)
Type of desired transport (e.g., “tcp”, “udp”) being requested in an allocation request.

PMIX_ALLOC_FABRIC_PLANE  "pmix.alloc.netplane" (char*)
ID string for the fabric plane to be used for the requested allocation.

PMIX_ALLOC_FABRIC_ENDPTS  "pmix.alloc.endpts" (size_t)
Number of endpoints to allocate per process in the job.

PMIX_ALLOC_FABRIC_ENDPTS_NODE  "pmix.alloc.endpts.nd" (size_t)
Number of endpoints to allocate per node for the job.

PMIX_PROC_MAP  "pmix.pmap" (char*)
Regular expression describing processes on each node in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the job realm.

PMIX_NODE_MAP  "pmix.nmap" (char*)
Regular expression of nodes currently hosting processes in the specified realm - see 16.2.3.2 for an explanation of its generation. Defaults to the job realm.
```

Optional Attributes

PMIx libraries that support this operation may support the following:

```
PMIX_ALLOC_BANDWIDTH  "pmix.alloc.bw" (float)
Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_QOS  "pmix.alloc.netqos" (char*)
Fabric quality of service level for the job being requested in an allocation request.

PMIX_SESSION_INFO  "pmix.ssn.info" (bool)
Return information regarding the session realm of the target process. In this context, indicates that the information provided in the PMIX_NODE_MAP is for the entire session and not just the indicated namespace. Thus, subsequent calls to this API may omit node-level information - e.g., the library may not need to include information on the devices on each node in a subsequent call.
```

The following optional attributes may be provided by the host environment to identify the programming model (as specified by the user) being executed within the application. The PMIx server library may utilize this information to harvest/forward model-specific environmental variables, record the programming model associated with the application, etc.

```
• PMIX_PROGRAMMING_MODEL  "pmix.pgm.model" (char*)
  Programming model being initialized (e.g., “MPI” or “OpenMP”).

• PMIX_MODEL_LIBRARY_NAME  "pmix.mdl.name" (char*)
```

324 PMIx Standard – Version 4.0 – December 2020
Programming model implementation ID (e.g., “OpenMPI” or “MPICH”).

- **PMIX_MODEL_LIBRARY_VERSION** "pmix.mld.vrs" (char*)

Programming model version string (e.g., “2.1.1”).

**Description**

Provide a function by which the RM can request application-specific setup data (e.g., environmental variables, fabric configuration and security credentials) from supporting PMIx server library subsystems prior to initiating launch of a job.

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 host environment 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 server hosts**

Host environments are required to execute this operation prior to launching a job. In addition to supported directives, the *info* array must include a description of the *job* using the **PMIX_NODE_MAP** and **PMIX_PROC_MAP** attributes.

Note that the function can be called on a per-application basis if the **PMIX_PROC_MAP** and **PMIX_NODE_MAP** are provided only for the corresponding application (as opposed to the entire job) each time.

---

**Advice to PMIx library implementers**

Support for harvesting of environmental variables and providing of local configuration information by the PMIx implementation is optional.

---

**16.2.11.1 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.
typedef void (*pmix_setup_application_cbfunc_t)(
    pmix_status_t status,
    pmix_info_t info[], size_t ninfo,
    void *provided_cbdata,
    pmix_op_cbfunc_t cbfunc, void *cbdata);

IN status
    returned status of the request (pmix_status_t)

IN info
    Array of info structures (array of handles)

IN ninfo
    Number of elements in the info array (integer)

IN provided_cbdata
    Data originally passed to call to PMIx_server_setup_application (memory reference)

IN cbfunc
    pmix_op_cbfunc_t function to be called when processing completed (function reference)

IN cbdata
    Data to be passed to the cbfunc callback function (memory reference)

Description
Define a function to be called by the PMIx server library for return of application-specific setup
data in response to a request from the host RM. The returned info array is owned by the PMIx
server library and will be free’d when the provided cbfunc is called.

16.2.11.2 Server Setup Application Attributes

PMIx v3.0 Attributes specifically defined for controlling contents of application setup data.

PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)
    Harvest and include relevant environmental variables.

PMIX_SETUP_APP_NONENVARS "pmix.setup.nenv" (bool)
    Include all relevant data other than environmental variables.

PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)
    Include all relevant data.

16.2.12 PMIx_Register_attributes

Summary
Register host environment attribute support for a function.
PMIx v4.0

Format

```c
pmix_status_t
PMIx_Register_attributes(char *function,
    pmix_regattr_t attrs[],
    size_t nattrs);
```

IN  `function`
    String name of function (string)

IN  `attrs`
    Array of `pmix_regattr_t` describing the supported attributes (handle)

IN  `nattrs`
    Number of elements in `attrs` (`size_t`)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Description

The `PMIx_Register_attributes` function is used by the host environment to register with its PMIx server library the attributes it supports for each `pmix_server_module_t` function. The `function` is the string name of the server module function (e.g., "register_events", "validate_credential", or "allocate") whose attributes are being registered. See the `pmix_regattr_t` entry for a description of the `attrs` array elements.

Note that the host environment can also query the library (using the `PMIx_Query_info_nb` API) for its attribute support both at the server, client, and tool levels once the host has executed `PMIx_server_init` since the server will internally register those values.

Advice to PMIx server hosts

Host environments are strongly encouraged to register all supported attributes immediately after initializing the library to ensure that user requests are correctly serviced.
Advice to PMIx library implementers

PMIx implementations are required to register all internally supported attributes for each API during initialization of the library (i.e., when the process calls their respective PMIx init function). Specifically, the implementation must not register supported attributes upon first call to a given API as this would prevent users from discovering supported attributes prior to first use of an API.

It is the implementation’s responsibility to associate registered attributes for a given pmix_server_module_t function with their corresponding user-facing API. Supported attributes must be reported to users in terms of their support for user-facing APIs, broken down by the level (see Section 5.4.6) 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.

16.2.12.1 Attribute registration constants

Constants supporting attribute registration.

PMIX_ERR_REPEAT_ATTR_REGISTRATION   The attributes for an identical function have already been registered at the specified level (host, server, or client).

16.2.12.2 Attribute registration structure

The pmix_regattr_t structure is used to register attribute support for a PMIx function.

```c
typedef struct pmix_regattr {
    char *name;
    pmix_key_t *string;
    pmix_data_type_t type;
    pmix_info_t *info;
    size_t ninfo;
    char **description;
} pmix_regattr_t;
```

Note that in this structure:

- the name is the actual name of the attribute - e.g., "PMIX_MAX_PROCS"
- the string is the literal string value of the attribute - e.g., "pmix.max.size" for the PMIX_MAX_PROCS attribute
- type must be a PMIx data type identifying the type of data associated with this attribute.
the info array contains machine-usable information regarding the range of accepted values. This may include entries for PMIX_MIN_VALUE, PMIX_MAX_VALUE, PMIX_ENUM_VALUE, or a combination of them. For example, an attribute that supports all positive integers might delineate it by including a pmix_info_t with a key of PMIX_MIN_VALUE, type of PMIX_INT, and value of zero. The lack of an entry for PMIX_MAX_VALUE indicates that there is no ceiling to the range of accepted values.

- ninfo indicates the number of elements in the info array

- The description field consists of a NULL-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values - e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the description and the number of elements in the info array is implied or required.

The attribute name and string fields must be NULL-terminated strings composed of standard alphanumeric values supported by common utilities such as strcmp.

Although not strictly required, both PMIx library implementers and host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them.

### 16.2.12.3 Attribute registration structure descriptive attributes

The following attributes relate to the nature of the values being reported in the pmix_regattr_t structures.

- **PMIX_MAX_VALUE** "pmix.descr.maxval" (varies)
  Used in pmix_regattr_t to describe the maximum valid value for the associated attribute.

- **PMIX_MIN_VALUE** "pmix.descr.minval" (varies)
  Used in pmix_regattr_t to describe the minimum valid value for the associated attribute.

- **PMIX_ENUM_VALUE** "pmix.descr.enum" (char*)
  Used in pmix_regattr_t to describe accepted values for the associated attribute.
  Numerical values shall be presented in a form convertible to the attribute’s declared data type. Named values (i.e., values defined by constant names via a typical C-language enum declaration) must be provided as their numerical equivalent.

### 16.2.12.4 Attribute registration structure support macros

The following macros are provided to support the pmix_regattr_t structure.

**Initialize the regattr structure**

Initialize the pmix_regattr_t fields
C

#include "pmix_regattr.h"

PMIx_REGATTR_CONSTRUCT(m)

IN m
Pointer to the structure to be initialized (pointer to pmix_regattr_t)

Destruct the regattr structure
Destruct the pmix_regattr_t fields, releasing all strings.

PMIx v4.0
PMIX_REGATTR_DESTRUCT(m)

IN m
Pointer to the structure to be destructed (pointer to pmix_regattr_t)

Create a regattr array
Allocate and initialize an array of pmix_regattr_t structures.

PMIx v4.0
PMIX_REGATTR_CREATE(m, n)

INOUT m
Address where the pointer to the array of pmix_regattr_t structures shall be stored (handle)

IN n
Number of structures to be allocated (size_t)

Free a regattr array
Release an array of pmix_regattr_t structures.

PMIx v4.0
PMIX_REGATTR_FREE(m, n)

INOUT m
Pointer to the array of pmix_regattr_t structures (handle)

IN n
Number of structures in the array (size_t)
Load a regattr structure

Load values into a `pmix_regattr_t` structure. The macro can be called multiple times to add as many strings as desired to the same structure by passing the same address and a NULL key to the macro. Note that the `t` type value must be given each time.

```c
PMIx v4.0

PMIX_REGATTR_LOAD(a, n, k, t, ni, v)
```

- **IN** `a`  
  Pointer to the structure to be loaded (pointer to `pmix_proc_t`)
- **IN** `n`  
  String name of the attribute (string)
- **IN** `k`  
  Key value to be loaded (`pmix_key_t`)
- **IN** `t`  
  Type of data associated with the provided key (`pmix_data_type_t`)
- **IN** `ni`  
  Number of `pmix_info_t` elements to be allocated in `info` (`size_t`)
- **IN** `v`  
  One-line description to be loaded (more can be added separately) (string)

Transfer a regattr to another regattr

Non-destructively transfer the contents of a `pmix_regattr_t` structure to another one.

```c
PMIx v4.0

PMIX_REGATTR_XFER(m, n)
```

- **INOUT** `m`  
  Pointer to the destination `pmix_regattr_t` structure (handle)
- **IN** `m`  
  Pointer to the source `pmix_regattr_t` structure (handle)

16.2.13 PMIx_server_setup_local_support

**Summary**

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.
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);

IN nspace
Namespace (string)

IN info
Array of info structures (array of handles)

IN ninfo
Number of elements in the info array (size_t)

IN cbfunc
Callback function pmix_op_cbfunc_t. A NULL function reference indicates that the
function is to be executed as a blocking operation (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
  will be returned in the provided cbfunc. Note that the library must not invoke the callback
  function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called

**Description**

Provide a function by which the local PMIx server can perform any application-specific operations
prior to spawning local clients of a given application. For example, a fabric library might need to
setup the local driver for “instant on” addressing. The data provided in the info array is the data
returned to the host RM by the callback function executed as a result of a call to

**PMIx_server_setup_application**.
Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting any local application processes from the specified namespace if information was obtained from a call to `PMIx_server_setup_application`.

Host environments must register the namespace using `PMIx_server_register_nspace` prior to calling this API to ensure that all namespace-related information required to support this function is available to the library. This eliminates the need to include any of the registration information in the `info` array passed to this API.

16.2.14 PMIx_server_IOF_deliver

Summary

Provide a function by which the host environment can pass forwarded Input/Output (IO) to the PMIx server library for distribution to its clients.

Format

```
PMIx v3.0

#include <pmix/deliver.h>

pmix_status_t
PMIx_server_IOF_deliver(const pmix_proc_t *source,
                        pmix_iof_channel_t channel,
                        const pmix_byte_object_t *bo,
                        const pmix_info_t info[], size_t ninfo,
                        pmix_op_cbfunc_t cbfunc, void *cbdata);
```

**IN** source
Pointer to `pmix_proc_t` identifying source of the IO (handle)

**IN** channel
IO channel of the data (`pmix_iof_channel_t`)

**IN** bo
Pointer to `pmix_byte_object_t` containing the payload to be delivered (handle)

**IN** info
Array of `pmix_info_t` metadata describing the data (array of handles)

**IN** ninfo
Number of elements in the `info` array (`size_t`)

**IN** cbfunc
Callback function `pmix_op_cbfunc_t`. A NULL function reference indicates that the function is to be executed as a blocking operation (function reference)

**IN** cbdata
Data to be passed to the callback function (memory reference)
Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

**Description**

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients. The PMIx server library is responsible for determining which of its clients have actually registered for the provided data and delivering it. The `cbfunc` callback function will be called once the PMIx server library no longer requires access to the provided data.

### 16.2.15 PMIx_server_collect_inventory

**Summary**

Collect inventory of resources on a node.

**Format**

```
PMIx v3.0

pmix_status_t
PMIx_server_collect_inventory(const pmix_info_t directives[],
size_t ndirs,
pmix_info_cbfunc_t cbfunc,
void *cbdata);
```

**IN** `directives`
- Array of `pmix_info_t` directing the request (array of handles)

**IN** `ndirs`
- Number of elements in the `directives` array (`size_t`)

**IN** `cbfunc`
- Callback function to return collected data (`pmix_info_cbfunc_t` function reference)

**IN** `cbdata`
- Data to be passed to the callback function (memory reference)

Returns **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant. In the event the function returns an error, the `cbfunc` will not be called.
Description

Provide a function by which the host environment can request its PMIx server library collect an inventory of local resources. Supported resources depends upon the PMIx implementation, but may include the local node topology and fabric interfaces.

Advice to PMIx server hosts

This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested information. Inventory collection is expected to be a rare event – at system startup and upon command from a system administrator. Inventory updates are expected to initiate a smaller operation involving only the changed information. For example, replacement of a node would generate an event to notify the scheduler with an inventory update without invoking a global inventory operation.

16.2.16 PMIx_server_deliver_inventory

Summary

Pass collected inventory to the PMIx server library for storage.

Format

PMIx v3.0

```c
pmix_status_t PMIx_server_deliver_inventory(const pmix_info_t info[],
                                          size_t ninfo,
                                          const pmix_info_t directives[],
                                          size_t ndirs,
                                          pmix_op_cbfunc_t cbfunc,
                                          void *cbdata);
```

IN info
Array of pmix_info_t containing the inventory (array of handles)

IN ninfo
Number of elements in the info array (size_t)

IN directives
Array of pmix_info_t directing the request (array of handles)

IN ndirs
Number of elements in the directives array (size_t)

IN cbfunc
Callback function pmix_op_cbfunc_t. A NULL function reference indicates that the function is to be executed as a blocking operation (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)
Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

- A PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

**Description**

Provide a function by which the host environment can pass inventory information obtained from a node (as a result of a call to `PMIx_server_collect_inventory`) to the PMIx server library for storage. Inventory data is subsequently used by the PMIx server library for allocations in response to `PMIx_server_setup_application`, and may be available to the library’s host via the `PMIx_Get` API (depending upon PMIx implementation). The `cbfunc` callback function will be called once the PMIx server library no longer requires access to the provided data.

16.2.17 **PMIx_server_generate_locality_string**

**Summary**

Generate a PMIx locality string from a given cpuset.

**Format**

```
PMIx v4.0

pmix_status_t
PMIx_server_generate_locality_string(const pmix_cpuset_t *cpuset,
char **locality);
```

**IN** `cpuset`
- Pointer to a `pmix_cpuset_t` containing the bitmap of assigned PUs (handle)

**OUT** `locality`
- String representation of the PMIx locality corresponding to the input bitmap (`char*`)

Returns either **PMIX_SUCCESS** indicating that the returned string contains the locality, or an appropriate PMIx error constant.
**Description**

Provide a function by which the host environment can generate a PMIx locality string for inclusion in the call to `PMIx_server_register_nspace`. This function shall only be called for local client processes, with the returned locality included in the job-level information (via the `PMIX_LOCALITY_STRING` attribute) provided to local clients. Local clients can use these strings as input to determine the relative locality of their local peers via the `PMIx_Get_relative_locality` API.

The function is required to return a string prefixed by the `source` field of the provided `cpuset` followed by a colon. The remainder of the string shall represent the corresponding locality as expressed by the underlying implementation.

### 16.2.18 PMIx_server_generate_cpuset_string

**Summary**

Generate a PMIx string representation of the provided cpuset.

**Format**

```c
pmix_status_t
PMIx_server_generate_cpuset_string(const pmix_cpuset_t *cpuset,
                                    char **cpuset_string);
```

**IN** `cpuset`

Pointer to a `pmix_cpuset_t` containing the bitmap of assigned PUs (handle)

**OUT** `cpuset_string`

String representation of the input bitmap (`char*`)

Returns either `PMIX_SUCCESS` indicating that the returned string contains the representation, or an appropriate PMIx error constant.

**Description**

Provide a function by which the host environment can generate a string representation of the cpuset bitmap for inclusion in the call to `PMIx_server_register_nspace`. This function shall only be called for local client processes, with the returned string included in the job-level information (via the `PMIX_CPUSET` attribute) provided to local clients. Local clients can use these strings as input to obtain their PU bindings via the `PMIx_Parse_cpuset_string` API.

The function is required to return a string prefixed by the `source` field of the provided `cpuset` followed by a colon. The remainder of the string shall represent the PUs to which the process is bound as expressed by the underlying implementation.
16.2.18.1 Cpuset Structure

The `pmix_cpuset_t` structure contains a character string identifying the source of the bitmap (e.g., "hwloc") and a pointer to the corresponding implementation-specific structure (e.g., `hwloc_cpuset_t`).

```c
PMIx v4.0
typedef struct pmix_cpuset {
    char *source;
    void *bitmap;
} pmix_cpuset_t;
```

16.2.18.2 Cpuset support macros

The following macros support the `pmix_cpuset_t` structure.

**Initialize the cpuset structure**

Initialize the `pmix_cpuset_t` fields.

```c
PMIx v4.0
PMIX_CPUSET_CONSTRUCT(m)
```

**IN** `m`  
Pointer to the structure to be initialized (pointer to `pmix_cpuset_t`)

**Destruct the cpuset structure**

Destruct the `pmix_cpuset_t` fields.

```c
PMIx v4.0
PMIX_CPUSET_DESTRUCT(m)
```

**IN** `m`  
Pointer to the structure to be destructed (pointer to `pmix_cpuset_t`)

**Create a cpuset array**

Allocate and initialize a `pmix_cpuset_t` array.

```c
PMIx v4.0
PMIX_CPUSET_CREATE(m, n)
```

**INOUT** `m`  
Address where the pointer to the array of `pmix_cpuset_t` structures shall be stored (handle)

**IN** `n`  
Number of structures to be allocated (size_t)
**Release a cpuset array**

Deconstruct and free a `pmix_cpuset_t` array.

```c
PMIX_CPUSET_FREE(m, n)
```

**INOUT m**
Address the array of `pmix_cpuset_t` structures to be released (handle)

**IN n**
Number of structures in the array (size_t)

### 16.2.19 PMIx_server_define_process_set

**Summary**
Define a PMIx process set.

**Format**

```c
pmix_status_t
PMIx_server_define_process_set(const pmix_proc_t members[],
                                size_t nmembers,
                                char *pset_name);
```

**IN members**
Pointer to an array of `pmix_proc_t` containing the identifiers of the processes in the process set (handle)

**IN nmembers**
Number of elements in `members` (integer)

**IN pset_name**
String name of the process set being defined (char*)

Returns either `PMIX_SUCCESS` or an appropriate PMIx error constant.

**Description**

Provide a function by which the host environment can create a process set. The PMIx server shall alert all local clients of the new process set (including process set name and membership) via the `PMIX_PROCESS_SET_DEFINE` event.

---

**Advice to PMIx server hosts**

The host environment is responsible for ensuring:

- consistent knowledge of process set membership across all involved PMIx servers; and
- that process set names do not conflict with system-assigned namespaces within the scope of the set
16.2.20 PMIx_server_delete_process_set

Summary
Delete a PMIx process set name

Format

PMIx v4.0

```c
pmix_status_t
PMIx_server_delete_process_set(char *pset_name);
```

IN pset_name
String name of the process set being deleted (char*)

Returns either PMIX_SUCCESS or an appropriate PMIx error constant.

Description
Provide a function by which the host environment can delete a process set name. The PMIx server shall alert all local clients of the process set name being deleted via the PMIX_PROCESS_SET_DELETE event. Deletion of the name has no impact on the member processes.

Advice to PMIx server hosts
The host environment is responsible for ensuring consistent knowledge of process set membership across all involved PMIx servers.

16.3 Server Function Pointers

PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the protocol. This method allows RMs to implement the server without being burdened with PMIx internal details. When a request is received from the client, the corresponding server function will be called with the information.

Any functions not supported by the RM can be indicated by a NULL for the function pointer. PMIx implementations are required to return a PMIX_ERR_NOT_SUPPORTED status to all calls to functions that require host environment support and are not backed by a corresponding server module entry. Host environments may, if they choose, include a function pointer for operations they have not yet implemented and simply return PMIX_ERR_NOT_SUPPORTED.

Functions that accept directives (i.e., arrays of pmix_info_t structures) must check any provided directives for those marked as required via the PMIX_INFO_REQD flag. PMIx client and server libraries are required to mark any such directives with the PMIX_INFO_REQD_PROCESSED flag should they have handled the request. Any required directive that has not been marked therefore becomes the responsibility of the host environment. If a required directive that hasn’t been
processed by a lower level cannot be supported by the host, then the
PMIX_ERR_NOT_SUPPORTED error constant must be returned. If the directive can be processed
by the host, then the host shall do so and mark the attribute with the
PMIX_INFO_REQD_PROCESSED flag.

The host RM will provide the function pointers in a pmix_server_module_t structure passed
to PMIx_server_init. The 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

16.3.1 pmix_server_module_t Module

Summary
List of function pointers that a PMIx server passes to PMIx_server_init during startup.

Format

```c
typedef struct pmix_server_module_4_0_0_t {
    /* v1x interfaces */
    pmix_server_client_connected_fn_t    client_connected;    // DEPRECATED
    pmix_server_client_finalized_fn_t   client_finalized;
    pmix_server_abort_fn_t              abort;
    pmix_server_fence_nb_fn_t           fence_nb;
    pmix_server_direct_modex_req_fn_t   direct_modex;
    pmix_server_publish_fn_t            publish;
    pmix_server_lookup_fn_t             lookup;
    pmix_server_unpublish_fn_t          unpublish;
    pmix_server_spawn_fn_t              spawn;
    pmix_server_connect_fn_t            connect;
    pmix_server_disconnect_fn_t         disconnect;
    pmix_server_register_events_fn_t    register_events;
    pmix_server_deregister_events_fn_t  deregister_events;
    pmix_server_listener_fn_t           listener;
    /* v2x interfaces */
    pmix_server_notify_event_fn_t       notify_event;
};
```

CHAPTER 16. SERVER-SPECIFIC INTERFACES 341
pmix_server_query_fn_t query;
pmix_server_tool_connection_fn_t tool_connected;
pmix_server_log_fn_t log);
pmix_server_alloc_fn_t allocate;
pmix_server_job_control_fn_t job_control;
pmix_server_monitor_fn_t monitor;
/* v3x interfaces */
pmix_server_get_credential fn_t get_credential;
pmix_server_validate_credential fn_t validate_credential;
pmix_server_iof_fn_t iof_pull;
pmix_server_stdio_fn_t push_stdio;
/* v4x interfaces */
pmix_server_grp_fn_t group;
pmix_server_fabric_fn_t fabric;
pmix_server_client_connected2 fn_t client_connected2;

} pmix_server_module_t;

Advice to PMIx server hosts

Note that some PMIx implementations require the use of C99-style designated initializers to clearly correlate each provided function pointer with the correct member of the pmix_server_module_t structure as the location/ordering of struct members may change over time.

16.3.2 pmix_server_client_connected_fn_t

Summary

Notify the host server that a client connected to this server. This function module entry has been DEPRECATED in favor of pmix_server_client_connected2 fn_t.
Format

```c
typedef pmix_status_t (*pmix_server_client_connected_fn_t)(
    const pmix_proc_t *proc,
    void* server_object,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```

**IN proc**  
`pmix_proc_t` structure (handle)

**IN server_object**  
Object reference (memory reference)

**IN cbfunc**  
Callback function `pmix_op_cbfunc_t` (function reference)

**IN cbdata**  
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the `cbfunc` will not be called

- A PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

**Description**
This function module entry has been DEPRECATED in favor of
`pmix_server_client_connected2_fn_t`. If both functions are provided, the PMIx library will ignore this function module entry in favor of its replacement.

### 16.3.3 pmix_server_client_connected2_fn_t

**Summary**
Notify the host server that a client connected to this server - this version of the original function definition has been extended to include an array of `pmix_info_t`, thereby allowing the PMIx server library to pass additional information identifying the client to the host environment.
Format

```c
typedef pmix_status_t (*pmix_server_client_connected2_fn_t)(
    const pmix_proc_t *proc,
    void* server_object,
    pmix_info_t info[], size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata)
```

### IN proc
- `pmix_proc_t` structure (handle)

### IN server_object
- object reference (memory reference)

### IN info
- Array of info structures (array of handles)

### IN ninfo
- Number of elements in the `info` array (integer)

### IN cbfunc
- Callback function `pmix_op_cbfunc_t` (function reference)

### IN cbdata
- Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called. The PMIx server library is to immediately terminate the connection.

**Description**

Notify the host environment that a client has called `PMIx_Init`. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The `server_object` parameter will be the value of the `server_object` parameter passed to `PMIx_server_register_client` by the host server when registering the connecting client. A host server can choose to not be notified when clients connect by setting `pmix_server_client_connected2_fn_t` to `NULL`.

It is possible that only a subset of the clients in a namespace call `PMIx_Init`. The server’s `pmix_server_client_connected2_fn_t` implementation should therefore not depend on
being called once per rank in a namespace or delay calling the callback function until all ranks have
connected. However, the host may rely on the `pmix_server_client_connected_fn_t`
function module entry being called for a given rank prior to any other function module entries
being executed on behalf of that rank.

16.3.4 `pmix_server_client_finalized_fn_t`

Summary

Notify the host environment that a client called `PMIx_Finalize`.

Format

```
PMIx v1.0

typedef pmix_status_t (*pmix_server_client_finalized_fn_t)(
    const pmix_proc_t *proc,
    void* server_object,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```

- **IN proc**
  - `pmix_proc_t` structure (handle)
- **IN server_object**
  - object reference (memory reference)
- **IN cbfunc**
  - Callback function `pmix_op_cbfunc_t` (function reference)
- **IN cbdata**
  - Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
  will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function
  prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
  returned `success` - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the `cbfunc` will not be called
Description
Notify the host environment that a client called PMIx_Finalize. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The server_object parameter will be the value of the server_object parameter passed to PMIx_server_register_client by the host server when registering the connecting client. If provided, an implementation of pmix_server_client_finalized_fn_t is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting pmix_server_client_finalized_fn_t to NULL.

Note that the host server is only being informed that the client has called PMIx_Finalize. The client might not have exited. If a client exits without calling PMIx_Finalize, the server support library will not call the pmix_server_client_finalized_fn_t implementation.

Advice to PMIx server hosts
This operation is an opportunity for a host server to update the status of the tasks it manages. It is also a convenient and well defined time to release resources used to support that client.

16.3.5 pmix_server_abort_fn_t

Summary
Notify the host environment that a local client called PMIx_Abort.

Format

```c
PMIx v1.0
typedef pmix_status_t (*pmix_server_abort_fn_t)(
  const pmix_proc_t *proc,
  void *server_object,
  int status,
  const char msg[],
  pmix_proc_t procs[],
  size_t nprocs,
  pmix_op_cbfunc_t cbfunc,
  void *cbdata);
```
<table>
<thead>
<tr>
<th>IN</th>
<th>proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmix_proc_t</td>
<td>structure identifying the process requesting the abort (handle)</td>
</tr>
<tr>
<td>IN</td>
<td>server_object</td>
</tr>
<tr>
<td></td>
<td>object reference (memory reference)</td>
</tr>
<tr>
<td>IN</td>
<td>status</td>
</tr>
<tr>
<td></td>
<td>exit status (integer)</td>
</tr>
<tr>
<td>IN</td>
<td>msg</td>
</tr>
<tr>
<td></td>
<td>exit status message (string)</td>
</tr>
<tr>
<td>IN</td>
<td>procs</td>
</tr>
<tr>
<td></td>
<td>Array of pmix_proc_t structures identifying the processes to be terminated (array of handles)</td>
</tr>
<tr>
<td>IN</td>
<td>nprocs</td>
</tr>
<tr>
<td></td>
<td>Number of elements in the procs array (integer)</td>
</tr>
<tr>
<td>IN</td>
<td>cbfunc</td>
</tr>
<tr>
<td></td>
<td>Callback function pmix_op_cbfunc_t (function reference)</td>
</tr>
<tr>
<td>IN</td>
<td>cbdata</td>
</tr>
<tr>
<td></td>
<td>Data to be passed to the callback function (memory reference)</td>
</tr>
</tbody>
</table>

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_PARAM_VALUE_NOT_SUPPORTED** indicating that the host environment supports this API, but the request includes processes that the host environment cannot abort - e.g., if the request is to abort subsets of processes from a namespace, or processes outside of the caller’s own namespace, and the host environment does not permit such operations. In this case, none of the specified processes will be terminated - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called
Description
A local client called \texttt{PMIx\_Abort}. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server library to release the client. The array of \textit{procs} indicates which processes are to be terminated. A \texttt{NULL} for the \textit{procs} array indicates that all processes in the caller’s namespace are to be aborted, including itself - this is the equivalent of passing a \texttt{pmix\_proc\_t} array element containing the caller’s namespace and a rank value of \texttt{PMIX\_RANK\_WILDCARD}.

16.3.6 \texttt{pmix\_server\_fencenb\_fn\_t}

Summary
At least one client called either \texttt{PMIx\_Fence} or \texttt{PMIx\_Fence\_nb}.

Format

\begin{verbatim}
PMIx v1.0

typedef pmix_status_t (*pmix_server_fencenb_fn_t)(
    const pmix_proc_t procs[],
    size_t nprocs,
    const pmix_info_t info[],
    size_t ninfo,
    char *data, size_t ndata,
    pmix_modex_cbfunc_t cbfunc,
    void *cbdata);
\end{verbatim}

\begin{itemize}
    \item \textbf{IN procs}
        Array of \texttt{pmix\_proc\_t} structures identifying operation participants(array of handles)
    \item \textbf{IN nprocs}
        Number of elements in the \textit{procs} array (integer)
    \item \textbf{IN info}
        Array of info structures (array of handles)
    \item \textbf{IN ninfo}
        Number of elements in the \textit{info} array (integer)
    \item \textbf{IN data}
        (string)
    \item \textbf{IN ndata}
        (integer)
    \item \textbf{IN cbfunc}
        Callback function \texttt{pmix\_modex\_cbfunc\_t} (function reference)
    \item \textbf{IN cbdata}
        Data to be passed to the callback function (memory reference)
\end{itemize}

Returns one of the following:
• **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.

• **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called

• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

### Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

The following attributes are required to be supported by all host environments:

**PMIX_COLLECT_DATA** "pmix.collect" (bool)

Collect all data posted by the participants using `PMIx_Put` that has been committed via `PMIx_Commit`, making the collection locally available to each participant at the end of the operation. By default, this will include all job-level information that was locally generated by PMIx servers unless excluded using the **PMIX_COLLECT_GENERATED_JOB_INFO** attribute.

### 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 (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

### Advice to PMIx server hosts

Host environment are required to return **PMIX_ERR_NOT_SUPPORTED** if passed an attributed marked as **PMIX_INFO_REQD** that they do not support, even if support for that attribute is optional.
Description

All local clients in the provided array of `procs` called either `PMIx_Fence` or `PMIx_Fence_nb`. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified `procs` array are required to participate in the `PMIx_Fence/PMIx_Fence_nb` operation. The callback is to be executed once every daemon hosting at least one participant has called the host server’s `pmix_server_fencenb_fn_t` function.

The provided data is to be collectively shared with all PMIx servers involved in the fence operation, and returned in the modex `cbfunc`. A `NULL` data value indicates that the local processes had no data to contribute.

The array of `info` structs is used to pass user-requested options to the server. This can include directives as to the algorithm to be used to execute the fence operation. The directives are optional unless the `PMIX_INFO_REQD` flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

------------------------ Advice to PMIx library implementers ------------------------

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

------------------------ Advice to PMIx server hosts ------------------------

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

```c
uint8_t *blob1, *blob2, *total;
size_t sz_blob1, sz_blob2, sz_total;

sz_total = sz_blob1 + sz_blob2;
total = (uint8_t*)malloc(sz_total);
memcpy(total, blob1, sz_blob1);
memcpy(&total[sz_blob1], blob2, sz_blob2);
```

Note that the ordering of the data blobs does not matter. The host is responsible for free’ing the `data` object passed to it by the PMIx server library.
16.3.6.1 Modex Callback Function

Summary

The `pmix_modex_cbfunc_t` is used by the `pmix_server_fencenb_fn_t` and `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex Business Card Exchange (BCX) data.

```
PMIx v1.0

typedef void (*pmix_modex_cbfunc_t)(pmix_status_t status,
    const char *data, size_t ndata,
    void *cbdata,
    pmix_release_cbfunc_t release_fn,
    void *release_cbdata);
```

**IN** status
Status associated with the operation (handle)

**IN** data
Data to be passed (pointer)

**IN** ndata
size of the data (`size_t`)

**IN** cbdata
Callback data passed to original API call (memory reference)

**IN** release_fn
Callback for releasing `data` (function pointer)

**IN** release_cbdata
Pointer to be passed to `release_fn` (memory reference)

Description

A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data in response to “fence” and “get” operations. The returned blob contains the data collected from each server participating in the operation.

16.3.7 `pmix_server_dmodex_req_fn_t`

Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified process to obtain and return a direct modex blob for that process.
Format

```
PMIx v1.0

typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
    const pmix_proc_t *proc,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_modex_cbfunc_t cbfunc,
    void *cbdata);
```

**IN** proc
- pmix_proc_t structure identifying the process whose data is being requested (handle)

**IN** info
- Array of info structures (array of handles)

**IN** ninfo
- Number of elements in the info array (integer)

**IN** cbfunc
- Callback function pmix_modex_cbfunc_t (function reference)

**IN** cbdata
- Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

---

**Required Attributes**

PMIx libraries are required to pass any provided attributes to the host environment for processing.

All host environments are required to support the following attributes:

**PMIX_REQUIRED_KEY** "pmix.req.key" (char*)
- Identifies a key that must be included in the requested information. If the specified key is not already available, then the PMIx servers are required to delay response to the dmodex request until either the key becomes available or the request times out.
The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**
Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to **PMIx_Put** and **PMIx_Commit**.

The array of info structs is used to pass user-requested options to the server. This can include a timeout to preclude an indefinite wait for data that may never become available. The directives are optional unless the mandatory flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

### 16.3.7.1 Dmodex attributes

**PMIX_REQUIRED_KEY** "pmix.req.key" (char*)

Identifies a key that must be included in the requested information. If the specified key is not already available, then the PMIx servers are required to delay response to the dmodex request until either the key becomes available or the request times out.

### 16.3.8 pmix_server_publish_fn_t

**Summary**
Publish data per the PMIx API specification.

**Format**

```c
PMIx v1.0

typedef pmix_status_t (*pmix_server_publish_fn_t)(
    const pmix_proc_t *proc,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```
IN proc
  pmix_proc_t structure of the process publishing the data (handle)
IN info
  Array of info structures (array of handles)
IN ninfo
  Number of elements in the info array (integer)
IN cbfunc
  Callback function pmix_op_cbfunc_t (function reference)
IN cbdata
  Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the host must not invoke the callback function prior to returning from the API.
- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called
- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

------------------ Required Attributes ------------------

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed info array:

**PMIX_USERID** "pmix.euid" (uint32_t)
  Effective user ID of the connecting process.

**PMIX_GRPID** "pmix.egid" (uint32_t)
  Effective group ID of the connecting process.

Host environments that implement this entry point are required to support the following attributes:

**PMIX_RANGE** "pmix.range" (pmix_data_range_t)
  Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

**PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)
Declare how long the datastore shall retain the provided data. The datastore is to delete the data upon reaching the persistence criterion.

Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

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.

16.3.9 **pmix_server_lookup_fn_t**

Summary

Lookup published data.
typedef pmix_status_t (*pmix_server_lookup_fn_t)(
const pmix_proc_t *proc,
char **keys,
const pmix_info_t info[],
size_t ninfo,
pmix_lookup_cbfunc_t cbfunc,
void *cbdata);

IN  proc
    pmix_proc_t structure of the process seeking the data (handle)
IN  keys
    (array of strings)
IN  info
    Array of info structures (array of handles)
IN  ninfo
    Number of elements in the info array (integer)
IN  cbfunc
    Callback function pmix_lookup_cbfunc_t (function reference)
IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
  will be returned in the provided cbfunc. Note that the host must not invoke the callback function
  prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the
  request, even though the function entry was provided in the server module - the cbfunc will not
  be called

- a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called

**Required Attributes**

PMIx libraries are required to pass any provided attributes to the host environment for processing.
In addition, the following attributes are required to be included in the passed info array:

**PMIX_USERID**  "pmix.euid" (uint32_t)

Effective user ID of the connecting process.
Host environments that implement this entry point are required to support the following attributes:

**PMIX_GRPID** "pmix.egid" (uint32_t)

Effective group ID of the connecting process.

Host environments that implement this entry point are required to support the following attributes:

**PMIX_RANGE** "pmix.range" (pmix_data_range_t)

Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

**PMIX_WAIT** "pmix.wait" (int)

Caller requests that the PMIx server wait until at least the specified number of values are found (a value of zero indicates all and is the default).

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

**Description**

Lookup published data. The host server will be passed a **NULL**-terminated array of string keys identifying the data being requested.

The array of info structs is used to pass user-requested options to the server. The default data range is left to the host environment, but expected to be **PMIX_RANGE_SESSION**. This can include a wait flag to indicate that the server should wait for all data to become available before executing the callback function, or should immediately callback with whatever data is available. In addition, a timeout can be specified on the wait to preclude an indefinite wait for data that may never be published.

**Advice to PMIx server hosts**

The **PMIX_USERID** and **PMIX_GRPID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn’t support a specified range so long as it is covered by some internally defined range.
16.3.10 pmix_server_unpublish_fn_t

Summary
Delete data from the data store.

Format

\[
\text{typedef pmix_status_t (*pmix_server_unpublish_fn_t)}(\text{const pmix_proc_t *proc,}} \]
\[
\text{char **keys,}} \]
\[
\text{const pmix_info_t info[]}, \]
\[
\text{size_t ninfo,}} \]
\[
\text{pmix_op_cbfunc_t cbfunc,}} \]
\[
\text{void *cbdata);}} \]
\]

IN \text{proc}
\text{pmix_proc_t structure identifying the process making the request (handle)}

IN \text{keys}
\text{(array of strings)}

IN \text{info}
\text{Array of info structures (array of handles)}

IN \text{ninfo}
\text{Number of elements in the info array (integer)}

IN \text{cbfunc}
\text{Callback function pmix_op_cbfunc_t (function reference)}

IN \text{cbdata}
\text{Data to be passed to the callback function (memory reference)}

Returns one of the following:

- \text{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.

- \text{PMIX_OPERATION_SUCCEEDED}, indicating that the request was immediately processed and returned success - the cbfunc will not be called.

- \text{PMIX_ERR_NOT_SUPPORTED}, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called.
Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.
In addition, the following attributes are required to be included in the passed info array:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  Effective user ID of the connecting process.

- **PMIX_GRP_ID** "pmix.egid" (uint32_t)
  Effective group ID of the connecting process.

Host environments that implement this entry point are required to support the following attributes:

- **PMIX_RANGE** "pmix.range" (pmix_data_range_t)
  Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

Delete data from the data store. The host server will be passed a NULL-terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The default data range is left to the host environment, but expected to be **PMIX_RANGE_SESSION**.
The callback is to be executed upon completion of the delete procedure.

Advice to PMIx server hosts

The **PMIX_USERID** and **PMIX_GRP_ID** of the requesting process will be provided to support authorization-based access to published information. The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn’t support a specified range so long as it is covered by some internally defined range.
16.3.11 pmix_server_spawn_fn_t

Summary
Spawn a set of applications/processes as per the PMIx_Spawn API.

Format

```c
typedef pmix_status_t (*pmix_server_spawn_fn_t)(
    const pmix_proc_t *proc,
    const pmix_info_t job_info[],
    size_t ninfo,
    const pmix_app_t apps[],
    size_t napps,
    pmix_spawn_cbfunc_t cbfunc,
    void *cbdata);
```

**IN** `proc`
- pmix_proc_t structure of the process making the request (handle)

**IN** `job_info`
- Array of info structures (array of handles)

**IN** `ninfo`
- Number of elements in the `jobinfo` array (integer)

**IN** `apps`
- Array of pmix_app_t structures (array of handles)

**IN** `napps`
- Number of elements in the `apps` array (integer)

**IN** `cbfunc`
- Callback function pmix_spawn_cbfunc_t (function reference)

**IN** `cbdata`
- Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

- **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 server libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed info array:

**PMIX_USERID** "pmix.euid" (uint32_t)
Effective user ID of the connecting process.

**PMIX_GRPID** "pmix.egid" (uint32_t)
Effective group ID of the connecting process.

**PMIX_SPAWNED** "pmix.spawned" (bool)
true if this process resulted from a call to PMIx_Spawn. Lack of inclusion (i.e., a return status of PMIX_ERR_NOT_FOUND) corresponds to a value of false for this attribute.

**PMIX_PARENT_ID** "pmix.parent" (pmix_proc_t)
Process identifier of the parent process of the specified process - typically used to identify the application process that caused the job containing the specified process to be spawned (e.g., the process that called PMIx_Spawn).

**PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)
The requesting process is a PMIx tool.

**PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)
The requesting process is a PMIx client.

Host environments that provide this module entry point are required to pass the PMIX_SPAWNED and PMIX_PARENT_ID attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the job_info or the info array of an element of the apps array:

**PMIX_WDIR** "pmix.wdir" (char*)
Working directory for spawned processes.

**PMIX_SET_SESSION_CWD** "pmix.ssncwd" (bool)
Set the current working directory to the session working directory assigned by the RM - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

**PMIX_PREFIX** "pmix.prefix" (char*)
Prefix to use for starting spawned processes - i.e., the directory where the executables can be found.

**PMIX_HOST** "pmix.host" (char*)
Comma-delimited list of hosts to use for spawned processes.

```
PMIX_HOSTFILE "pmix.hostfile" (char*)
Hostfile to use for spawned processes.
```

The following attributes are optional for host environments that support this operation:

```
PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)
Hostfile containing hosts to add to existing allocation.

PMIX_ADD_HOST "pmix.addhost" (char*)
Comma-delimited list of hosts to add to the allocation.

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
Preload executables onto nodes prior to executing launch procedure.

PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
Comma-delimited list of files to pre-position on nodes prior to executing launch procedure.

PMIX_PERSONALITY "pmix.pers" (char*)
Name of personality corresponding to programming model used by application - supported values depend upon PMIx implementation.

PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
Display process mapping upon spawn.

PMIX_PPR "pmix.ppr" (char*)
Number of processes to spawn on each identified resource.

PMIX_MAPBY "pmix.mapby" (char*)
Process mapping policy - when accessed using PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the provided namespace. Supported values are launcher specific.

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. Supported values are launcher specific.

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. Supported values are launcher specific.

PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
Spawned process rank that is to receive any forwarded stdin.

PMIX_FWD_STDIN "pmix.fwd.stdin" (pmix_rank_t)
The requester intends to push information from its stdin to the indicated process. The local spawn agent should, therefore, ensure that the stdin channel to that process remains available. A rank of PMIX_RANK_WILDCARD indicates that all processes in the spawned job are potential recipients. The requester will issue a call to PMIx_IOF_push to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
Requests that the ability to forward the stdout of the spawned processes be maintained. The requester will issue a call to PMIx_IOF_pull to specify the callback function and other options for delivery of the forwarded output.

PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
Requests that the ability to forward the stderr of the spawned processes be maintained. The requester will issue a call to PMIx_IOF_pull to specify the callback function and other options for delivery of the forwarded output.

PMIX_DEBUGGER_DAEMONS "pmix.debugger" (bool)
Included in the pmix_info_t array of a pmix_app_t, this attribute declares that the application consists of debugger daemons and shall be governed accordingly. If used as the sole pmix_app_t in a PMIx_Spawn request, then the PMIX_DEBUG_TARGET attribute must also be provided (in either the job_info or in the info array of the pmix_app_t) to identify the namespace to be debugged so that the launcher can determine where to place the spawned daemons. If neither PMIX_DEBUG_DAEMONS_PER_PROC nor PMIX_DEBUG_DAEMONS_PER_NODE is specified, then the launcher shall default to a placement policy of one daemon per process in the target job.

PMIX_TAG_OUTPUT "pmix.tagout" (bool)
Tag stdout/stderr with the identity of the source process - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
Timestamp output - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
Merge stdout and stderr streams - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
Direct output (both stdout and stderr) into files of form "<filename>.rank" - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

PMIX_INDEX_ARGV "pmix.indxargv" (bool)
Mark the `argv` with the rank of the process.

**PMIX_CPUS_PER_PROC** "pmix.cpuperproc" (uint32_t)
Number of PUs to assign to each rank - when accessed using `PMIx_Get`, use the
**PMIX_RANK_WILDCARD** value for the rank to discover the PUs/process assigned to the
provided namespace.

**PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)
Do not place processes on the head node.

**PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)
Do not oversubscribe the nodes - i.e., do not place more processes than allocated slots on a
node.

**PMIX_REPORT_BINDINGS** "pmix.repbind" (bool)
Report bindings of the individual processes.

**PMIX_CPU_LIST** "pmix.cpulist" (char*)
List of PUs to use for this job - when accessed using `PMIx_Get`, use the
**PMIX_RANK_WILDCARD** value for the rank to discover the PU list used for the provided
namespace.

**PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)
Application supports recoverable operations.

**PMIX_JOB_CONTINUOUS** "pmix.continuous" (bool)
Application is continuous, all failed processes should be immediately restarted.

**PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)
Maximum number of times to restart a process - when accessed using `PMIx_Get`, use the
**PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
namespace.

**PMIX_TIMEOUT** "pmix.timeout" (int)
Time in seconds before the specified operation should time out (zero indicating infinite) and
return the `PMIX_ERR_TIMEOUT` error. Care should be taken to avoid race conditions
caused by multiple layers (client, server, and host) simultaneously timing the operation.

---

**Description**

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.
16.3.11.1 Server spawn attributes

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.

16.3.12 pmix_server_connect_fn_t

Summary
Record the specified processes as connected.

Format
PMIx v1.0

```c
typedef pmix_status_t (*pmix_server_connect_fn_t)(
  const pmix_proc_t procs[],
  size_t nprocs,
  const pmix_info_t info[],
  size_t ninfo,
  pmix_op_cbfunc_t cbfunc,
  void *cbdata);
```

IN procs
  Array of pmix_proc_t structures identifying participants (array of handles)

IN nprocs
  Number of elements in the procs array (integer)

IN info
  Array of info structures (array of handles)

IN ninfo
  Number of elements in the info array (integer)

IN cbfunc
  Callback function pmix_op_cbfunc_t (function reference)

IN cbdata
  Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called
• **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called

• a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

---

### Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

---

### Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_TIMEOUT** "pmix.timeout" (int)

Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

---

### Description

Record the processes specified by the `procs` array as `connected` as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server’s `pmix_server_connect_fn_t` function, and the host environment has completed any supporting operations required to meet the terms of the PMIx definition of `connected` processes.

---

### Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

---

### Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

---

### 16.3.13 pmix_server_disconnect_fn_t

**Summary**

Disconnect a previously connected set of processes.
Format

```c
typedef pmix_status_t (*pmix_server_disconnect_fn_t)(
    const pmix_proc_t procs[],
    size_t nprocs,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```

IN  procs
    Array of `pmix_proc_t` structures identifying participants (array of handles)

IN  nprocs
    Number of elements in the `procs` array (integer)

IN  info
    Array of info structures (array of handles)

IN  ninfo
    Number of elements in the `info` array (integer)

IN  cbfunc
    Callback function `pmix_op_cbfunc_t` (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the `cbfunc` will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.
Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the **pmix_server_disconnect_fn_t** function, and the host environment has completed any required supporting operations.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective.

A **PMIX_ERR_INVALID_OPERATION** error must be returned if the specified set of **procs** was not previously **connected** via a call to the **pmix_server_connect_fn_t** function.

16.3.14 **pmix_server_register_events_fn_t**

Summary

Register to receive notifications for the specified events.
typedef pmix_status_t (*pmix_server_register_events_fn_t)(
    pmix_status_t *codes,
    size_t ncodes,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);

IN  codes
    Array of pmix_status_t values (array of handles)

IN  ncodes
    Number of elements in the codes array (integer)

IN  info
    Array of info structures (array of handles)

IN  ninfo
    Number of elements in the info array (integer)

IN  cbfunc
    Callback function pmix_op_cbfunc_t (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

• PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
  will be returned in the provided cbfunc. Note that the host must not invoke the callback function
  prior to returning from the API.

• PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called

• PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
  request, even though the function entry was provided in the server module - the cbfunc will not
  be called

• a PMIx error constant indicating either an error in the input or that the request was immediately
  processed and failed - the cbfunc will not be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.
In addition, the following attributes are required to be included in the passed info array:

PMIX_USERID  "pmix.euid" (uint32_t)
    Effective user ID of the connecting process.
PMIX_GRPID "pmix.egid" (uint32_t)
   Effective group ID of the connecting process.

Description
Register to receive notifications for the specified status codes. The info array included in this API is reserved for possible future directives to further steer notification.

Advice to PMIx library implementers
The PMIx server library must track all client registrations for subsequent notification. This module function shall only be called when:

- the client has requested notification of an environmental code (i.e., a PMIx codes in the range between PMIX_EVENT_SYS_BASE and PMIX_EVENT_SYS_OTHER, inclusive) or codes that lies outside the defined PMIx range of constants; and

- the PMIx server library has not previously requested notification of that code - i.e., the host environment is to be contacted only once a given unique code value

Advice to PMIx server hosts
The host environment is required to pass to its PMIx server library all non-environmental events that directly relate to a registered namespace without the PMIx server library explicitly requesting them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in the range between PMIX_EVENT_SYS_BASE and PMIX_EVENT_SYS_OTHER (inclusive).

16.3.15 pmix_server_deregister_events_fn_t

Summary
Deregister to receive notifications for the specified events.
Format

```c
typedef pmix_status_t (*pmix_server_deregister_events_fn_t)(
    pmix_status_t *codes,
    size_t ncodes,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```

**IN** codes

Array of `pmix_status_t` values (array of handles)

**IN** ncodes

Number of elements in the `codes` array (integer)

**IN** cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

**IN** cbdata

Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

**Description**

Deregister to receive notifications for the specified events to which the PMIx server has previously registered.

**Advice to PMIx library implementers**

The PMIx server library must track all client registrations. This module function shall only be called when:

- the library is deregistering environmental codes (i.e., a PMIx codes in the range between `PMIX_EVENT_SYS_BASE` and `PMIX_EVENT_SYS_OTHER`, inclusive) or codes that lies outside the defined PMIx range of constants; and
• no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it remain.

16.3.16 pmix_server_notify_event_fn_t

Summary
Notify the specified processes of an event.

Format

```c
typedef pmix_status_t (*pmix_server_notify_event_fn_t)(
    pmix_status_t code,
    const pmix_proc_t *source,
    pmix_data_range_t range,
    pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata);
```

**IN** code
The `pmix_status_t` event code being referenced structure (handle)

**IN** source
`pmix_proc_t` of process that generated the event (handle)

**IN** range
`pmix_data_range_t` range over which the event is to be distributed (handle)

**IN** info
Optional array of `pmix_info_t` structures containing additional information on the event (array of handles)

**IN** ninfo
Number of elements in the `info` array (integer)

**IN** cbfunc
Callback function `pmix_op_cbfunc_t` (function reference)

**IN** cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

• **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function prior to returning from the API.
• **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called

• **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.

Host environments that provide this module entry point are required to support the following attributes:

**PMIX_RANGE** "*pmix.range*" (*pmix_data_range_t*)

Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

---

**Description**

Notify the specified processes (described through a combination of *range* and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. The process generating the event is provided in the *source* parameter, and any further descriptive information is included in the *info* array.

Note that the PMIx server library is not allowed to echo any event given to it by its host via the **PMIx_Notify_event** API back to the host through the **pmix_server_notify_event_fn_t** server module function.

--- Advice to PMIx server hosts ---

The callback function is to be executed once the host environment no longer requires that the PMIx server library maintain the provided data structures. It does not necessarily indicate that the event has been delivered to any process, nor that the event has been distributed for delivery.

---

**16.3.17 pmix_server_listener_fn_t**

**Summary**

Register a socket the host server can monitor for connection requests.
typedef pmix_status_t (*pmix_server_listener_fn_t)(
    int listening_sd,
    pmix_connection_cbfunc_t cbfunc,
    void *cbdata);

IN incoming_sd
   (integer)

IN cbfunc
   Callback function pmix_connection_cbfunc_t (function reference)

IN cbdata
   (memory reference)

Returns PMIX_SUCCESS indicating that the request is accepted, or a negative value corresponding
to a PMIx error constant indicating that the request has been rejected.

Description
Register a socket the host environment can monitor for connection requests, harvest them, and then
call the PMIx server library’s internal callback function for further processing. A listener thread is
essential to efficiently harvesting connection requests from large numbers of local clients such as
occur when running on large SMPs. The host server listener is required to call accept on the
incoming connection request, and then pass the resulting socket to the provided cbfunc. A NULL
for this function will cause the internal PMIx server to spawn its own listener thread.

16.3.17.1 PMIx Client Connection Callback Function

Summary
Callback function for incoming connection request from a local client.

typedef void (*pmix_connection_cbfunc_t)(
    int incoming_sd, void *cbdata);

IN incoming_sd
   (integer)

IN cbdata
   (memory reference)

Description
Callback function for incoming connection requests from local clients - only used by host
environments that wish to directly handle socket connection requests.
16.3.18  pmix_server_query_fn_t

Summary
Query information from the resource manager.

Format

```c
typedef pmix_status_t (*pmix_server_query_fn_t)(
    pmix_proc_t *proct,
    pmix_query_t *queries,
    size_t nqueries,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata);
```

IN  proct
    pmix_proc_t structure of the requesting process (handle)

IN  queries
    Array of pmix_query_t structures (array of handles)

IN  nqueries
    Number of elements in the queries array (integer)

IN  cbfunc
    Callback function pmix_info_cbfunc_t (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
  will be returned in the provided cbfunc. Note that the host must not invoke the callback function
  prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
  returned success - the cbfunc will not be called

- **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
PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed info array:

**Required Attributes**

- **PMIX_USERID** "pmix.euid" (uint32_t)
  - Effective user ID of the connecting process.
- **PMIX_GRPID** "pmix.egid" (uint32_t)
  - Effective group ID of the connecting process.

**Optional Attributes**

The following attributes are optional for host environments that support this operation:

- **PMIX_QUERY_NAMESPACES** "pmix.qry.ns" (char*)
  - Request a comma-delimited list of active namespaces. NO QUALIFIERS.
- **PMIX_QUERY_JOB_STATUS** "pmix.qry.jst" (pmix_status_t)
  - Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose status is being queried.
- **PMIX_QUERY_QUEUE_LIST** "pmix.qry.qlst" (char*)
  - Request a comma-delimited list of scheduler queues. NO QUALIFIERS.
- **PMIX_QUERY_QUEUE_STATUS** "pmix.qry.qst" (char*)
  - Returns status of a specified scheduler queue, expressed as a string. OPTIONAL QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being requested.
- **PMIX_QUERY_PROC_TABLE** "pmix.qry.ptable" (char*)
  - Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried.
- **PMIX_QUERY_LOCAL_PROC_TABLE** "pmix.qry.lptable" (char*)
  - Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each process in the specified namespace executing on the same node as the requester, ordered by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose local process table is being queried. OPTIONAL QUALIFIER: PMIX_HOSTNAME indicating the host whose local process table is being queried. By default, the query assumes that the host upon which the request was made is to be used.
- **PMIX_QUERY SPAWN SUPPORT** "pmix.qry.spawn" (bool)
  - Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS.
- **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
  - Return a comma-delimited list of supported debug attributes. NO QUALIFIERS.
PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
Return information on memory usage for the processes indicated in the qualifiers.
OPTIONAL QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of specific process(es) whose memory usage is being requested.

PMIX_QUERY_LOCALONLY "pmix.qry.local" (bool)
Constrain the query to local information only. NO QUALIFIERS.

PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
Report only average values for sampled information. NO QUALIFIERS.

PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
Report minimum and maximum values. NO QUALIFIERS.

PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
String identifier of the allocation whose status is being requested. NO QUALIFIERS.

PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
OPTIONAL QUALIFIERS: PMIX_NSPACE of the namespace whose info is being requested (defaults to allocation containing the caller).

Description
Query information from the host environment. The query will include the namespace/rank of the process that is requesting the info, an array of pmix_query_t describing the request, and a callback function/data for the return.

Advice to PMIx library implementers
The PMIx server library should not block in this function as the host environment may, depending upon the information being requested, require significant time to respond.

16.3.19 pmix_server_tool_connection_fn_t
Summary
Register that a tool has connected to the server.
typedef void (*pmix_server_tool_connection_fn_t)(
    pmix_info_t info[], size_t ninfo,
    pmix_tool_connection_cbfunc_t cbfunc,
    void *cbdata);

IN  info
    Array of pmix_info_t structures (array of handles)
IN  ninfo
    Number of elements in the info array (integer)
IN  cbfunc
    Callback function pmix_tool_connection_cbfunc_t (function reference)
IN  cbdata
    Data to be passed to the callback function (memory reference)

PMIx libraries are required to pass the following attributes in the info array:

PMIX_USERID  "pmix.euid" (uint32_t)
    Effective user ID of the connecting process.

PMIX_GRPID  "pmix.egid" (uint32_t)
    Effective group ID of the connecting process.

PMIX_TOOL_NSPACE  "pmix.tool.nspace" (char*)
    Name of the namespace to use for this tool. This must be included only if the tool already has an assigned namespace.

PMIX_TOOL_RANK  "pmix.tool.rank" (uint32_t)
    Rank of this tool. This must be included only if the tool already has an assigned rank.

PMIX_CREDENTIAL  "pmix.cred" (char*)
    Security credential assigned to the process.
Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)

Requests that the ability to forward the `stdout` of the spawned processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and other options for delivery of the forwarded output.

**PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)

Requests that the ability to forward the `stderr` of the spawned processes be maintained. The requester will issue a call to `PMIx_IOF_pull` to specify the callback function and other options for delivery of the forwarded output.

**PMIX_FWD_STDIN** "pmix.fwd.stdin" (pmix_rank_t)

The requester intends to push information from its `stdin` to the indicated process. The local spawn agent should, therefore, ensure that the `stdin` channel to that process remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the spawned job are potential recipients. The requester will issue a call to `PMIx_IOF_push` to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

**PMIX_VERSION_INFO** "pmix.version" (char*)

PMIx version of the library being used by the connecting process.

---

Description

Register that a tool has connected to the server, possibly requesting that the tool be assigned a namespace/rank identifier for further interactions. The `pmix_info_t` array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

If the tool already has an assigned process identifier, then this must be indicated in the `info` array. The host is responsible for checking that the provided namespace does not conflict with any currently known assignments, returning an appropriate error in the callback function if a conflict is found.

The host environment is solely responsible for authenticating and authorizing the connection using whatever means it deems appropriate. If certificates or other authentication information are required, then the tool must provide them. The conclusion of those operations shall be communicated back to the PMIx server library via the callback function.

Approval or rejection of the connection request shall be returned in the `status` parameter of the `pmix_tool_connection_cbfunc_t`. If the connection is refused, the PMIx server library must terminate the connection attempt. The host must not execute the callback function prior to returning from the API.
16.3.19.1 Tool connection attributes

Attributes associated with tool connections.

PMIX_USERID "pmix.euid" (uint32_t)
Effective user ID of the connecting process.

PMIX_GRPID "pmix.egid" (uint32_t)
Effective group ID of the connecting process.

PMIX_VERSION_INFO "pmix.version" (char*)
PMIx version of the library being used by the connecting process.

16.3.19.2 PMIx Tool Connection Callback Function

Summary
Callback function for incoming tool connections.

Format

PMIx v2.0

typedef void (*pmix_tool_connection_cbfunc_t)(
    pmix_status_t status,
    pmix_proc_t *proc, void *cbdata);

IN status
    pmix_status_t value (handle)

IN proc
    pmix_proc_t structure containing the identifier assigned to the tool (handle)

IN cbdata
    Data to be passed (memory reference)

Description
Callback function for incoming tool connections. The host environment shall provide a
namespace/rank identifier for the connecting tool.

Advice to PMIx server hosts

It is assumed that rank=0 will be the normal assignment, but allow for the future possibility of a
parallel set of tools connecting, and thus each process requiring a unique rank.

16.3.20 pmix_server_log_fn_t

Summary
Log data on behalf of a client.
typedef void (*pmix_server_log_fn_t)(
    const pmix_proc_t *client,
    const pmix_info_t data[], size_t ndata,
    const pmix_info_t directives[], size_t ndirs,
    pmix_op_cbfunc_t cbfunc, void *cbdata);

IN client
    pmix_proc_t structure (handle)
IN data
    Array of info structures (array of handles)
IN ndata
    Number of elements in the data array (integer)
IN directives
    Array of info structures (array of handles)
IN ndirs
    Number of elements in the directives array (integer)
IN cbfunc
    Callback function pmix_op_cbfunc_t (function reference)
IN cbdata
    Data to be passed to the callback function (memory reference)

--------------------------- Required Attributes ---------------------------

Pmix libraries are required to pass any provided attributes to the host environment for processing.
In addition, the following attributes are required to be included in the passed info array:

PMIX_USERID "pmix.euid" (uint32_t)
    Effective user ID of the connecting process.

PMIX_GRPID "pmix.egid" (uint32_t)
    Effective group ID of the connecting process.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_LOG_STDERR "pmix.log.stderr" (char*)
    Log string to stderr.

PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
    Log string to stdout.

PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog.

### Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_LOG_MSG** "pmix.log.msg" *(pmix_byte_object_t)*
  - Message blob to be sent somewhere.

- **PMIX_LOG_EMAIL** "pmix.log.email" *(pmix_data_array_t)*
  - Log via email based on `pmix_info_t` containing directives.

- **PMIX_LOG_EMAIL_ADDR** "pmix.log.emaddr" *(char*)
  - Comma-delimited list of email addresses that are to receive the message.

- **PMIX_LOG_EMAIL_SUBJECT** "pmix.log.emsub" *(char*)
  - Subject line for email.

- **PMIX_LOG_EMAIL_MSG** "pmix.log.emmsg" *(char*)
  - Message to be included in email.

### Description

Log data on behalf of a client. This function is not intended for output of computational results, but rather for reporting status and error messages. The host must not execute the callback function prior to returning from the API.

### 16.3.21 pmix_server_alloc_fn_t

#### Summary

Request allocation operations on behalf of a client.
typedef pmix_status_t (*pmix_server_alloc_fn_t)(
    const pmix_proc_t *client,
    pmix_alloc_directive_t directive,
    const pmix_info_t data[],
    size_t ndata,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata);

IN  client
    pmix_proc_t structure of process making request (handle)

IN  directive
    Specific action being requested (pmix_alloc_directive_t)

IN  data
    Array of info structures (array of handles)

IN  ndata
    Number of elements in the data array (integer)

IN  cbfunc
    Callback function pmix_info_cbfunc_t (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

**Required Attributes**

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed info array:

**PMIX_USERID** "pmix.euid" (uint32_t)
    Effective user ID of the connecting process.
PMIX_GRP_ID "pmix.egid" (uint32_t)

Effective group ID of the connecting process.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_ALLOC_ID "pmix.alloc.id" (char*)

A string identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to PMIX_Spawn.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)

The number of nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)

Number of PUs being requested in an allocation request.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)

Total session time (in seconds) being requested in an allocation request.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)

Regular expression of the specific nodes being requested in an allocation request.

PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)

Regular expression of the number of PUs for each node being requested in an allocation request.

PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)

Regular expression of the specific PUs being requested in an allocation request.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)

Number of Megabytes[base2] of memory (per process) being requested in an allocation request.

PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)

Array of pmix_info_t describing requested fabric resources. This must include at least: PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
The key to be used when accessing this requested fabric allocation. The fabric allocation will be returned/stored as a `pmix_data_array_t` of `pmix_info_t` whose first element is composed of this key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as "32000-32100, 33005, 38123-38146". Additional array entries will consist of any provided resource request directives, along with their assigned values. Examples include:

- `PMIX_ALLOC_FABRIC_TYPE` - the type of resources provided;
- `PMIX_ALLOC_FABRIC_PLAN` - if applicable, what plane the resources were assigned from; `PMIX_ALLOC_FABRIC_QOS` - the assigned QoS; `PMIX_ALLOC_BANDWIDTH` - the allocated bandwidth; `PMIX_ALLOC_FABRIC_SEC_KEY` - a security key for the requested fabric allocation. NOTE: the array contents may differ from those requested, especially if `PMIX_INFO_REQD` was not set in the request.

```
PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
Fabric bandwidth (in Megabits\[base2]/sec) for the job being requested in an allocation request.

PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
Fabric quality of service level for the job being requested in an allocation request.
```

Description
Request new allocation or modifications to an existing allocation on behalf of a client. Several broad categories are envisioned, including the ability to:

- Request allocation of additional resources, including memory, bandwidth, and compute for an existing allocation. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities.
- Return no-longer-required resources to the scheduler. This includes the loan of resources back to the scheduler with a promise to return them upon subsequent request.

The callback function provides a `status` to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of `pmix_info_t` structures.

16.3.22 `pmix_server_job_control_fn_t`

Summary
Execute a job control action on behalf of a client.
typedef pmix_status_t (*pmix_server_job_control_fn_t)(
    const pmix_proc_t *requestor,
    const pmix_proc_t targets[],
    size_t ntargets,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata);

IN   requestor
    pmix_proc_t structure of requesting process (handle)

IN   targets
    Array of proc structures (array of handles)

IN   ntargets
    Number of elements in the targets array (integer)

IN   directives
    Array of info structures (array of handles)

IN   ndirs
    Number of elements in the info array (integer)

IN   cbfunc
    Callback function pmix_info_cbfunc_t (function reference)

IN   cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called
Required Attributes

PMIx libraries are required to pass any attributes provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the passed info array:

**PMIX_USERID** "pmix.euid" (uint32_t)
   Effective user ID of the connecting process.

**PMIX_GRPID** "pmix.egid" (uint32_t)
   Effective group ID of the connecting process.

Host environments that provide this module entry point are required to support the following attributes:

**PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (char*)
   Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

**PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (bool)
   Pause the specified processes.

**PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (bool)
   Resume ("un-pause") the specified processes.

**PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (bool)
   Forcibly terminate the specified processes and cleanup.

**PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (int)
   Send given signal to specified processes.

**PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)
   Politely terminate the specified processes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

**PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)
   Cancel the specified request - the provided request ID must match the
   **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control**. An ID of
   NULL implies cancel all requests from this requestor.

**PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)
   Restart the specified processes using the given checkpoint ID.

**PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)
   Checkpoint the specified processes and assign the given ID to it.
**PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)

Use event notification to trigger a process checkpoint.

**PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)

Use the given signal to trigger a process checkpoint.

**PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)

Time in seconds to wait for a checkpoint to complete.

**PMIX_JOB_CTRL_CHECKPOINT_METHOD**

"pmix.jctrl.ckmethod" (pmix_data_array_t)

Array of pmix_info_t declaring each method and value supported by this application.

**PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)

Regular expression identifying nodes that are to be provisioned.

**PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvnimg" (char*)

Name of the image that is to be provisioned.

**PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

Indicate that the job can be pre-empted.

---

**Description**

Execute a job control action on behalf of a client. The targets array identifies the processes to which the requested job control action is to be applied. A NULL value can be used to indicate all processes in the caller's namespace. The use of PMIX_RANK_WILDCARD can also be used to indicate that all processes in the given namespace are to be included.

The directives are provided as pmix_info_t structures in the directives array. The callback function provides a status to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the pmix_info_cbfunc_t array of pmix_info_t structures.

---

**16.3.23 pmix_server_monitor_fn_t**

**Summary**

Request that a client be monitored for activity.
typedef pmix_status_t (*pmix_server_monitor_fn_t)(
    const pmix_proc_t *requestor,
    const pmix_info_t *monitor,
    pmix_status_t error,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata);

IN  requestor
    pmix_proc_t  structure of requesting process (handle)

IN  monitor
    pmix_info_t  identifying the type of monitor being requested (handle)

IN  error
    Status code to use in generating event if alarm triggers (integer)

IN  directives
    Array of info structures (array of handles)

IN  ndirs
    Number of elements in the info array (integer)

IN  cbfunc
    Callback function pmix_info_cbfunc_t (function reference)

IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the host must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called

This entry point is only called for monitoring requests that are not directly supported by the PMIx server library itself.
**Required Attributes**

If supported by the PMIx server library, then the library must not pass any supported attributes to the host environment. Any attributes provided by the client that are not directly supported by the server library must be passed to the host environment if it provides this module entry. In addition, the following attributes are required to be included in the passed `info` array:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  
  Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (uint32_t)
  
  Effective group ID of the connecting process.

Host environments are not required to support any specific monitoring attributes.

**Optional Attributes**

The following attributes may be implemented by a host environment.

- **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)
  
  Provide a string identifier for this request.

- **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)
  
  Identifier to be canceled (NULL means cancel all monitoring for this process).

- **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)
  
  The application desires to control the response to a monitoring event - i.e., the application is requesting that the host environment not take immediate action in response to the event (e.g., terminating the job).

- **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)
  
  Register to have the PMIx server monitor the requestor for heartbeats.

- **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)
  
  Time in seconds before declaring heartbeat missed.

- **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)
  
  Number of heartbeats that can be missed before generating the event.

- **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)
  
  Register to monitor file for signs of life.

- **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)
  
  Monitor size of given file is growing to determine if the application is running.

- **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)
  
  Monitor time since last access of given file to determine if the application is running.

- **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)
  
  Monitor time since last modified of given file to determine if the application is running.
**Descriptio**
Request that a client be monitored for activity.

### 16.3.24 pmix_server_get_cred_fn_t

**Summary**
Request a credential from the host environment.

**Format**

```c
typedef pmix_status_t (*pmix_server_get_cred_fn_t)(
    const pmix_proc_t *proc,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_credential_cbfunc_t cbfunc,
    void *cbdata);
```

**IN**
- `proc` pmix_proc_t structure of requesting process (handle)
- `directives` Array of info structures (array of handles)
- `ndirs` Number of elements in the info array (integer)
- `cbfunc` Callback function to return the credential (pmix_credential_cbfunc_t function reference)
- `cbdata` Data to be passed to the callback function (memory reference)

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`
- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called
Required Attributes

If the PMIx library does not itself provide the requested credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed info array:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  - Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (uint32_t)
  - Effective group ID of the connecting process.

Optional Attributes

The following attributes are optional for host environments that support this operation:

- **PMIX_CRED_TYPE** "pmix.sec.ctype" (char*)
  - When passed in PMIx_Get_credential, a prioritized, comma-delimited list of desired credential types for use in environments where multiple authentication mechanisms may be available. When returned in a callback function, a string identifier of the credential type.

- **PMIX_TIMEOUT** "pmix.timeout" (int)
  - Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

Request a credential from the host environment.

16.3.24.1 Credential callback function

Summary

Callback function to return a requested security credential
Format

**PMIx v3.0**

```c
typedef void (*pmix_credential_cbfunc_t)(
    pmix_status_t status,
    pmix_byte_object_t *credential,
    pmix_info_t info[], size_t ninfo,
    void *cbdata);
```

**IN status**
- `pmix_status_t` value (handle)

**IN credential**
- `pmix_byte_object_t` structure containing the security credential (handle)

**IN info**
- Array of provided by the system to pass any additional information about the credential - e.g., the identity of the issuing agent. (handle)

**IN ninfo**
- Number of elements in `info` (`size_t`)

**IN cbdata**
- Object passed in original request (memory reference)

**Description**
Define a callback function to return a requested security credential. Information provided by the issuing agent can subsequently be used by the application for a variety of purposes. Examples include:

- checking identified authorizations to determine what requests/operations are feasible as a means to steering workflows
- compare the credential type to that of the local SMS for compatibility

**Advice to users**
The credential is opaque and therefore understandable only by a service compatible with the issuer. The `info` array is owned by the PMIx library and is not to be released or altered by the receiving party.

**16.3.25 pmix_server_validate_cred_fn_t**

**Summary**
Request validation of a credential.
Format

```c
typedef pmix_status_t (*pmix_server_validate_cred_fn_t)(
    const pmix_proc_t *proc,
    const pmix_byte_object_t *cred,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_validation_cbfunc_t cbfunc,
    void *cbdata);
```

IN `proc`  
`pmix_proc_t` structure of requesting process (handle)

IN `cred`  
Pointer to `pmix_byte_object_t` containing the credential (handle)

IN `directives`  
Array of info structures (array of handles)

IN `ndirs`  
Number of elements in the `info` array (integer)

IN `cbfunc`  
Callback function to return the result (`pmix_validation_cbfunc_t` function reference)

IN `cbdata`  
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`
- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the `cbfunc` will not be called
- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

Required Attributes

If the PMIx library does not itself validate the credential, then it is required to pass any attributes provided by the client to the host environment for processing. In addition, it must include the following attributes in the passed `info` array:

**PMIX_USERID** "pmix.euid" (uint32_t)

  Effective user ID of the connecting process.
PMIX_GRP_ID "pmix.egid" (uint32_t)
Effective group ID of the connecting process.

Host environments are not required to support any specific attributes.

Optional Attributes

The following attributes are optional for host environments that support this operation:

PMIX_TIMEOUT "pmix.timeout" (int)
Time in seconds before the specified operation should time out (zero indicating infinite) and return the PMIX_ERR_TIMEOUT error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description
Request validation of a credential obtained from the host environment via a prior call to the pmix_server_get_cred_fn_t module entry.

16.3.26 Credential validation callback function

Summary
Callback function for security credential validation.
typedef void (*pmix_validation_cbfunc_t)(
    pmix_status_t status,
    pmix_info_t info[], size_t ninfo,
    void *cbdata);

IN status
    pmix_status_t value (handle)

IN info
    Array of pmix_info_t provided by the system to pass any additional information about the
    authentication - e.g., the effective userid and group id of the certificate holder, and any related
    authorizations (handle)

IN ninfo
    Number of elements in info (size_t)

IN cbdata
    Object passed in original request (memory reference)

The returned status shall be one of the following:

- **PMIX_SUCCESS**, indicating that the request was processed and returned success (i.e., the
  credential was both valid and any information it contained was successfully processed). Details
  of the result will be returned in the info array

- a PMIx error constant indicating either an error in the parsing of the credential or that the request
  was refused

**Description**
Define a validation callback function to indicate if a provided credential is valid, and any
corresponding information regarding authorizations and other security matters.

**Advice to users**
The precise contents of the array will depend on the host environment and its associated security
system. At the minimum, it is expected (but not required) that the array will contain entries for the
**PMIX_USERID** and **PMIX_GRPID** of the client described in the credential. The info array is
owned by the PMIx library and is not to be released or altered by the receiving party.

**16.3.27 pmix_server_iоф_fn_t**

**Summary**
Request the specified IO channels be forwarded from the given array of processes.
typedef pmix_status_t (*pmix_server_iof_fn_t)(
    const pmix_proc_t procs[],
    size_t nprocs,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_iof_channel_t channels,
    pmix_op_cbfunc_t cbfunc, void *cbdata);

IN procs
Array pmix_proc_t identifiers whose IO is being requested (handle)

IN nprocs
Number of elements in procs (size_t)

IN directives
Array of pmix_info_t structures further defining the request (array of handles)

IN ndirs
Number of elements in the info array (integer)

IN channels
Bitmask identifying the channels to be forwarded (pmix_iof_channel_t)

IN cbfunc
Callback function pmix_op_cbfunc_t (function reference)

IN cbdata
Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called
Required Attributes

The following attributes are required to be included in the passed `info` array:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (uint32_t)
  Effective group ID of the connecting process.

Host environments that provide this module entry point are required to support the following attributes:

- **PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (uint32_t)
  The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

- **PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (bool)
  In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.

- **PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (bool)
  In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).

Optional Attributes

The following attributes may be supported by a host environment.

- **PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (uint32_t)
  Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be flushed to the callback upon a call to deregister the respective channel.

- **PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (uint32_t)
  Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
Description

Request the specified IO channels be forwarded from the given array of processes. An error shall be returned in the callback function if the requested service from any of the requested processes cannot be provided.

Advice to PMIx library implementers

The forwarding of stdin is a push process - processes cannot request that it be pulled from some other source. Requests including the PMIX_FWD_STDIN_CHANNEL channel will return a PMIX_ERR_NOT_SUPPORTED error.

16.3.27.1 IOF delivery function

Summary

Callback function for delivering forwarded IO to a process.

Format

| PMIx v3.0 |
|-----------|-----------|
| C         |

typedef void (*pmix_iof_cbfunc_t)(
  size_t iofhdlr, pmix_iof_channel_t channel,
  pmix_proc_t *source, char *payload,
  pmix_info_t info[], size_t ninfo);

IN iofhdlr
Registration number of the handler being invoked (size_t)

IN channel
bitmask identifying the channel the data arrived on (pmix_iof_channel_t)

IN source
Pointer to a pmix_proc_t identifying the namespace/rank of the process that generated the data (char*)

IN payload
Pointer to character array containing the data.

IN info
Array of pmix_info_t provided by the source containing metadata about the payload. This could include PMIX_IOF_COMPLETE (handle)

IN ninfo
Number of elements in info (size_t)
Define a callback function for delivering forwarded IO to a process. This function will be called whenever data becomes available, or a specified buffering size and/or time has been met.

**Advice to users**

Multiple strings may be included in a given payload, and the payload may not be NULL terminated. The user is responsible for releasing the payload memory. The info array is owned by the PMIx library and is not to be released or altered by the receiving party.

### 16.3.28 pmix_server_stdin_fn_t

**Summary**
Pass standard input data to the host environment for transmission to specified recipients.

**Format**

```c
#include <pmix/pmix.h>

typedef pmix_status_t (*pmix_server_stdin_fn_t)(
    const pmix_proc_t *source,
    const pmix_proc_t targets[],
    size_t ntargets,
    const pmix_info_t directives[],
    size_t ndirs,
    const pmix_byte_object_t *bo,
    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

**Description**

1. **IN source**
   - pmix_proc_t structure of source process (handle)
2. **IN targets**
   - Array of pmix_proc_t target identifiers (handle)
3. **IN ntargets**
   - Number of elements in the targets array (integer)
4. **IN directives**
   - Array of info structures (array of handles)
5. **IN ndirs**
   - Number of elements in the info array (integer)
6. **IN bo**
   - Pointer to pmix_byte_object_t containing the payload (handle)
7. **IN cbfunc**
   - Callback function pmix_op_cbfunc_t (function reference)
8. **IN cbdata**
   - Data to be passed to the callback function (memory reference)
Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the `cbfunc` will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

---

**Required Attributes**

The following attributes are required to be included in the passed `info` array:

- **PMIX_USERID** "pmix.euid" (uint32_t)
  - Effective user ID of the connecting process.

- **PMIX_GRPID** "pmix.egid" (uint32_t)
  - Effective group ID of the connecting process.

---

**Description**

Passes stdin to the host environment for transmission to specified recipients. The host environment is responsible for forwarding the data to all locations that host the specified `targets` and delivering the payload to the PMIx server library connected to those clients.

---

**16.3.29 pmix_server_grp_fn_t**

**Summary**

Request group operations (construct, destruct, etc.) on behalf of a set of processes.
typedef pmix_status_t (*pmix_server_grp_fn_t)(
    pmix_group_operation_t op,
    char grp[],
    const pmix_proc_t procs[],
    size_t nprocs,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata);

IN  op
    pmix_group_operation_t value indicating operation the host is requested to perform
        (integer)
IN  grp
    Character string identifying the group (string)
IN  procs
    Array of pmix_proc_t identifiers of participants (handle)
IN  nprocs
    Number of elements in the procs array (integer)
IN  directives
    Array of info structures (array of handles)
IN  ndirs
    Number of elements in the info array (integer)
IN  cbfunc
    Callback function pmix_info_cbfunc_t (function reference)
IN  cbdata
    Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided cbfunc. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the cbfunc will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the cbfunc will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the cbfunc will not be called
The following attributes may be supported by a host environment.

**PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)
Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, `size_t` value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to `PMIX_RANGE_SESSION`.

**PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)
Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

**PMIX_GROUP_ENDPT_DATA** "pmix.grp.endpt" (`pmix_byte_object_t`)
Data collected during group construction to ensure communication between group members is supported upon completion of the operation.

**PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)
Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is `false`.

**PMIX_RANGE** "pmix.range" (`pmix_data_range_t`)
Define constraints on the processes that can access the provided data. Only processes that meet the constraints are allowed to access it.

The following attributes may be included in the host’s response:

**PMIX_GROUP_ID** "pmix.grp.id" (`char*`)
User-provided group identifier - as the group identifier may be used in PMIx operations, the user is required to ensure that the provided ID is unique within the scope of the host environment (e.g., by including some user-specific or application-specific prefix or suffix to the string).

**PMIX_GROUP_MEMBERSHIP** "pmix.grp.mbrs" (`pmix_data_array_t*`)
Array `pmix_proc_t` identifiers identifying the members of the specified group.

**PMIX_GROUP_CONTEXT_ID** "pmix.grp.ctxid" (`size_t`)
Context identifier assigned to the group by the host RM.

**PMIX_GROUP_ENDPT_DATA** "pmix.grp.endpt" (`pmix_byte_object_t`)
Data collected during group construction to ensure communication between group members is supported upon completion of the operation.
Perform the specified operation across the identified processes, plus any special actions included in
the directives. Return the result of any special action requests in the callback function when the
operation is completed. Actions may include a request (PMIX_GROUP_ASSIGN_CONTEXT_ID )
that the host assign a unique numerical (size_t) ID to this group - if given, the PMIX_RANGE
attribute will specify the range across which the ID must be unique (default to
PMIX_RANGE_SESSION).

### 16.3.29.1 Group Operation Constants

#### PMIx v4.0

The pmix_group_operation_t structure is a uint8_t value for specifying group
operations. All values were originally defined in version 4 of the standard unless otherwise marked.

- **PMIX_GROUP_CONSTRUCT**— Construct a group composed of the specified processes - used by
  a PMIx server library to direct host operation.
- **PMIX_GROUP_DESTRUCT**— Destruct the specified group - used by a PMIx server library to
direct host operation.

### 16.3.30 pmix_server_fabric_fn_t

#### Summary

Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

#### Format

```
PMIx v4.0

typedef pmix_status_t (*pmix_server_fabric_fn_t)(
    const pmix_proc_t *requestor,
    pmix_fabric_operation_t op,
    const pmix_info_t directives[],
    size_t ndirs,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata);
```

#### Description

- **IN** `requestor`— identifying the requestor (handle)
- **IN** `op`— pmix_fabric_operation_t value indicating operation the host is requested to perform
  (integer)
- **IN** `directives`— Array of info structures (array of handles)
- **IN** `ndirs`— Number of elements in the info array (integer)
IN cbfunc
  Callback function `pmix_info_cbfunc_t` (function reference)

IN cbdata
  Data to be passed to the callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called

- **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the request, even though the function entry was provided in the server module - the `cbfunc` will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

--- Required Attributes ---

The following directives are required to be supported by all hosts to aid users in identifying the fabric and (if applicable) the device to whom the operation references:

**PMIX_FABRIC_VENDOR** "pmix.fab.vndr" (string)
  Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

**PMIX_FABRIC_IDENTIFIER** "pmix.fab.id" (string)
  An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

**PMIX_FABRIC_PLANE** "pmix.fab.plane" (string)
  ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly as a key in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in the overall system.

**PMIX_FABRIC_DEVICE_INDEX** "pmix.fabdev.idx" (uint32_t)
  Index of the device within an associated communication cost matrix.

---

**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 `pmix_fabric_t` 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 `pmix_info_t` using the `PMIX_FABRIC_DEVICE_INDEX` attribute.
The term tool widely refers to programs executed by the user or system administrator on a command line. Tools frequently interact with either the SMS, user applications, or both to perform administrative and support functions. For example, a debugger tool might be used to remotely control the processes of a parallel application, monitoring their behavior on a step-by-step basis. Historically, such tools were custom-written for each specific host environment due to the customized and/or proprietary nature of the environment’s interfaces.

The advent of PMIx offers the possibility for creating portable tools capable of interacting with multiple RMs without modification. Possible use-cases include:

- querying the status of scheduling queues and estimated allocation time for various resource options
- job submission and allocation requests
- querying job status for executing applications
- launching, monitoring, and debugging applications

Enabling these capabilities requires some extensions to the PMIx Standard (both in terms of APIs and attributes), and utilization of client-side APIs for more tool-oriented purposes.

This chapter defines specific APIs related to tools, provides tool developers with an overview of the support provided by PMIx, and serves to guide RM vendors regarding roles and responsibilities of RMs to support tools. As the number of tool-specific APIs and attributes is fairly small, the bulk of the chapter serves to provide a "theory of operation" for tools and debuggers. Description of the APIs themselves is therefore deferred to the Section 17.5 later in the chapter.

17.1 Connection Mechanisms

The key to supporting tools lies in providing mechanisms by which a tool can connect to a PMIx server. Application processes are able to connect because their local RM daemon provides them with the necessary contact information upon execution. A command-line tool, however, isn’t spawned by an RM daemon, and therefore lacks the information required for rendezvous with a PMIx server.

Once a tool has started, it initializes PMIx as a tool (via `PMIx_tool_init`) if its access is restricted to PMIx-based informational services such as `PMIx_Query_info`. However, if the
tool intends to start jobs, then it must include the `PMIX_LAUNCHER` attribute to inform the library of that intent so that the library can initialize and provide access to the corresponding support.

Support for tools requires that the PMIx server be initialized with an appropriate attribute indicating that tool connections are to be allowed. Separate attributes are provided to "fine-tune" this permission by allowing the environment to independently enable (or disable) connections from tools executing on nodes other than the one hosting the server itself. The PMIx server library shall provide an opportunity for the host environment to authenticate and approve each connection request from a specific tool by calling the `pmix_server_tool_connection_fn_t" hook" provided in the server module for that purpose. Servers in environments that do not provide this "hook" shall automatically reject all tool connection requests.

Tools can connect to any local or remote PMIx server provided they are either explicitly given the required connection information, or are able to discover it via one of several defined rendezvous protocols. Connection discovery centers around the existence of rendezvous files containing the necessary connection information, as illustrated in Fig. 17.1.

![Figure 17.1.: Tool rendezvous files](image)

The contents of each rendezvous file are specific to a given PMIx implementation, but should at least contain the namespace and rank of the server along with its connection URI. Note that tools linked to one PMIx implementation are therefore unlikely to successfully connect to PMIx server libraries from another implementation.

The top of the directory tree is defined by either the `PMIX_SYSTEM_TMPDIR` attribute (if given) or the `TMPDIR` environmental variable. PMIx servers that are designated as system servers by including the `PMIX_SERVER_SYSTEM_SUPPORT` attribute when calling `PMIx_server_init` will create a rendezvous file in this top-level directory. The filename will be of the form `pmix.sys.hostname`, where `hostname` is the string returned by the `gethostname` system call. Note that only one PMIx server on a node can be designated as the system server.

Non-system PMIx servers will create a set of three rendezvous files in the directory defined by either the `PMIX_SERVER_TMPDIR` attribute or the `TMPDIR` environmental variable:
• \texttt{pmix.host.tool.nspace} where \textit{host} is the string returned by the \texttt{gethostname} system call and \textit{nspace} is the namespace of the server.

• \texttt{pmix.host.tool.pid} where \textit{host} is the string returned by the \texttt{gethostname} system call and \textit{pid} is the PID of the server.

• \texttt{pmix.host.tool} where \textit{host} is the string returned by the \texttt{gethostname} system call. Note that servers which are not given a namespace-specific \texttt{PMIX\_SERVER\_TMPDIR} attribute may not generate this file due to conflicts should multiple servers be present on the node.

The files are identical and may be implemented as symlinks to a single instance. The individual file names are composed so as to aid the search process should a tool wish to connect to a server identified by its namespace or PID.

Servers will additionally provide a rendezvous file in any given location if the path (either absolute or relative) and filename is specified either during \texttt{PMIx\_server\_init} using the \texttt{PMIX\_LAUNCHER\_RENDZVOUS\_FILE} attribute, or by the \texttt{PMIX\_LAUNCHER\_RNDZ\_FILE} environmental variable prior to executing the process containing the server. This latter mechanism may be the preferred mechanism for tools such as debuggers that need to fork/exec a launcher (e.g., "mpiexec") and then rendezvous with it. This is described in more detail in Section 17.2.2.

Rendezvous file ownerships are set to the UID and GID of the server that created them, with permissions set according to the desires of the implementation and/or system administrator policy. All connection attempts are first governed by read access privileges to the target rendezvous file - thus, the combination of permissions, UID, and GID of the rendezvous files act as a first-level of security for tool access.

A tool may connect to as many servers at one time as the implementation supports, but is limited to designating only one such connection as its primary server. This is done to avoid confusion when the tool calls an API as to which server should service the request. The first server the tool connects to is automatically designated as the primary server.

Tools are allowed to change their primary server at any time via the \texttt{PMIx\_tool\_set\_server} API, and to connect/disconnect from a server as many times as desired. Note that standing requests (e.g., event registrations) with the current primary server may be lost and/or may not be transferred when transitioning to another primary server - PMIx implementors are not required to maintain or transfer state across tool-server connections.

Tool process identifiers are assigned by one of the following methods:

• If \texttt{PMIX\_TOOL\_NSPACE} is given, then the namespace of the tool will be assigned that value.
  – If \texttt{PMIX\_TOOL\_RANK} is also given, then the rank of the tool will be assigned that value.
  – If \texttt{PMIX\_TOOL\_RANK} is not given, then the rank will be set to a default value of zero.

• If a process ID is not provided and the tool connects to a server, then one will be assigned by the host environment upon connection to that server.
• If a process ID is not provided and the tool does not connect to a server (e.g., if `PMIX_TOOL_DO_NOT_CONNECT` is given), then the tool shall self-assign a unique identifier. This is often done using some combination involving hostname and PID.

Tool process identifiers remain constant across servers. Thus, it is critical that a system-wide unique namespace be provided if the tool itself sets the identifier, and that host environments provide a system-wide unique identifier in the case where the identifier is set by the server upon connection. The host environment is required to reject any connection request that fails to meet this criterion.

For simplicity, the following descriptions will refer to the:

- `PMIX_SYSTEM_TMPDIR` as the directory specified by either the `PMIX_SYSTEM_TMPDIR` attribute (if given) or the `TMPDIR` environmental variable.
- `PMIX_SERVER_TMPDIR` as the directory specified by either the `PMIX_SERVER_TMPDIR` attribute or the `TMPDIR` environmental variable.

The rendezvous methods are automatically employed for the initial tool connection during `PMIx_tool_init` unless the `PMIX_TOOL_DO_NOT_CONNECT` attribute is specified, and on all subsequent calls to `PMIx_tool_attach_to_server`.

### 17.1.1 Rendezvousing with a local server

Connection to a local PMIx server is pursued according to the following precedence chain based on attributes contained in the call to the `PMIx_tool_init` or `PMIx_tool_attach_to_server` APIs. Servers to which the tool already holds a connection will be ignored. Except where noted, the PMIx library will return an error if the specified file cannot be found, the caller lacks permissions to read it, or the server specified within the file does not respond to or accept the connection — the library will not proceed to check for other connection options as the user specified a particular one to use.

Note that the PMIx implementation may choose to introduce a "delayed connection" protocol between steps in the precedence chain - i.e., the library may cycle several times, checking for creation of the rendezvous file each time after a delay of some period of time, thereby allowing the tool to wait for the server to create the rendezvous file before either returning an error or continuing to the next step in the chain.

- If `PMIX_TOOL_ATTACHMENT_FILE` is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it. The format of the attachment file is identical to the rendezvous files described in earlier in this section. An error will be returned if the specified file cannot be found.

- If `PMIX_SERVER_URI` or `PMIX_TCP_URI` is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. `PMIX_SERVER_URI` is the preferred method as it is more generalized — `PMIX_TCP_URI` is provided for those cases where the user specifically wants to use a TCP transport for the connection and wants to error out if one isn’t available or cannot be used.
• If `PMIX_SERVER_PIDINFO` was provided, then the tool will search for a rendezvous file created by a PMIx server of the given PID in the `PMIX_SERVER_TMPDIR` directory. An error will be returned if a matching rendezvous file cannot be found.

• If `PMIX_SERVER_NSNAME` is given, then the tool will search for a rendezvous file created by a PMIx server of the given namespace in the `PMIX_SERVER_TMPDIR` directory. An error will be returned if a matching rendezvous file cannot be found.

• If `PMIX_CONNECT_TO_SYSTEM` is given, then the tool will search for a system-level rendezvous file created by a PMIx server in the `PMIX_SYSTEM_TMPDIR` directory. An error will be returned if a matching rendezvous file cannot be found.

• If `PMIX_CONNECT_SYSTEM_FIRST` is given, then the tool will look for a system-level rendezvous file created by a PMIx server in the `PMIX_SYSTEM_TMPDIR` directory. If found, then the tool will attempt to connect to it. In this case, no error will be returned if the rendezvous file is not found or connection is refused — the PMIx library will silently continue to the next option.

• By default, the tool will search the directory tree under the `PMIX_SERVER_TMPDIR` directory for rendezvous files of PMIx servers, attempting to connect to each it finds until one accepts the connection. If no rendezvous files are found, or all contacted servers refuse connection, then the PMIx library will return an error. No "delayed connection" protocols may be utilized at this point.

Note that there can be multiple local servers - one from the system plus others from launchers and active jobs. The PMIx tool connection search method is not guaranteed to pick a particular server unless directed to do so. Tools can obtain a list of servers available on their local node using the `PMIx_Query_info` APIs with the `PMIX_QUERY_AVAIL_SERVERS` key.

17.1.2 Connecting to a remote server

Connecting to remote servers is complicated due to the lack of access to the previously-described rendezvous files. Two methods are required to be supported, both based on the caller having explicit knowledge of either connection information or a path to a local file that contains such information:

• If `PMIX_TOOL_ATTACHMENT_FILE` is given, then the tool will attempt to read the specified file and connect to the server based on the information contained within it. The format of the attachment file is identical to the rendezvous files described in earlier in this section.

• If `PMIX_SERVER_URI` or `PMIX_TCP_URI` is given, then connection will be attempted to the server at the specified URI. Note that it is an error for both of these attributes to be specified. `PMIX_SERVER_URI` is the preferred method as it is more generalized — `PMIX_TCP_URI` is provided for those cases where the user specifically wants to use the TCP transport for the connection and wants to error out if it isn’t available or cannot be used.

Additional methods may be provided by particular PMIx implementations. For example, the tool may use `ssh` to launch a `probe` process onto the remote node so that the probe can search the `PMIX_SYSTEM_TMPDIR` and `PMIX_SERVER_TMPDIR` directories for rendezvous files,
relaying the discovered information back to the requesting tool. If sufficient information is found to allow for remote connection, then the tool can use it to establish the connection. Note that this method is not required to be supported - it is provided here as an example and left to the discretion of PMIx implementors.

17.1.3 Attaching to running jobs

When attaching to a running job, the tool must connect to a PMIx server that is associated with that job - e.g., a server residing in the host environment’s local daemon that spawned one or more of the job’s processes, or the server residing in the launcher that is overseeing the job. Identifying an appropriate server can sometimes prove challenging, particularly in an environment where multiple job launchers may be in operation, possibly under control of the same user.

In cases where the user has only the one job of interest in operation on the local node (e.g., when engaged in an interactive session on the node from which the launcher was executed), the normal rendezvous file discovery method can often be used to successfully connect to the target job, even in the presence of jobs executed by other users. The permissions and security authorizations can, in many cases, reliably ensure that only the one connection can be made. However, this is not guaranteed in all cases.

The most common method, therefore, for attaching to a running job is to specify either the PID of the job’s launcher or the namespace of the launcher’s job (note that the launcher’s namespace frequently differs from the namespace of the job it has launched). Unless the application processes themselves act as PMIx servers, connection must be to the servers in the daemons that oversee the application. This is typically either daemons specifically started by the job’s launcher process, or daemons belonging to the host environment, that are responsible for starting the application’s processes and oversee their execution.

Identifying the correct PID or namespace can be accomplished in a variety of ways, including:

- Using typical OS or host environment tools to obtain a listing of active jobs and perusing those to find the target launcher.
- Using a PMIx-based tool attached to a system-level server to query the active jobs and their command lines, thereby identifying the application of interest and its associated launcher.
- Manually recording the PID of the launcher upon starting the job.

Once the namespace and/or PID of the target server has been identified, either of the previous methods can be used to connect to it.

17.1.4 Tool initialization attributes

The following attributes are passed to the PMIx_tool_init API for use when initializing the PMIx library.

PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
Name of the namespace to use for this tool.

- **PMIX_TOOL_RANK** "pmix.tool.rank" (uint32_t)
  
  Rank of this tool.

- **PMIX_LAUNCHER** "pmix.tool.launcher" (bool)
  
  Tool is a launcher and needs to create rendezvous files.

### 17.1.5 Tool initialization environmental variables

The following environmental variables are used during `PMIx_tool_init` and `PMIx_server_init` to control various rendezvous-related operations when the process is started manually (e.g., on a command line) or by a fork/exec-like operation.

- **PMIX_LAUNCHER_RNDZ_URI**
  
  The spawned tool is to be connected back to the spawning tool using the given URI so that the spawning tool can provide directives (e.g., a `PMIx_Spawn` command) to it.

- **PMIX_LAUNCHER_RNDZ_FILE**
  
  If the specified file does not exist, this variable contains the absolute path of the file where the spawned tool is to store its connection information so that the spawning tool can connect to it. If the file does exist, it contains the information specifying the server to which the spawned tool is to connect.

- **PMIX_KEEPALIVE_PIPE**
  
  An integer read-end of a POSIX pipe that the tool should monitor for closure, thereby indicating that the parent tool has terminated. Used, for example, when a tool fork/exec’s an intermediate launcher that should self-terminate if the originating tool exits.

Note that these environmental variables should be cleared from the environment after use and prior to forking child processes to avoid potentially unexpected behavior by the child processes.

### 17.1.6 Tool connection attributes

These attributes are defined to assist PMIx-enabled tools to connect with a PMIx server by passing them into either the `PMIx_tool_init` or the `PMIx_tool_attach_to_server` APIs - thus, they are not typically accessed via the `PMIx_Get` API.

- **PMIX_SERVER_PIDINFO** "pmix.srvr.pidinfo" (pid_t)
  
  PID of the target PMIx server for a tool.

- **PMIX_CONNECT_TO_SYSTEM** "pmix.cnct.sys" (bool)
  
  The requester requires that a connection be made only to a local, system-level PMIx server.

- **PMIX_CONNECT_SYSTEM_FIRST** "pmix.cnct.sys.first" (bool)
  
  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_HOSTNAME** "pmix.srvr.host" (char*)
  
  Host where target PMIx server is located.

- **PMIX_CONNECT_MAX_RETRIES** "pmix.tool.mretries" (uint32_t)
Maximum number of times to try to connect to PMIx server - the default value is implementation specific.

PMIX_CONNECT_RETRY_DELAY "pmix.tool.retry" (uint32_t)
Time in seconds between connection attempts to a PMIx server - the default value is implementation specific.

PMIX_TOOL_DO_NOT_CONNECT "pmix.tool.nocon" (bool)
The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

PMIX_TOOL_CONNECT_OPTIONAL "pmix.tool.conopt" (bool)
The tool shall connect to a server if available, but otherwise continue to operate unconnected.

PMIX_TOOL_ATTACHMENT_FILE "pmix.tool.attach" (char*)
Pathname of file containing connection information to be used for attaching to a specific server.

PMIX_LAUNCHER_RENDEZVOUS_FILE "pmix.tool.lncrnd" (char*)
Pathname of file where the launcher is to store its connection information so that the spawning tool can connect to it.

PMIX_PRIMARY_SERVER "pmix.pri.srvr" (bool)
The server to which the tool is connecting shall be designated the primary server once connection has been accomplished.

PMIX_WAIT_FOR_CONNECTION "pmix.wait.conn" (bool)
Wait until the specified process has connected to the requesting tool or server, or the operation times out (if the PMIX_TIMEOUT directive is included in the request).

17.2 Launching Applications with Tools

Tool-directed launches require that the tool include the PMIX_LAUNCHER attribute when calling PMIx_tool_init. Two launch modes are supported:

- **Direct launch** where the tool itself is directly responsible for launching all processes, including debugger daemons, using either the RM or daemons launched by the tool – i.e., there is no intermediate launcher (IL) such as mpiexec. The case where the tool is self-contained (i.e., uses its own daemons without interacting with an external entity such as the RM) lies outside the scope of this Standard; and

- **Indirect launch** where all processes are started via an IL such as mpiexec and the tool itself is not directly involved in launching application processes or debugger daemons. Note that the IL may utilize the RM to launch processes and/or daemons under the tool’s direction.

Either of these methods can be executed interactively or by a batch script. Note that not all host environments may support the direct launch method.
17.2.1 Direct launch

In the direct-launch use-case (Fig. 17.2), the tool itself performs the role of the launcher. Once invoked, the tool connects to an appropriate PMIx server - e.g., a system-level server hosted by the RM. The tool is responsible for assembling the description of the application to be launched (e.g., by parsing its command line) into a spawn request containing an array of pmix_app_t applications and pmix_info_t job-level information. An allocation of resources may or may not have been made in advance – if not, then the spawn request must include allocation request information.

In addition to the attributes described in PMIx_Spawn, the tool may optionally wish to include the following tool-specific attributes in the job_info argument to that API (the debugger-related attributes are discussed in more detail in Section 17.4):

- **PMIX_FWD_STDIN** "pmix.fwd.stdin" (pmix_rank_t)
  The requester intends to push information from its stdin to the indicated process. The local spawn agent should, therefore, ensure that the stdin channel to that process remains available. A rank of PMIX_RANK_WILDCARD indicates that all processes in the spawned job are potential recipients. The requester will issue a call to PMIx_IOF_push to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

- **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)
  Requests that the ability to forward the stdout of the spawned processes be maintained. The requester will issue a call to PMIx_IOF_pull to specify the callback function and other options for delivery of the forwarded output.
• **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)
  Requests that the ability to forward the **stderr** of the spawned processes be maintained.
  The requester will issue a call to **PMIx_IOF_pull** to specify the callback function and
  other options for delivery of the forwarded output.

• **PMIX_FWD_STD_DIAG** "pmix.fwd.stddiag" (bool)
  Requests that the ability to forward the diagnostic channel (if it exists) of the spawned
  processes be maintained. The requester will issue a call to **PMIx_IOF_pull** to specify
  the callback function and other options for delivery of the forwarded output.

• **PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (uint32_t)
  The requested size of the PMIx server cache in bytes for each specified channel. By
default, the server is allowed (but not required) to drop all bytes received beyond the max
  size.

• **PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (bool)
  In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the
  cache.

• **PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (bool)
  In an overflow situation, the PMIx server is to drop any new bytes received until room
  becomes available in the cache (default).

• **PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (uint32_t)
  Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until
  the specified number of bytes is collected to avoid being called every time a block of IO
  arrives. The PMIx tool library will execute the callback and reset the collection counter
  whenever the specified number of bytes becomes available. Any remaining buffered data
  will be **flushed** to the callback upon a call to deregister the respective channel.

• **PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (uint32_t)
  Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
  size, this prevents IO from being held indefinitely while waiting for another payload to
  arrive.

• **PMIX_IOF_TAG_OUTPUT** "pmix.iof.tag" (bool)
  Requests that output be prefixed with the nspace,rank of the source and a string
  identifying the channel (**stdout, stderr**, etc.).

• **PMIX_IOF_TIMESTAMP_OUTPUT** "pmix.iof.ts" (bool)
  Requests that output be marked with the time at which the data was received by the tool -
  note that this will differ from the time at which the data was collected from the source.

• **PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)
  Requests that output be formatted in XML.

• **PMIX_NOHUP** "pmix.nohup" (bool)
Any processes started on behalf of the calling tool (or the specified namespace, if such specification is included in the list of attributes) should continue after the tool disconnects from its server.

- **PMIX_NOTIFY_JOB_EVENTS** "pmix.note.jev" (bool)
  Requests that the launcher generate the PMIX_EVENT_JOB_START, PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events. Each event is to include at least the namespace of the corresponding job and a PMIX_EVENT_TIMESTAMP indicating the time the event occurred. Note that the requester must register for these individual events, or capture and process them by registering a default event handler instead of individual handlers and then process the events based on the returned status code. Another common method is to register one event handler for all job-related events, with a separate handler for non-job events - see PMIx_Register_event_handler for details.

- **PMIX_NOTIFY_COMPLETION** "pmix.notecomp" (bool)
  Requests that the launcher generate the PMIX_EVENT_JOB_END event for normal or abnormal termination of the spawned job. The event shall include the returned status code (PMIX_JOB_TERM_STATUS) for the corresponding job; the identity (PMIX_PROCID) and exit status (PMIX_EXIT_CODE) of the first failed process, if applicable; and a PMIX_EVENT_TIMESTAMP indicating the time the termination occurred. Note that the requester must register for the event or capture and process it within a default event handler.

- **PMIX_LOG_JOB_EVENTS** "pmix.log.jev" (bool)
  Requests that the launcher log the PMIX_EVENT_JOB_START, PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events using PMIx_Log, subject to the logging attributes of Section 12.4.3.

- **PMIX_LOG_COMPLETION** "pmix.logcomp" (bool)
  Requests that the launcher log the PMIX_EVENT_JOB_END event for normal or abnormal termination of the spawned job using PMIx_Log, subject to the logging attributes of Section 12.4.3. The event shall include the returned status code (PMIX_JOB_TERM_STATUS) for the corresponding job; the identity (PMIX_PROCID) and exit status (PMIX_EXIT_CODE) of the first failed process, if applicable; and a PMIX_EVENT_TIMESTAMP indicating the time the termination occurred.

- **PMIX_DEBUG_STOP_ON_EXEC** "pmix.dbg.exec" (bool)
  Included in either the pmix_info_t array in a pmix_app_t description (if the directive applies only to that application) or in the job_info array if it applies to all applications in the given spawn request. Indicates that the application is being spawned under a debugger, and that the local launch agent is to pause the resulting application processes on first instruction for debugger attach. The launcher (RM or IL) is to generate the PMIX_LAUNCH_COMPLETE event when all processes are stopped at the exec point.

- **PMIX_DEBUG_STOP_IN_INIT** "pmix.dbg.init" (bool)
Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The PMIx client library in each resulting application process shall notify its PMIx server that it is pausing and then pause during `PMIx_Init` of the spawned processes until either released by debugger modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have reached the pause point.

- **PMIX_DEBUG_WAIT_FOR_NOTIFY** "pmix.dbg.notify" (bool)

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The resulting application processes are to notify their server (by generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some application-determined location and pause at that point until either released by debugger modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they are at the pause point.

The tool then calls the `PMIx_Spawn` API so that the PMIx library can communicate the spawn request to the server.

Upon receipt, the PMIx server library passes the spawn request to its host RM daemon for processing via the `pmix_server_spawn_fn_t` server module function. If this callback was not provided, then the PMIx server library will return the `PMIX_ERR_NOT_SUPPORTED` error status.

If an allocation must be made, then the host environment is responsible for communicating the request to its associated scheduler. Once resources are available, the host environment initiates the launch process to start the job. The host environment must parse the spawn request for relevant directives, returning an error if any required directive cannot be supported. Optional directives may be ignored if they cannot be supported.

Any error while executing the spawn request must be returned by `PMIx_Spawn` to the requester. Once the spawn request has succeeded in starting the specified processes, the request will return `PMIX_SUCCESS` back to the requester along with the namespace of the started job. Upon termination of the spawned job, the host environment must generate a `PMIX_EVENT_JOB_END` event for normal or abnormal termination if requested to do so. The event shall include:

- the returned status code (`PMIX_JOB_TERM_STATUS`) for the corresponding job;
- the identity (`PMIX_PROCID`) and exit status (`PMIX_EXIT_CODE`) of the first failed process, if applicable;
- a `PMIX_EVENT_TIMESTAMP` indicating the time the termination occurred; plus
17.2.2 Indirect launch

In the indirect launch use-case, the application processes are started via an intermediate launcher (e.g., mpiexec) that is itself started by the tool (see Fig 17.3). Thus, at a high level, this is a two-stage launch procedure to start the application: the tool (henceforth referred to as the *initiator*) starts the IL, which then starts the applications. In practice, additional steps may be involved if, for example, the IL starts its own daemons to shepherd the application processes.

A key aspect of this operational mode is the avoidance of any requirement that the initiator parse and/or understand the command line of the IL. Instead, the indirect launch procedure supports either of two methods: one where the initiator assumes responsibility for parsing its command line to obtain the application as well as the IL and its options, and another where the initiator defers the command line parsing to the IL. Both of these methods are described in the following sections.

17.2.2.1 Initiator-based command line parsing

This method utilizes a first call to the `PMIx_Spawn` API to start the IL itself, and then uses a second call to `PMIx_Spawn` to request that the IL spawn the actual job. The burden of analyzing the initial command line to separately identify the IL’s command line from the application itself falls upon the initiator. An example is provided below:

```
$ initiator --launcher "mpiexec --verbose" -n 3 ./app <appoptions>
```

The initiator spawns the IL using the same procedure for launching an application - it begins by assembling the description of the IL into a spawn request containing an array of `pmix_app_t` and `pmix_info_t` job-level information. Note that this step does not include any information regarding the application itself - only the launcher is included. In addition, the initiator must include the rendezvous URI in the environment so the IL knows how to connect back to it.

An allocation of resources for the IL itself may or may not be required – if it is, then the allocation must be made in advance or the spawn request must include allocation request information.

The initiator may optionally wish to include the following tool-specific attributes in the `job_info` argument to `PMIx_Spawn` - note that these attributes refer only to the behavior of the IL itself and not the eventual job to be launched:

- **PMIX_FWD_STDIN** "pmix.fwd.stdin" (pmix_rank_t)
  
  The requester intends to push information from its stdin to the indicated process. The local spawn agent should, therefore, ensure that the stdin channel to that process remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the spawned job are potential recipients. The requester will issue a call to `PMIx_IOF_push` to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

- **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)
Requests that the ability to forward the \texttt{stdout} of the spawned processes be maintained. The requester will issue a call to \texttt{PMIx\_IOF\_pull} to specify the callback function and other options for delivery of the forwarded output.

- \texttt{PMIX\_FWD\_STDERR} "pmix.fwd.stderr" (bool)
  Requests that the ability to forward the \texttt{stderr} of the spawned processes be maintained. The requester will issue a call to \texttt{PMIx\_IOF\_pull} to specify the callback function and other options for delivery of the forwarded output.

- \texttt{PMIX\_FWD\_STDDIA} "pmix.fwd.stddiag" (bool)
  Requests that the ability to forward the diagnostic channel (if it exists) of the spawned processes be maintained. The requester will issue a call to \texttt{PMIx\_IOF\_pull} to specify the callback function and other options for delivery of the forwarded output.

- \texttt{PMIX\_IOF\_CACHE\_SIZE} "pmix.iof.csize" (uint32\_t)
  The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

- \texttt{PMIX\_IOF\_DROP\_OLDEST} "pmix.iof.old" (bool)
  In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.

- \texttt{PMIX\_IOF\_DROP\_NEWEST} "pmix.iof.new" (bool)
  In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).

- \texttt{PMIX\_IOF\_BUFFERING\_SIZE} "pmix.iof.bsize" (uint32\_t)
  Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter.
whenever the specified number of bytes becomes available. Any remaining buffered data
will be flushed to the callback upon a call to deregister the respective channel.

- **PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (uint32_t)
  Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
  size, this prevents IO from being held indefinitely while waiting for another payload to
  arrive.

- **PMIX_IOF_TAG_OUTPUT** "pmix.iof.tag" (bool)
  Requests that output be prefixed with the nspace,rank of the source and a string
  identifying the channel (stdout, stderr, etc.).

- **PMIX_IOF_TIMESTAMP_OUTPUT** "pmix.iof.ts" (bool)
  Requests that output be marked with the time at which the data was received by the tool -
  note that this will differ from the time at which the data was collected from the source.

- **PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)
  Requests that output be formatted in XML.

- **PMIX_NOHUP** "pmix.nohup" (bool)
  Any processes started on behalf of the calling tool (or the specified namespace, if such
  specification is included in the list of attributes) should continue after the tool disconnects
  from its server.

- **PMIX_LAUNCHER_DAEMON** "pmix.lnch.dmn" (char*)
  Path to executable that is to be used as the backend daemon for the launcher. This replaces
  the launcher’s own daemon with the specified executable. Note that the user is therefore
  responsible for ensuring compatibility of the specified executable and the host launcher.

- **PMIX_FORKEXEC_AGENT** "pmix.frkex.agnt" (char*)
  Path to executable that the launcher’s backend daemons are to fork/exec in place of the
  actual application processes. The fork/exec agent shall connect back (as a PMIx tool) to
  the launcher’s daemon to receive its spawn instructions, and is responsible for starting the
  actual application process it replaced. See Section 17.4.3 for details.

- **PMIX_EXEC_AGENT** "pmix.exec.agnt" (char*)
  Path to executable that the launcher’s backend daemons are to fork/exec in place of the
  actual application processes. The launcher’s daemon shall pass the full command line of
  the application on the command line of the exec agent, which shall not connect back to the
  launcher’s daemon. The exec agent is responsible for exec’ing the specified application
  process in its own place. See Section 17.4.3 for details.

- **PMIX_DEBUG_STOP_IN_INIT** "pmix.dbg.init" (bool)
  Included in either the pmix_info_t array in a pmix_app_t description (if the
directive applies only to that application) or in the job_info array if it applies to all
applications in the given spawn request. Indicates that the specified application is being
spawned under a debugger. The PMIx client library in each resulting application process
shall notify its PMIx server that it is pausing and then pause during PMIx_Init of the
spawned processes until either released by debugger modification of an appropriate variable or receipt of the PMIX_DEBUGGER_RELEASE event. The launcher (RM or IL) is responsible for generating the PMIX_DEBUG_WAITING_FOR_NOTIFY event when all processes have reached the pause point. In this context, the initiator is directing the IL to stop in PMIx_tool_init. This gives the initiator a chance to connect to the IL and register for events prior to the IL launching the application job.

and the following optional variables in the environment of the IL:

- PMIX_KEEPALIVEPIPE - an integer read-end of a POSIX pipe that the IL should monitor for closure, thereby indicating that the initiator has terminated.

The initiator then calls the PMIx_Spawn API so that the PMIx library can either communicate the spawn request to a server (if connected to one), or locally spawn the IL itself if not connected to a server and the PMIx implementation includes self-spawn support. PMIx_Spawn shall return an error if neither of these conditions is met.

When initialized by the IL, the PMIx_tool_init function must perform two operations:

- check for the presence of the PMIX_KEEPALIVE_PIPE environmental variable - if provided, then the library shall monitor the pipe for closure, providing a PMIX_EVENT_JOB_END event when the pipe closes (thereby indicating the termination of the initiator). The IL should register for this event after completing PMIx_tool_init - the initiator’s namespace can be obtained via a call to PMIx_Get with the PMIX_PARENT_ID key. Note that this feature will only be available if the spawned IL is local to the initiator.

- check for the PMIX_LAUNCHER_RNDZ_URI environmental parameter - if found, the library shall connect back to the initiator using the PMIx_tool_attach_to_server API, retaining its current server as its primary server.

Once the IL completes PMIx_tool_init, it must register for the PMIX_EVENT_JOB_END termination event and then idle until receiving that event - either directly from the initiator, or from the PMIx library upon detecting closure of the keepalive pipe. The IL idles in the intervening time as it is solely acting as a relay (if connected to a server that is performing the actual application launch) or as a PMIx server responding to spawn requests.

Upon return from the PMIx_Spawn API, the initiator should set the spawned IL as its primary server using the PMIx_tool_set_server API with the nspace returned by PMIx_Spawn and any valid rank (a rank of zero would ordinarily be used as only one IL process is typically started). It is advisable to set a connection timeout value when calling this function. The initiator can then proceed to spawn the actual application according to the procedure described in Section 17.2.1.

17.2.2.2 IL-based command line parsing

In the case where the initiator cannot parse its command line, it must defer that parsing to the IL. A common example is provided below:

$ initiator mpiexec --verbose -n 3 ./app <appoptions>
For this situation, the initiator proceeds as above with only one notable exception: instead of calling 
\texttt{PMIx\_Spawn} twice (once to start the IL and again to start the actual application), the initiator only 
calls that API one time:

- The \textit{app} parameter passed to the spawn request contains only one \texttt{pmix\_app\_t} that contains 
  the entire command line, including both launcher and application(s).
- The launcher executable must be in the \textit{app.cmd} field and in \textit{app.argv[0]}, with the rest of the 
  command line appended to the \textit{app.argv} array.
- Any job-level directives for the IL itself (e.g., \texttt{PMIX\_FORKEXEC\_AGENT} or 
  \texttt{PMIX\_FWD\_STDOUT}) are included in the \textit{job\_info} parameter of the call to \texttt{PMIx\_Spawn}.
- The job-level directives must include both the \texttt{PMIX\_SPAWN\_TOOl} attribute indicating that the 
  initiator is spawning a tool, and the \texttt{PMIX\_DEBUG\_STOP\_IN\_INIT} attribute directing the IL 
  to stop during the call to \texttt{PMIx\_tool\_init}. The latter directive allows the initiator to connect 
  to the IL prior to launch of the application.
- The \texttt{PMIX\_LAUNCHER\_RNDZ\_URI} and \texttt{PMIX\_KEEPALIVE\_PIPE} environmental variables 
  are provided to the launcher in its environment via the \textit{app.env} field.
- The IL must use \texttt{PMIx\_Get} with the \texttt{PMIX\_LAUNCH\_DIRECTIVES} key to obtain any 
  initiator-provided directives (e.g., \texttt{PMIX\_DEBUG\_STOP\_IN\_INIT} or 
  \texttt{PMIX\_DEBUG\_STOP\_ON\_EXEC}) aimed at the application(s) it will spawn.

Upon return from \texttt{PMIx\_Spawn}, the initiator must:

- use the \texttt{PMIx\_tool\_set\_server} API to set the spawned IL as its primary server
- register with that server to receive the \texttt{PMIX\_LAUNCH\_COMPLETE} event. This allows the 
  initiator to know when the IL has completed launch of the application
- release the IL from its "hold" in \texttt{PMIx\_tool\_init} by issuing the 
  \texttt{PMIX\_DEBUGGER\_RELEASE} event, specifying the IL as the custom range. Upon receipt of the 
  event, the IL is free to parse its command line, apply any provided directives, and execute the 
  application.

Upon receipt of the \texttt{PMIX\_LAUNCH\_COMPLETE} event, the initiator should register to receive 
notification of completion of the returned namespace of the application. Receipt of the 
\texttt{PMIX\_EVENT\_JOB\_END} event provides a signal that the initiator may itself terminate.

\subsection*{17.2.3 Tool spawn-related attributes}

Tools are free to utilize the spawn attributes available to applications (see 11.2.4) when 
constructing a spawn request, but can also utilize the following attributes that are specific to 
tool-based spawn operations:

\texttt{PMIX\_FWD\_STDIN} "pmix.fwd.stdin" (pmix\_rank\_t)
The requester intends to push information from its stdin to the indicated process. The local spawn agent should, therefore, ensure that the stdin channel to that process remains available. A rank of PMIX_RANK_WILDCARD indicates that all processes in the spawned job are potential recipients. The requester will issue a call to PMIx_IOF_push to initiate the actual forwarding of information to specified targets - this attribute simply requests that the IL retain the ability to forward the information to the designated targets.

PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
Requests that the ability to forward the stdout of the spawned processes be maintained. The requester will issue a call to PMIx_IOF_pull to specify the callback function and other options for delivery of the forwarded output.

PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
Requests that the ability to forward the stderr of the spawned processes be maintained. The requester will issue a call to PMIx_IOF_pull to specify the callback function and other options for delivery of the forwarded output.

PMIX_FWD_STDDIAG "pmix.fwd.stddiag" (bool)
Requests that the ability to forward the diagnostic channel (if it exists) of the spawned processes be maintained. The requester will issue a call to PMIx_IOF_pull to specify the callback function and other options for delivery of the forwarded output.

PMIX_NOHUP "pmix.nohup" (bool)
Any processes started on behalf of the calling tool (or the specified namespace, if such specification is included in the list of attributes) should continue after the tool disconnects from its server.

PMIX_LAUNCHER_DAEMON "pmix.lnch.dmn" (char*)
Path to executable that is to be used as the backend daemon for the launcher. This replaces the launcher’s own daemon with the specified executable. Note that the user is therefore responsible for ensuring compatibility of the specified executable and the host launcher.

PMIX_FORKEXEC_AGENT "pmix.frkex.agnt" (char*)
Path to executable that the launcher’s backend daemons are to fork/exec in place of the actual application processes. The fork/exec agent shall connect back (as a PMix tool) to the launcher’s daemon to receive its spawn instructions, and is responsible for starting the actual application process it replaced. See Section 17.4.3 for details.

PMIX_EXEC_AGENT "pmix.exec.agnt" (char*)
Path to executable that the launcher’s backend daemons are to fork/exec in place of the actual application processes. The launcher’s daemon shall pass the full command line of the application on the command line of the exec agent, which shall not connect back to the launcher’s daemon. The exec agent is responsible for exec’ing the specified application process in its own place. See Section 17.4.3 for details.

PMIX_LAUNCH_DIRECTIVES "pmix.lnch.dirs" (pmix_data_array_t*)
Array of pmix_info_t containing directives for the launcher - a convenience attribute for retrieving all directives with a single call to PMIx_Get.

17.2.4 Tool rendezvous-related events

The following constants refer to events relating to rendezvous of a tool and launcher during spawn of the IL.
PMIX_LAUNCHER_READY An application launcher (e.g., mpiexec) shall generate this event to signal a tool that started it that the launcher is ready to receive directives/commands (e.g., PMIx_Spawn). This is only used when the initiator is able to parse the command line itself, or the launcher is started as a persistent Distributed Virtual Machine (DVM).

17.3 IO Forwarding

Underlying the operation of many tools is a common need to forward stdin from the tool to targeted processes, and to return stdout/stderr from those processes to the tool (e.g., for display on the user’s console). Historically, each tool developer was responsible for creating their own IO forwarding subsystem. However, the introduction of PMIx as a standard mechanism for interacting between applications and the host environment has made it possible to relieve tool developers of this burden.

This section defines functions by which tools can request forwarding of input/output to/from other processes and serves as a design guide to:

- provide tool developers with an overview of the expected behavior of the PMIx IO forwarding support;
- guide RM vendors regarding roles and responsibilities expected of the RM to support IO forwarding; and
- provide insight into the thinking of the PMIx community behind the definition of the PMIx IO forwarding APIs.

Note that 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.

The responsibility of the host environment in forwarding of IO falls into the following areas:

- Capturing output from specified 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 to the requesting tool.

It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the requesting client. This may require caching of output until a forwarding registration is received, as governed by the corresponding IO forwarding attributes of Section 17.3.5 that are supported by the implementation.

17.3.1 Forwarding stdout/stderr

At an appropriate point in its operation (usually during startup), a tool will utilize the PMIx_tool_init function to connect to a PMIx server. The PMIx server can be hosted by an
RM daemon or could be embedded in a library-provided starter program such as mpiexec - in terms of IO forwarding, the operations remain the same either way. For purposes of this discussion, we will assume the server is in an RM daemon and that the application processes are directly launched by the RM, as shown in Fig 17.4.

![Figure 17.4.: Forwarding stdout/stderr](image)

Once the tool has connected to the target server, it can request that processes be spawned on its behalf or that output from a specified set of existing processes in a given executing application be forwarded to it. Requests to spawn processes should include the `PMIX_FWD_STDIN`, `PMIX_FWD_STDOUT`, and/or `PMIX_FWD_STDERR` attributes if the tool intends to request that the corresponding streams be forwarded at some point during execution.

Note that requests to capture output from existing processes via the `PMIx_IOF_pull` API, and/or to forward input to specified processes via the `PMIX_IOF_push` API, can only succeed if the required attributes to retain that ability were passed when the corresponding job was spawned. The host is required to return an error for all such requests in cases where this condition is not met.

Two modes are supported when requesting that the host forward standard output/error via the `PMIx_IOF_pull` API - these can be controlled by including one of the following attributes in the `info` array passed to that function:

- **PMIX_IOF_COPY** "pmix.iof.cpy" (bool)
  Requests that the host environment deliver a copy of the specified output stream(s) to the tool, letting the stream(s) continue to also be delivered to the default location. This allows the tool to tap into the output stream(s) without redirecting it from its current final destination.

- **PMIX_IOF_REDIRECT** "pmix.iof.redir" (bool)
Requests that the host environment intercept the specified output stream(s) and deliver it to the requesting tool instead of its current final destination. This might be used, for example, during a debugging procedure to avoid injection of debugger-related output into the application’s results file. The original output stream(s) destination is restored upon termination of the tool. This is the default mode of operation.

When requesting to forward `stdout/stderr`, the tool can specify several formatting options to be used on the resulting output stream. These include:

- **PMIX_IOF_TAG_OUTPUT** "pmix.iof.tag" (bool)
  Requests that output be prefixed with the nspace, rank of the source and a string identifying the channel (`stdout`, `stderr`, etc.).

- **PMIX_IOF_TIMESTAMP_OUTPUT** "pmix.iof.ts" (bool)
  Requests that output be marked with the time at which the data was received by the tool - note that this will differ from the time at which the data was collected from the source.

- **PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)
  Requests that output be formatted in XML.

The PMIx client in the tool is responsible for formatting the output stream. Note that output from multiple processes will often be interleaved due to variations in arrival time - ordering of output is not guaranteed across processes and/or nodes.

### 17.3.2 Forwarding stdin

A tool is not necessarily a child of the RM as it may have been started directly from the command line. Thus, provision must be made for the tool to collect its `stdin` and pass it to the host RM (via the PMIx server) for forwarding. Two methods of support for forwarding of `stdin` are defined:

- **internal collection** by the PMIx tool library itself. This is requested via the `PMIX_IOF_PUSH_STDIN` attribute in the `PMIx_IOF_push` call. When this mode is selected, the tool library begins collecting all `stdin` data and internally passing it to the local server for distribution to the specified target processes. All collected data is sent to the same targets until `stdin` is closed, or a subsequent call to `PMIx_IOF_push` is made that includes the `PMIX_IOF_COMPLETE` attribute indicating that forwarding of `stdin` is to be terminated.

- **external collection** directly by the tool. It is assumed that the tool will provide its own code/mode for collecting its `stdin` as the tool developers may choose to insert some filtering and/or editing of the stream prior to forwarding it. In addition, the tool can directly control the targets for the data on a per-call basis – i.e., each call to `PMIx_IOF_push` can specify its own set of target recipients for that particular `blob` of data. Thus, this method provides maximum flexibility, but requires that the tool developer provide their own code to capture `stdin`. 
Note that it is the responsibility of the RM to forward data to the host where the target process(es) are executing, and for the host daemon on that node to deliver the data to the `stdin` of target process(es). The PMIx server on the remote node is not involved in this process. Systems that do not support forwarding of `stdin` shall return `PMIX_ERR_NOT_SUPPORTED` in response to a forwarding request.

**Advice to users**

Scalable forwarding of `stdin` represents a significant challenge. Most environments will at least handle a `send-to-1` model whereby `stdin` is forwarded to a single identified process, and occasionally an additional `send-to-all` model where `stdin` is forwarded to all processes in the application. Users are advised to check their host environment for available support as the distribution method lies outside the scope of PMIx.

`Stdin` buffering by the RM and/or PMIx library can be problematic. If any targeted recipient is slow reading data (or decides never to read data), then the data must be buffered in some intermediate daemon or the PMIx tool library itself. Thus, piping a large amount of data into `stdin` can result in a very large memory footprint in the system management stack or the tool. Best practices, therefore, typically focus on reading of input files by application processes as opposed to forwarding of `stdin`.

---

**Figure 17.5.: Forwarding stdin**

---
17.3.3 IO Forwarding Channels

PMIx v3.0  The `pmix_iof_channel_t` structure is a `uint16_t` type that defines a set of bit-mask flags for specifying IO forwarding channels. These can be bitwise OR'd together to reference multiple channels.

- `PMIX_FWD_NO_CHANNELS`  Forward no channels.
- `PMIX_FWD_STDIN_CHANNEL`  Forward stdin.
- `PMIX_FWD_STDOUT_CHANNEL`  Forward stdout.
- `PMIX_FWD_STDERR_CHANNEL`  Forward stderr.
- `PMIX_FWD_STDDIAG_CHANNEL`  Forward stddiag, if available.
- `PMIX_FWD_ALL_CHANNELS`  Forward all available channels.

17.3.4 IO Forwarding constants

- `PMIX_ERR_IOF_FAILURE`  An IO forwarding operation failed - the affected channel will be included in the notification.
- `PMIX_ERR_IOF_COMPLETE`  IO forwarding of the standard input for this process has completed - i.e., the stdin file descriptor has closed.

17.3.5 IO Forwarding attributes

The following attributes are used to control IO forwarding behavior at the request of tools. Use of the attributes is optional - any option not provided will revert to some implementation-specific value.

- `PMIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t)`  The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.
- `PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)`  In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.
- `PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)`  In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).
- `PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)`  Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be flushed to the callback upon a call to deregister the respective channel.
- `PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)`  Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
- `PMIX_IOF_COMPLETE "pmix.iof.cmp" (bool)`
Indicates that the specified IO channel has been closed by the source.

**PMIX_IOF_TAG_OUTPUT** "pmix.iof.tag" (bool)
Requests that output be prefixed with the nspace, rank of the source and a string identifying the channel (stdout, stderr, etc.).

**PMIX_IOF_TIMESTAMP_OUTPUT** "pmix.iof.ts" (bool)
Requests that output be marked with the time at which the data was received by the tool - note that this will differ from the time at which the data was collected from the source.

**PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)
Requests that output be formatted in XML.

**PMIX_IOF_PUSH_STDIN** "pmix.iof.stdin" (bool)
Requests that the PMIx library collect the stdin of the requester and forward it to the processes specified in the PMIx_IOF_push call. All collected data is sent to the same targets until stdin is closed, or a subsequent call to PMIx_IOF_push is made that includes the PMIX_IOF_COMPLETE attribute indicating that forwarding of stdin is to be terminated.

**PMIX_IOF_COPY** "pmix.iof.cpy" (bool)
Requests that the host environment deliver a copy of the specified output stream(s) to the tool, letting the stream(s) continue to also be delivered to the default location. This allows the tool to tap into the output stream(s) without redirecting it from its current final destination.

**PMIX_IOF_REDIRECT** "pmix.iof.redir" (bool)
Requests that the host environment intercept the specified output stream(s) and deliver it to the requesting tool instead of its current final destination. This might be used, for example, during a debugging procedure to avoid injection of debugger-related output into the application’s results file. The original output stream(s) destination is restored upon termination of the tool.

### 17.4 Debugger Support

Debuggers are a class of tool that merits special consideration due to their particular requirements for access to job-related information and control over process execution. The primary advantage of using PMIx for these purposes lies in the resulting portability of the debugger as it can be used with any system and/or programming model that supports PMIx. In addition to the general tool support described above, debugger support includes:

- Co-location, co-spawn, and communication wireup of debugger daemons for scalable launch. This includes providing debugger daemons with endpoint connection information across the daemons themselves.

- Identification of the job that is to be debugged. This includes automatically providing debugger daemons with the job-level information for their target job.

Debuggers can also utilize the options in the PMIx_Spawn API to exercise a degree of control over spawned jobs for debugging purposes. For example, a debugger can utilize the environmental parameter attributes of Section 11.2.4 to request LD_PRELOAD of a memory interceptor library...
prior to spawning an application process, or interject a custom fork/exec agent to shepherd the application process.

A key element of the debugging process is the ability of the debugger to require that processes pause at some well-defined point, thereby providing the debugger with an opportunity to attach and control execution. The actual implementation of the pause lies outside the scope of PMIx - it typically requires either the launcher or the application itself to implement the necessary operations. However, PMIx does provide several standard attributes by which the debugger can specify the desired attach point:

- **PMIX_DEBUG_STOP_ON_EXEC** "pmix.dbg.exec" (bool)
  
  Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the application is being spawned under a debugger, and that the local launch agent is to pause the resulting application processes on first instruction for debugger attach. The launcher (RM or IL) is to generate the `PMIX_LAUNCH_COMPLETE` event when all processes are stopped at the exec point. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

- **PMIX_DEBUG_STOP_IN_INIT** "pmix.dbg.init" (bool)
  
  Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The PMIx client library in each resulting application process shall notify its PMIx server that it is pausing and then pause during `PMIx_Init` of the spawned processes until either released by debugger modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have reached the pause point. PMIx implementations that do not support this operation shall return an error from `PMIx_Init` if this behavior is requested. Launchers that cannot support this operation shall return an error from the `PMIx_Spawn` API if this behavior is requested.

- **PMIX_DEBUG_WAIT_FOR_NOTIFY** "pmix.dbg.notify" (bool)
  
  Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The resulting application processes are to notify their server (by generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some application-determined location and pause at that point until either released by debugger modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they are at the pause point. Launchers that cannot support this operation shall
return an error from the PMIx_Spawn API if this behavior is requested.

Note that there is no mechanism by which the PMIx library or the launcher can verify that an application will recognize and support the PMIX_DEBUG_WAIT_FOR_NOTIFY request. Debuggers utilizing this attachment method must, therefore, be prepared to deal with the case where the application fails to recognize and/or honor the request.

If the PMIx implementation and/or the host environment support it, debuggers can utilize the PMIx_Query_info API to determine which features are available via the PMIX_QUERY_ATTRIBUTE_SUPPORT attribute.

- PMIX_DEBUG_STOP_IN_INIT by checking PMIX_CLIENT_ATTRIBUTES for the PMIx_Init API.

- PMIX_DEBUG_STOP_ON_EXEC by checking PMIX_HOST_ATTRIBUTES for the PMIx_Spawn API.

The target namespace or process (as given by the debugger in the spawn request) shall be provided to each daemon in its job-level information via the PMIX_DEBUG_TARGET attribute. Debugger daemons are responsible for self-determining their specific target process(es), and can then utilize the PMIx_Query_info API to obtain information about them (see Fig 17.6) e.g., to obtain the PIDs of the local processes to which they need to attach. PMIx provides the pmix_proc_info_t structure for organizing information about a process’ PID, location, and state. Debuggers may request information on a given job at two levels:

- PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
  Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose process table is being queried.

- PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
  Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each process in the specified namespace executing on the same node as the requester, ordered by process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace whose local process table is being queried. OPTIONAL QUALIFIER: PMIX_HOSTNAME indicating the host whose local process table is being queried. By default, the query assumes that the host upon which the request was made is to be used.

Note that the information provided in the returned proctable represents a snapshot in time. Any process, regardless of role (tool, client, debugger, etc.) can obtain the proctable of a given namespace so long as it has the system-determined authorizations to do so. The list of namespaces available via a given server can be obtained using the PMIx_Query_info API with the PMIX_QUERY_NAMESPACES key.

Debugger daemons can be started in two ways - either at the same time the application is spawned, or separately at a later time.
17.4.1 Co-Location of Debugger Daemons

Debugging operations typically require the use of daemons that are located on the same node as the processes they are attempting to debug. The debugger can, of course, specify its own mapping method when issuing its spawn request or utilize its own internal launcher to place the daemons. However, when attaching to a running job, PMIx provides debuggers with a simplified method for requesting that the launcher associated with the job co-locate the required daemons. Debuggers can request co-location of their daemons by adding the following attributes to the `PMIx_Spawn` used to spawn them:

- `PMIX_DEBUGGER_DAEMONS` - indicating that the launcher is being asked to spawn debugger daemons.

- `PMIX_DEBUG_TARGET` - indicating the job or process that is to be debugged. This allows the launcher to identify the processes to be debugged and their location. Note that the debugger job shall be assigned its own namespace (different from that of the job it is being spawned to debug) and each daemon will be assigned a unique rank within that namespace.

- `PMIX_DEBUG_DAEMONS_PER_PROC` - specifies the number of debugger daemons to be co-located per target process.

- `PMIX_DEBUG_DAEMONS_PER_NODE` - specifies the number of debugger daemons to be co-located per node where at least one target process is executing.

Debugger daemons spawned in this manner shall be provided with the typical PMIx information for their own job plus the target they are to debug via the `PMIX_DEBUG_TARGET` attribute. The debugger daemons spawned on a given node are responsible for self-determining their specific target process(es) - e.g., by referencing their own `PMIX_LOCAL_RANK` in the daemon debugger job versus the corresponding `PMIX_LOCAL_RANK` of the target processes on the node. Note that...
the debugger will be attaching to the application processes at some arbitrary point in the
application’s execution unless some method for pausing the application (e.g., by providing a PMIx
directive at time of launch, or via a tool using the PMIx_Job_control API to direct that the
process be paused) has been employed.

--- Advice to users ---

Note that the tool calling PMIx_Spawn to request the launch of the debugger daemons is not
included in the resulting job - i.e., the debugger daemons do not inherit the namespace of the tool.
Thus, collective operations and notifications that target the debugger daemon job will not include
the tool unless the namespace/rank of the tool is explicitly included.

17.4.2 Co-Spawn of Debugger Daemons

In the case where a job is being spawned under the control of a debugger, PMIx provides a shortcut
method for spawning the debugger’s daemons in parallel with the job. This requires that the
debugger be specified as one of the pmix_app_t in the same spawn command used to start the
job. The debugger application must include at least the PMIX_DEBUGGER_DAEMONS attribute
identifying itself as a debugger, and may utilize either a mapping option to direct daemon
placement, or one of the PMIX_DEBUG_DAEMONS_PER_PROC or
PMIX_DEBUG_DAEMONS_PER_NODE directives.

The launcher must not include information regarding the debugger daemons in the job-level info
provided to the rest of the pmix_app_t, nor in any calculated rank values (e.g.,
PMIX_NODE_RANK or PMIX_LOCAL_RANK) in those applications. The debugger job is to be
assigned its own namespace and each debugger daemon shall receive a unique rank - i.e., the
debugger application is to be treated as a completely separate PMIx job that is simply being started
in parallel with the user’s applications. The launcher is free to implement the launch as a single
operation for both the applications and debugger daemons (preferred), or may stage the launches as
required. The launcher shall not return from the PMIx_Spawn command until all included
applications and the debugger daemons have been started.

Attributes that apply to both the debugger daemons and the application processes can be specified
in the job_info array passed into the PMIx_Spawn API. Attributes that either (a) apply solely to
the debugger daemons or to one of the applications included in the spawn request, or (b) have
values that differ from those provided in the job_info array, should be specified in the info array in
the corresponding pmix_app_t. Note that PMIx job pause attributes (e.g.,
PMIX_DEBUG_STOP_IN_INIT) do not apply to applications (defined in pmix_app_t) where
the PMIX_DEBUGGER_DAEMONS attribute is set to true.

Debugger daemons spawned in this manner shall be provided with the typical PMIx information for
their own job plus the target they are to debug via the PMIX_DEBUG_TARGET attribute. The
debugger daemons spawned on a given node are responsible for self-determining their specific
target process(es) - e.g., by referencing their own PMIX_LOCAL_RANK in the daemon debugger
job versus the corresponding PMIX_LOCAL_RANK of the target processes on the node.
Advice to users

Note that the tool calling `PMIx_Spawn` to request the launch of the debugger daemons is not included in the resulting job - i.e., the debugger daemons do not inherit the namespace of the tool. Thus, collective operations and notifications that target the debugger daemon job will not include the tool unless the namespace/rank of the tool is explicitly included.

The `PMIx_Spawn` API only supports the return of a single namespace resulting from the spawn request. In the case where the debugger job is co-spawned with the application, the spawn function shall return the namespace of the application and not the debugger job. Tools requiring access to the namespace of the debugger job must query the launcher for the spawned namespaces to find the one belonging to the debugger job.

17.4.3 Debugger Agents

Individual debuggers may, depending upon implementation, require varying degrees of control over each application process when it is started beyond those available via directives to `PMIx_Spawn`. PMIx offers two mechanisms to help provide a means of meeting these needs.

The `PMIX_FORKEXEC_AGENT` attribute allows the debugger to specify an intermediate process (the Fork/Exec Agent (FEA)) for spawning the actual application process (see Fig. 17.7a), thereby interposing the debugger daemon between the application process and the launcher’s daemon. Instead of spawning the application process, the launcher will spawn the FEA, which will connect back to the PMIx server as a tool to obtain the spawn description of the application process it is to spawn. The PMIx server in the launcher’s daemon shall not register the fork/exec agent as a local client process, nor shall the launcher include the agent in any of the job-level values (e.g., `PMIX_RANK` within the job or `PMIX_LOCAL_RANK` on the node) provided to the application process. The launcher shall treat the collection of FEAs as a debugger job equivalent to the co-spawn use-case described in Section 17.4.2.

In contrast, the `PMIX_EXEC_AGENT` attribute (Fig. 17.7b) allows the debugger to specify an agent that will perform some preparatory actions and then exec the eventual application process to replace itself. In this scenario, the exec agent is provided with the application process’ command line as arguments on its command line (e.g., "./agent appargv[0] appargv[1]") and does not connect back to the host’s PMIx server. It is the responsibility of the exec agent to properly separate its own command line arguments (if any) from the application description.
17.4.4 Tracking the job lifecycle

There are a wide range of events a debugger can register to receive, but three are specifically defined for tracking a job’s progress:

- **PMIX_EVENT_JOB_START** indicates when the first process in the job has been spawned.
- **PMIX_LAUNCH_COMPLETE** indicates when the last process in the job has been spawned.
- **PMIX_EVENT_JOB_END** indicates that all processes have terminated.

Each event is required to contain at least the namespace of the corresponding job and a **PMIX_EVENT_TIMESTAMP** indicating the time the event occurred. In addition, the **PMIX_EVENT_JOB_END** event shall contain the returned status code (**PMIX_JOB_TERM_STATUS**) for the corresponding job, plus the identity (**PMIX_PROCID**) and exit status (**PMIX_EXIT_CODE**) of the first failed process, if applicable. Generation of these events by the launcher can be requested by including the **PMIX_NOTIFY_JOB_EVENTS** attributes in the spawn request. Note that these events can be logged via the **PMix_Log** API by including the **PMIX_LOG_JOB_EVENTS** attribute - this can be done either in conjunction with generated events, or in place of them.

Alternatively, if the debugger or tool solely wants to be alerted to job termination, then including the **PMIX_NOTIFY_COMPLETION** attribute in the spawn request would suffice. This attribute directs the launcher to provide just the **PMIX_EVENT_JOB_END** event. Note that this event can be logged via the **PMix_Log** API by including the **PMIX_LOG_COMPLETION** attribute - this can be done either in conjunction with the generated event, or in place of it.

Advice to users

The PMIx server is required to cache events in order to avoid race conditions - e.g., when a tool is trying to register for the **PMIX_EVENT_JOB_END** event from a very short-lived job. Accordingly, registering for job-related events can result in receiving events relating to jobs other than the one of interest.
Users are therefore advised to specify the job whose events are of interest by including the `PMIX_EVENT_AFFECTED_PROC` or `PMIX_EVENT_AFFECTED_PROCS` attribute in the `info` array passed to the `PMix_Register_event_handler` API.

### 17.4.4.1 Job lifecycle events

- **PMIX_EVENT_JOB_START** The first process in the job has been spawned - includes `PMIX_EVENT_TIMESTAMP` as well as the `PMIX_JOBID` and/or `PMIX_NSPACE` of the job.
- **PMIX_LAUNCH_COMPLETE** All processes in the job have been spawned - includes `PMIX_EVENT_TIMESTAMP` as well as the `PMIX_JOBID` and/or `PMIX_NSPACE` of the job.
- **PMIX_EVENT_JOB_END** All processes in the job have terminated - includes `PMIX_EVENT_TIMESTAMP` when the last process terminated as well as the `PMIX_JOBID` and/or `PMIX_NSPACE` of the job.
- **PMIX_EVENT_SESSION_START** The allocation has been instantiated and is ready for use - includes `PMIX_EVENT_TIMESTAMP` as well as the `PMIX_SESSION_ID` of the allocation. This event is issued after any system-controlled prologue has completed, but before any user-specified actions are taken.
- **PMIX_EVENT_SESSION_END** The allocation has terminated - includes `PMIX_EVENT_TIMESTAMP` as well as the `PMIX_SESSION_ID` of the allocation. This event is issued after any user-specified actions have completed, but before any system-controlled epilogue is performed.

The following events relate to processes within a job:

- **PMIX_EVENT_PROC_TERMINATED** The specified process(es) terminated - normal or abnormal termination will be indicated by the `PMIX_PROC_TERM_STATUS` in the `info` array of the notification. Note that a request for individual process events can generate a significant event volume from large-scale jobs.
- **PMIX_ERR_PROC_TERM_WO_SYNC** Process terminated without calling `PMix_Finalize`, or was a member of an assemblage formed via `PMix_Connect` and terminated or called `PMix_Finalize` without first calling `PMix_Disconnect` (or its non-blocking form) from that assemblage.

The following constants may be included via the `PMIX_JOB_TERM_STATUS` attributed in the `info` array in the `PMIX_EVENT_JOB_END` event notification to provide more detailed information regarding the reason for job abnormal termination:

- **PMIX_ERR_JOB_CANCELED** The job was canceled by the host environment.
- **PMIX_ERR_JOB_ABORTED** One or more processes in the job called abort, causing the job to be terminated.
- **PMIX_ERR_JOB_KILLED_BY_CMD** The job was killed by user command.
- **PMIX_ERR_JOB_ABORTED_BY_SIG** The job was aborted due to receipt of an error signal (e.g., SIGKILL).
PMIX_ERR_JOB_TERM_WO_SYNC  The job was terminated due to at least one process terminating without calling PMIx_Finalize, or was a member of an assemblage formed via PMix_Connect and terminated or called PMix_Finalize without first calling PMix_Disconnect (or its non-blocking form) from that assemblage.

PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED  The job was terminated due to one or more processes exceeding a specified sensor limit.

PMIX_ERR_JOB_NON_ZERO_TERM  The job was terminated due to one or more processes exiting with a non-zero status.

PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT  The job was aborted due to receipt of a system event.

17.4.4.2 Job lifecycle attributes

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_EVENT_PROC_TERMINATED event is optional and host environments may choose to provide it only upon request.

17.4.5 Debugger-related constants

The following constants are used in events used to coordinate applications and the debuggers attaching to them.

PMIX_DEBUG_WAITING_FOR_NOTIFY  All processes in the job to be debugged are paused waiting for a release at some point within the application. The application shall remain in a paused state awaiting release until receipt of the PMIX_DEBUGGER_RELEASE.

PMIX_DEBUGGER_RELEASE  Release processes that are paused at the PMIX_DEBUG_WAIT_FOR_NOTIFY point in the target application.

17.4.6 Debugger attributes

Attributes used to assist debuggers - these are values that can either be passed to the PMIx_Spawn APIs or accessed by a debugger itself using the PMIx_Get API with the PMIX_RANK_WILDCARD rank.

PMIX_DEBUG_STOP_ON_EXEC "pmix.dbg.exec" (bool)
Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the application is being spawned under a debugger, and that the local launch agent is to pause the resulting application processes on first instruction for debugger attach. The launcher (RM or IL) is to generate the `PMIX_LAUNCH_COMPLETE` event when all processes are stopped at the exec point.

**PMIX_DEBUG_STOP_IN_INIT** "pmix.dbg.init" (bool)

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The PMIx client library in each resulting application process shall notify its PMIx server that it is pausing and then pause during `PMIX_Init` of the spawned processes until either released by debugger modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have reached the pause point.

**PMIX_DEBUG_WAIT_FOR_NOTIFY** "pmix.dbg.notify" (bool)

Included in either the `pmix_info_t` array in a `pmix_app_t` description (if the directive applies only to that application) or in the `job_info` array if it applies to all applications in the given spawn request. Indicates that the specified application is being spawned under a debugger. The resulting application processes are to notify their server (by generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event) when they reach some application-determined location and pause at that point until either released by debugger modification of an appropriate variable or receipt of the `PMIX_DEBUGGER_RELEASE` event. The launcher (RM or IL) is responsible for generating the `PMIX_DEBUG_WAITING_FOR_NOTIFY` event when all processes have indicated they are at the pause point.

**PMIX_DEBUG_TARGET** "pmix.dbg.tgt" (pmix_proc_t*)

Identifier of process(es) to be debugged - a rank of `PMIX_RANK_WILDCARD` indicates that all processes in the specified namespace are to be included.

**PMIX_DEBUGGER_DAEMONS** "pmix.debugger" (bool)

Included in the `pmix_info_t` array of a `pmix_app_t`, this attribute declares that the application consists of debugger daemons and shall be governed accordingly. If used as the sole `pmix_app_t` in a `PMIX_Spawn` request, then the `PMIX_DEBUG_TARGET` attribute must also be provided (in either the `job_info` or in the `info` array of the `pmix_app_t`) to identify the namespace to be debugged so that the launcher can determine where to place the spawned daemons. If neither `PMIX_DEBUG_DAEMONS_PER_PROC` nor `PMIX_DEBUG_DAEMONS_PER_NODE` is specified, then the launcher shall default to a placement policy of one daemon per process in the target job.

**PMIX_COSPAWN_APP** "pmix.cospawn" (bool)

Designated application is to be spawned as a disconnected job - i.e., the launcher shall not include the application in any of the job-level values (e.g., `PMIX_RANK` within the job)
provided to any other application process generated by the same spawn request. Typically
used to cospawn debugger daemons alongside an application.

**PMIX_DEBUG_DAEMONS_PER_PROC** "pmix.dbg.dpproc" (uint16_t)
Number of debugger daemons to be spawned per application process. The launcher is to pass
the identifier of the namespace to be debugged by including the **PMIX_DEBUG_TARGET**
attribute in the daemon’s job-level information. The debugger daemons spawned on a given
node are responsible for self-determining their specific target process(es) - e.g., by
referencing their own **PMIX_LOCAL_RANK** in the daemon debugger job versus the
corresponding **PMIX_LOCAL_RANK** of the target processes on the node.

**PMIX_DEBUG_DAEMONS_PER_NODE** "pmix.dbg.dpnd" (uint16_t)
Number of debugger daemons to be spawned on each node where the target job is executing.
The launcher is to pass the identifier of the namespace to be debugged by including the
**PMIX_DEBUG_TARGET** attribute in the daemon’s job-level information. The debugger
daemons spawned on a given node are responsible for self-determining their specific target
process(es) - e.g., by referencing their own **PMIX_LOCAL_RANK** in the daemon debugger
job versus the corresponding **PMIX_LOCAL_RANK** of the target processes on the node.

**PMIX_QUERY_PROC_TABLE** "pmix qry.ptable" (char*)
Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
**PMIX_NSPACE** indicating the namespace whose process table is being queried.

**PMIX_QUERY_LOCAL_PROC_TABLE** "pmix qry.lptable" (char*)
Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
process in the specified namespace executing on the same node as the requester, ordered by
process job rank. REQUIRED QUALIFIER: **PMIX_NSPACE** indicating the namespace
whose local process table is being queried. OPTIONAL QUALIFIER: **PMIX_HOSTNAME**
indicating the host whose local process table is being queried. By default, the query assumes
that the host upon which the request was made is to be used.

### 17.5 Tool-Specific APIs

PMIx-based tools automatically have access to all PMIx client functions. Tools designated as a
*launcher* or a *server* will also have access to all PMIx server functions. There are, however, an
additional set of functions (described in this section) that are specific to a PMIx tool. Access to
those functions require use of the tool initialization routine.

#### 17.5.1 PMIx_tool_init

**Summary**

Initialize the PMIx library for operating as a tool, optionally connecting to a specified PMIx server.
PMIx v2.0

Format

```c
pmix_status_t
PMIx_tool_init(pmix_proc_t *proc,
        pmix_info_t info[], size_t ninfo);
```

| INOUT | proc        | pmix_proc_t structure (handle) |
| IN    | info        | Array of pmix_info_t structures (array of handles) |
| IN    | ninfo       | Number of elements in the info array (size_t) |

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

Required Attributes

The following attributes are required to be supported by all PMIx libraries:

- **PMIX_TOOL_NSPACE** "pmix.tool.nspace" (char*)
  Name of the namespace to use for this tool.

- **PMIX_TOOL_RANK** "pmix.tool.rank" (uint32_t)
  Rank of this tool.

- **PMIX_TOOL_DO_NOT_CONNECT** "pmix.tool.nocon" (bool)
  The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

- **PMIX_TOOL_ATTACHMENT_FILE** "pmix.tool.attach" (char*)
  Pathname of file containing connection information to be used for attaching to a specific server.

- **PMIX_SERVER_URI** "pmix.srvr.uri" (char*)
  URI of the PMIx server to be contacted.

- **PMIX_TCP_URI** "pmix.tcp.uri" (char*)
  The URI of the PMIx server to connect to, or a file name containing it in the form of file:<name of file containing it>.

- **PMIX_SERVER_PIDINFO** "pmix.srvr.pidinfo" (pid_t)
  PID of the target PMIx server for a tool.

- **PMIX_SERVER_NSPACE** "pmix.srv.nspace" (char*)
  Name of the namespace to use for this PMIx server.

- **PMIX_CONNECT_TO_SYSTEM** "pmix.cnct.sys" (bool)
  The requester requires that a connection be made only to a local, system-level PMIx server.

- **PMIX_CONNECT_SYSTEM_FIRST** "pmix.cnct.sys.first" (bool)
Preferentially, look for a system-level PMIx server first.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

**PMIX_CONNECT_RETRY_DELAY** "pmix.tool.retry" (uint32_t)
Time in seconds between connection attempts to a PMIx server - the default value is implementation specific.

**PMIX_CONNECT_MAX_RETRIES** "pmix.tool.mretries" (uint32_t)
Maximum number of times to try to connect to PMIx server - the default value is implementation specific.

**PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)
POSIX mode_t (9 bits valid). If the library supports socket connections, this attribute may be supported for setting the socket mode.

**PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)
If provided, directs that the TCP URI be reported and indicates the desired method of reporting: ‘−’ for stdout, ‘+’ for stderr, or filename. If the library supports TCP socket connections, this attribute may be supported for reporting the URI.

**PMIX_TCP_IF_INCLUDE** "pmix.tcp.ifinclude" (char*)
Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

**PMIX_TCP_IF_EXCLUDE** "pmix.tcp.ifexclude" (char*)
Comma-delimited list of devices and/or CIDR notation to exclude when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces that are not to be used.

**PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)
The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

**PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)
The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

**PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)
Set to true to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

**PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)
Set to true to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.
PMIX_EXTERNAL_PROGRESS  "pmix.evext" (bool)

The host shall progress the PMIx library via calls to PMIx_Progress

PMIX_EVENT_BASE  "pmix.evbase" (void*)

Pointer to an event_base to use in place of the internal progress thread. All PMIx library
events are to be assigned to the provided event base. The event base must be compatible with
the event library used by the PMIx implementation - e.g., either both the host and PMIx
library must use libevent, or both must use libev. Cross-matches are unlikely to work and
should be avoided - it is the responsibility of the host to ensure that the PMIx
implementation supports (and was built with) the appropriate event library.

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 initialization
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, by calling the PMIx_tool_attach_to_server function. If provided,
the proc structure will be set to a zero-length namespace and a rank of PMIX_RANK_UNDEF unless
the PMIX_TOOL_NSPACE and PMIX_TOOL_RANK attributes are included in the info array.

In all other cases, the PMIx tool library will automatically attempt to connect to a PMIx server
according to the precedence chain described in Section 17.1. If successful, the function will return
PMIX_SUCCESS and will fill the process structure (if provided) with the assigned namespace and
rank of the tool. The server to which the tool connects will be designated its primary server. 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.

Note that the PMIx tool library is referenced counted, and so multiple calls to PMIx_tool_init
are allowed. If the tool is not connected to any server when this API is called, then the tool will
attempt to connect to a server unless the PMIX_TOOL_DO_NOT_CONNECT is included in the call
to API.

17.5.2  PMIx_tool_finalize

Summary

Finalize the PMIx tool library.
PMIx tool finalize

```
#define PMIx_tool_finalize(void);

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

**Description**

Finalize the PMIx tool library, closing all existing connections to servers. An error code will be returned if, for some reason, a connection cannot be cleanly terminated — in such cases, the connection is dropped. Upon detecting loss of the connection, the PMIx server shall cleanup all associated records of the tool.

---

**17.5.3 PMIx_tool_disconnect**

**Summary**

Disconnect the PMIx tool from the specified server connection while leaving the tool library initialized.

```
#define PMIx_tool_disconnect(const pmix_proc_t *server);

IN server

pmix_proc_t structure (handle)

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

**Description**

Close the current connection to the specified server, if one has been made, while leaving the PMIx library initialized. An error code will be returned if, for some reason, the connection cannot be cleanly terminated - in this case, the connection is dropped. In either case, the library will remain initialized. Upon detecting loss of the connection, the PMIx server shall cleanup all associated records of the tool.

Note that if the server being disconnected is the current primary server, then all operations requiring support from a server will return the PMIX_ERR_UNREACH error until the tool either designates an existing connection to be the primary server or, if no other connections exist, the tool establishes a connection to a PMIx server.
17.5.4 PMIx_tool_attach_to_server

Summary
Establish a connection to a PMIx server.

Format

PMIx v4.0

```c
pmix_status_t PMIx_tool_attach_to_server(pmix_proc_t *proc,
                                         pmix_proc_t *server,
                                         pmix_info_t info[],
                                         size_t ninfo);
```

**INOUT** `proc`
Point to `pmix_proc_t` structure (handle)

**INOUT** `server`
Point to `pmix_proc_t` structure (handle)

**IN** `info`
Array of `pmix_info_t` structures (array of handles)

**IN** `ninfo`
Number of elements in the `info` array (`size_t`)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

**Required Attributes**

The following attributes are required to be supported by all PMIx libraries:

**PMIX_TOOL_ATTACHMENT_FILE** "pmix.tool.attach" (char*)
Pathname of file containing connection information to be used for attaching to a specific server.

**PMIX_SERVER_URI** "pmix.srvr.uri" (char*)
URI of the PMIx server to be contacted.

**PMIX_TCP_URI** "pmix.tcp.uri" (char*)
The URI of the PMIx server to connect to, or a file name containing it in the form of `file:<name of file containing it>`.

**PMIX_SERVER_PIDINFO** "pmix.srvr.pidinfo" (pid_t)
PID of the target PMIx server for a tool.

**PMIX_SERVER_NSPACE** "pmix.srv.nspace" (char*)
Name of the namespace to use for this PMIx server.

**PMIX_CONNECT_TO_SYSTEM** "pmix.cnct.sys" (bool)
The requester requires that a connection be made only to a local, system-level PMIx server.
**PMIX_CONNECT_SYSTEM_FIRST** "pmix.cnct.sys.first" (bool)
Preferentially, look for a system-level PMIx server first.

**PMIX_PRIMARY_SERVER** "pmix.pri.srvr" (bool)
The server to which the tool is connecting shall be designated the primary server once connection has been accomplished.

---

**Description**
Establish a connection to a server. This function can be called at any time by a PMIx tool to create a new connection to a server. If a specific server is given and the tool is already attached to it, then the API shall return **PMIX_SUCCESS** without taking any further action. In all other cases, the tool will attempt to discover a server using the method described in Section 17.1, ignoring all candidates to which it is already connected. The **PMIX_ERR_UNREACH** error shall be returned if no new connection is made.

The process identifier assigned to this tool is returned in the provided proc structure. Passing a value of **NULL** for the proc parameter is allowed if the user wishes solely to connect to a PMIx server and does not require return of the identifier at that time.

The process identifier of the server to which the tool attached is returned in the server structure. Passing a value of **NULL** for the proc parameter is allowed if the user wishes solely to connect to a PMIx server and does not require return of the identifier at that time.

Note that the **PMIX_PRIMARY_SERVER** attribute must be included in the info array if the server being connected to is to become the primary server, or a call to **PMIx_tool_set_server** must be provided immediately after the call to this function.

---

**Advice to PMIx library implementers**
When a tool connects to a server that is under a different namespace manager (e.g., host RM) from the prior server, the namespace in the identifier of the tool must remain unique in the new universe. If the namespace of the tool fails to meet this criteria in the new universe, then the new namespace manager is required to return an error and the connection attempt must fail.

---

**Advice to users**
Some PMIx implementations may not support connecting to a server that is not under the same namespace manager (e.g., host RM) as the server to which the tool is currently connected.
17.5.5 PMIx_tool_get_servers

Summary
Get an array containing the `pmix_proc_t` process identifiers of all servers to which the tool is currently connected.

Format

```c
pmix_status_t
PMIx_tool_get_servers(pmix_proc_t *servers[], size_t *nservers);
```

OUT servers
Address where the pointer to an array of `pmix_proc_t` structures shall be returned (handle)

INOUT nservers
Address where the number of elements in `servers` shall be returned (handle)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

Description
Return an array containing the `pmix_proc_t` process identifiers of all servers to which the tool is currently connected. The process identifier of the current primary server shall be the first entry in the array, with the remaining entries in order of attachment from earliest to most recent.

17.5.6 PMIx_tool_set_server

Summary
Designate a server as the tool’s primary server.

Format

```c
pmix_status_t
PMIx_tool_set_server(const pmix_proc_t *server,
                     pmix_info_t info[], size_t ninfo);
```

IN server
`pmix_proc_t` structure (handle)

IN info
Array of `pmix_info_t` structures (array of handles)

IN ninfo
Number of elements in the `info` array (`size_t`)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.
Required Attributes

The following attributes are required to be supported by all PMIx libraries:

1. **PMIX_WAIT_FOR_CONNECTION** "pmix.wait.conn" (bool)
   - Wait until the specified process has connected to the requesting tool or server, or the operation times out (if the **PMIX_TIMEOUT** directive is included in the request).

2. **PMIX_TIMEOUT** "pmix.timeout" (int)
   - Time in seconds before the specified operation should time out (zero indicating infinite) and return the **PMIX_ERR_TIMEOUT** error. Care should be taken to avoid race conditions caused by multiple layers (client, server, and host) simultaneously timing the operation.

Description

Designate the specified server to be the tool’s primary server for all subsequent API calls.

17.5.7 **PMIx_IOF_pull**

Summary

Register to receive output forwarded from a set of remote processes.

Format

```c
PMIx v3.0
```

```c
pmix_status_t
PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,
const pmix_info_t directives[], size_t ndirs,
pmix_iof_channel_t channel,
pmix_iof_cbfunc_t cbfunc,
pmix_hdlr_reg_cbfunc_t regcbfunc,
void *regcbdata);
```

**IN** `procs`
- Array of proc structures identifying desired source processes (array of handles)

**IN** `nprocs`
- Number of elements in the `procs` array (integer)

**IN** `directives`
- Array of `pmix_info_t` structures (array of handles)

**IN** `ndirs`
- Number of elements in the `directives` array (integer)

**IN** `channel`
- Bitmask of IO channels included in the request (`pmix_iof_channel_t`)
IN cbfunc
  Callback function for delivering relevant output (pmix_iof_cbfunc_t function reference)

IN regcbfunc
  Function to be called when registration is completed (pmix_hdlr_reg_cbfunc_t function reference)

IN regcbdata
  Data to be passed to the regcbfunc callback function (memory reference)

Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant. In the event
the function returns an error, the regcbfunc will not be called.

Required Attributes

The following attributes are required for PMIx libraries that support IO forwarding:

PMIX_IOF_CACHE_SIZE "pmix.iof.csize" (uint32_t)
  The requested size of the PMIx server cache in bytes for each specified channel. By default,
  the server is allowed (but not required) to drop all bytes received beyond the max size.

PMIX_IOF_DROP_OLDEST "pmix.iof.old" (bool)
  In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the
  cache.

PMIX_IOF_DROP_NEWEST "pmix.iof.new" (bool)
  In an overflow situation, the PMIx server is to drop any new bytes received until room
  becomes available in the cache (default).

Optional Attributes

The following attributes are optional for PMIx libraries that support IO forwarding:

PMIX_IOF_BUFFERING_SIZE "pmix.iof.bsize" (uint32_t)
  Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the
  specified number of bytes is collected to avoid being called every time a block of IO arrives.
  The PMIx tool library will execute the callback and reset the collection counter whenever the
  specified number of bytes becomes available. Any remaining buffered data will be flushed to
  the callback upon a call to deregister the respective channel.

PMIX_IOF_BUFFERING_TIME "pmix.iof.btime" (uint32_t)
  Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
  size, this prevents IO from being held indefinitely while waiting for another payload to
  arrive.

PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool)
  Requests that output be prefixed with the nspace, rank of the source and a string identifying
  the channel (stdout, stderr, etc.).

PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts" (bool)
Requests that output be marked with the time at which the data was received by the tool - note that this will differ from the time at which the data was collected from the source.

**PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)

Requests that output be formatted in XML.

**Description**

Register to receive output forwarded from a set of remote processes.

**Advice to users**

Providing a NULL function pointer for the `cbfunc` parameter will cause output for the indicated channels to be written to their corresponding `stdout/stderr` file descriptors. Use of `PMIX_RANK_WILDCARD` to specify all processes in a given namespace is supported but should be used carefully due to bandwidth and memory footprint considerations.

### 17.5.8 PMIx_IOF_deregister

**Summary**

Deregister from output forwarded from a set of remote processes.

**Format**

```c
pmix_status_t
PMIx_IOF_deregister(size_t iofhdlr,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_op_cbfunc_t cbfunc, void *cbdata);
```

- **iofhdlr**
  
  Registration number returned from the `pmix_hdlr_reg_cbfunc_t` callback from the call to `PMIx_IOF_pull(size_t)`

- **directives**
  
  Array of `pmix_info_t` structures (array of handles)

- **ndirs**
  
  Number of elements in the `directives` array (integer)

- **cbfunc**
  
  Callback function to be called when deregistration has been completed. (function reference)

- **cbdata**
  
  Data to be passed to the `cbfunc` callback function (memory reference)

Returns one of the following:
- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided \( cbfunc \). Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned success - the \( cbfunc \) will not be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the \( cbfunc \) will not be called

**Description**

Deregister from output forwarded from a set of remote processes.

--- Advice to PMIx library implementers ---

Any currently buffered IO should be flushed upon receipt of a deregistration request. All received IO after receipt of the request shall be discarded.

--- 17.5.9 PMIx_IOF_push ---

**Summary**

Push data collected locally (typically from \( \text{stdin} \) or a file) to \( \text{stdin} \) of the target recipients.

**Format**

PMIx v3.0

```
const pmix_proc_t targets[], size_t ntargets,
const pmix_byte_object_t *bo,
const pmix_info_t directives[], size_t ndirs,
pmix_op_cbfunc_t cbfunc, void *cbdata);
```

**IN** targets

Array of proc structures identifying desired target processes (array of handles)

**IN** ntargets

Number of elements in the \( \text{targets} \) array (integer)

**IN** bo

Pointer to \( \text{pmix_byte_object_t} \) containing the payload to be delivered (handle)

**IN** directives

Array of \( \text{pmix_info_t} \) structures (array of handles)

**IN** ndirs

Number of elements in the \( \text{directives} \) array (integer)

**IN** directives

Array of \( \text{pmix_info_t} \) structures (array of handles)
IN `cbfunc`
   Callback function to be called when operation has been completed. (`pmix_op_cbfunc_t` function reference)

IN `cbdata`
   Data to be passed to the `cbfunc` callback function (memory reference)

Returns one of the following:

- **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result will be returned in the provided `cbfunc`. Note that the library must not invoke the callback function prior to returning from the API.

- **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned `success` - the `cbfunc` will not be called.

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called.

--- Required Attributes ---

The following attributes are required for PMIx libraries that support IO forwarding:

**PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (uint32_t)
   The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

**PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (bool)
   In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.

**PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (bool)
   In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).

--- Optional Attributes ---

The following attributes are optional for PMIx libraries that support IO forwarding:

**PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (uint32_t)
   Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be flushed to the callback upon a call to deregister the respective channel.

**PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (uint32_t)
   Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
**PMIX_IOF_PUSH_STDIN** "pmix.iof.stdin" (bool)

Requests that the PMIx library collect the stdin of the requester and forward it to the processes specified in the **PMIX_IOF_push** call. All collected data is sent to the same targets until stdin is closed, or a subsequent call to **PMIX_IOF_push** is made that includes the **PMIX_IOF_COMPLETE** attribute indicating that forwarding of stdin is to be terminated.

---

**Description**

Called either to:

- push data collected by the caller themselves (typically from stdin or a file) to stdin of the target recipients;
- request that the PMIx library automatically collect and push the stdin of the caller to the target recipients; or
- indicate that automatic collection and transmittal of stdin is to stop

---

**Advice to users**

Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires the caller to maintain the *bo* data object - it does not indicate delivery of the payload to the targets. Use of **PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but should be used carefully due to bandwidth and memory footprint considerations.
APPENDIX A

Python Bindings

While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of PMIx to that specific language. Support for other languages is captured in the Standard by describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes. Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.

Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx “class” (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and therefore those have been included in the “client” class. Servers can utilize any of those functions plus a set focused on operations not commonly executed by an application process. Finally, tools can also act as servers but have their own initialization function.

A.1 Design Considerations

Several issues arose during design of the Python bindings:

A.1.1 Error Codes vs Python Exceptions

The C programming language reports errors through the return of the corresponding integer status codes. PMIx has defined a range of negative values for this purpose. However, Python has the option of raising exceptions that effectively operate as interrupts that can be trapped if the program appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and return PMIx status codes in lieu of raising exceptions as this method was considered more consistent for those working in both domains.

A.1.2 Representation of Structured Data

PMIx utilizes a number of C-language structures to efficiently bundle related information. For example, the PMIx process identifier is represented as a struct containing a character array for the namespace and a 32-bit unsigned integer for the process rank. There are several options for translating such objects to Python – e.g., the PMIx process identifier could be represented as a two-element tuple (nspace, rank) or as a dictionary ‘nspace’: name, ‘rank’: 0. Exploration found no discernible benefit to either representation, nor was any clearly identifiable rationale developed that would lead a user to expect one versus the other for a given PMIx data type. Consistency in the translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion. Hence, the decision was made to express all complex datatypes as Python dictionaries.
A.2 Datatype Definitions

PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range integers (e.g., uint32_t), and structures. Each datatype is represented by a named unsigned 16-bit integer (uint16_t) constant. Users are advised to use the named PMIx constants for indicating datatypes instead of integer values to ensure compatibility with future PMIx versions.

With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 12 directly translate to Python. However, Python lacks the size-specific value definitions of C (e.g., uint8_t) and thus some care must be taken to protect against overflow/underflow situations when moving between the languages. Python bindings that accept values including PMIx datatypes shall therefore have the datatype and associated value checked for compatibility with their PMIx-defined equivalents, returning an error if:

- datatypes not defined by PMIx are encountered
- provided values fall outside the range of the C-equivalent definition - e.g., if a value identified as PMIX_UINT8 lies outside the uint8_t range

Note that explicit labeling of PMIx data type, 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 data types in the two languages.
<table>
<thead>
<tr>
<th>C-Definition</th>
<th>PMIx Name</th>
<th>Python Definition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>PMIX_BOOL</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>PMIX_BYTE</td>
<td>A single element byte array (i.e., a byte array of length one)</td>
<td></td>
</tr>
<tr>
<td>char*</td>
<td>PMIX_STRING</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>size_t</td>
<td>PMIX_SIZE</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>pid_t</td>
<td>PMIX_PID</td>
<td>integer</td>
<td>value shall be limited to the uint32_t range</td>
</tr>
<tr>
<td>int, int8_t, int16_t, int32_t, int64_t</td>
<td>PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64</td>
<td>integer</td>
<td>value shall be limited to its corresponding range</td>
</tr>
<tr>
<td>uint, uint8_t, uint16_t, uint32_t, uint64_t</td>
<td>PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64</td>
<td>integer</td>
<td>value shall be limited to its corresponding range</td>
</tr>
<tr>
<td>float, double</td>
<td>PMIX_FLOAT, PMIX_DOUBLE</td>
<td>float</td>
<td>value shall be limited to its corresponding range</td>
</tr>
<tr>
<td>struct timeval</td>
<td>PMIX_TIMEVAL</td>
<td>{'sec': sec, 'usec': microsec}</td>
<td>each field is an integer value</td>
</tr>
<tr>
<td>time_t</td>
<td>PMIX_TIME</td>
<td>integer</td>
<td>limited to positive values</td>
</tr>
<tr>
<td>pmix_data_type_t</td>
<td>PMIX_DATA_TYPE</td>
<td>integer</td>
<td>value shall be limited to the uint16_t range</td>
</tr>
<tr>
<td>pmix_status_t</td>
<td>PMIX_STATUS</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>pmix_key_t</td>
<td>N/A</td>
<td>string</td>
<td>The string’s length shall be limited to one less than the size of the pmix_key_t array (to reserve space for the terminating NULL)</td>
</tr>
<tr>
<td>pmix_nspace_t</td>
<td>N/A</td>
<td>string</td>
<td>The string’s length shall be limited to one less than the size of the pmix_nspace_t array (to reserve space for the terminating NULL)</td>
</tr>
</tbody>
</table>
Table A.1.: C-to-Python Datatype Correspondence

<table>
<thead>
<tr>
<th>C-Definition</th>
<th>PMIx Name</th>
<th>Python Definition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmix_rank_t</td>
<td>PMIX_PROC_RANK</td>
<td>integer</td>
<td>value shall be limited to the uint32_t range excepting the reserved values near UINT32_MAX</td>
</tr>
<tr>
<td>pmix_proc_t</td>
<td>PMIX_PROC</td>
<td>{'nspace': nspace, 'rank': rank}</td>
<td>namespace is a Python string and rank is an integer value. The namespace 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</td>
</tr>
<tr>
<td>pmix_byte_object_t</td>
<td>PMIX_BYTE_OBJECT</td>
<td>{'bytes': bytes, 'size': size}</td>
<td>bytes is a Python byte array and size is the integer number of bytes in that array.</td>
</tr>
<tr>
<td>pmix_persistence_t</td>
<td>PMIX_PERSISTENCE</td>
<td>integer</td>
<td>value shall be limited to the uint8_t range</td>
</tr>
<tr>
<td>pmix_scope_t</td>
<td>PMIX_SCOPE</td>
<td>integer</td>
<td>value shall be limited to the uint8_t range</td>
</tr>
<tr>
<td>pmix_data_range_t</td>
<td>PMIX_RANGE</td>
<td>integer</td>
<td>value shall be limited to the uint8_t range</td>
</tr>
<tr>
<td>pmix_proc_state_t</td>
<td>PMIX_PROC_STATE</td>
<td>integer</td>
<td>value shall be limited to the uint8_t range</td>
</tr>
<tr>
<td>pmix_proc_info_t</td>
<td>PMIX_PROC_INFO</td>
<td>{'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state}</td>
<td>proc is a Python proc dictionary; hostname and executable are Python strings; and pid, exitcode, and state are Python integers</td>
</tr>
</tbody>
</table>
## Table A.1.: C-to-Python Datatype Correspondence

<table>
<thead>
<tr>
<th>C-Definition</th>
<th>PMIx Name</th>
<th>Python Definition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pmix_data_array_t</code></td>
<td>PMIX_DATA_ARRAY</td>
<td>{'type': type, 'array': array}</td>
<td><em>type</em> is the PMIx type of object in the array and <em>array</em> is a Python <em>list</em> containing the individual array elements. Note that <em>array</em> can consist of <em>any</em> PMIx types, including (for example) a Python <em>info</em> object that itself contains an <em>array</em> value.</td>
</tr>
<tr>
<td><code>pmix_info_directives_t</code></td>
<td>PMIX_INFO_DIRECTIVES</td>
<td>list</td>
<td>List of integer values (defined in Section 3.2.10)</td>
</tr>
<tr>
<td><code>pmix_alloc_directive_t</code></td>
<td>PMIX_ALLOC_DIRECTIVE</td>
<td>integer</td>
<td>Value shall be limited to the <em>uint8_t</em> range</td>
</tr>
<tr>
<td><code>pmix_iof_channel_t</code></td>
<td>PMIX_IOF_CHANNEL</td>
<td>list</td>
<td>List of integer values (defined in Section 17.3.3)</td>
</tr>
<tr>
<td><code>pmix_envar_t</code></td>
<td>PMIX_ENVAR</td>
<td>{'envar': envar, 'value': value, 'separator': separator}</td>
<td>Envar and value are Python strings, and separator a single-character Python string</td>
</tr>
<tr>
<td><code>pmix_value_t</code></td>
<td>PMIX_VALUE</td>
<td>{'value': value, 'val_type': type}</td>
<td><em>type</em> is the PMIx datatype of <em>value</em>, and value is the associated value expressed in the appropriate Python form for the specified datatype</td>
</tr>
<tr>
<td><code>pmix_info_t</code></td>
<td>PMIX_INFO</td>
<td>{'key': key, 'flags': flags, value': value, 'val_type': type}</td>
<td>Key is a Python string key, flags is an <em>info directives</em> value, <em>type</em> is the PMIx datatype of <em>value</em>, and value is the associated value expressed in the appropriate Python form for the specified datatype</td>
</tr>
<tr>
<td><code>pmix_pdata_t</code></td>
<td>PMIX_PDATA</td>
<td>{'proc': {'nspace': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type}</td>
<td>Proc is a Python proc dictionary; key is a Python string key; <em>type</em> is the PMIx datatype of <em>value</em>; and value is the associated value expressed in the appropriate Python form for the specified datatype</td>
</tr>
<tr>
<td>C-Definition</td>
<td>PMIx Name</td>
<td>Python Definition</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pmix_app_t</td>
<td>PMIX_APP</td>
<td>{'cmd': cmd, 'argv': [argv], 'env': [env], 'maxprocs': maxprocs, 'info': [info]}</td>
<td><em>cmd is a Python string; argv and env are Python lists containing Python strings; maxprocs is an integer; and info is a Python list of info values</em></td>
</tr>
<tr>
<td>pmix_query_t</td>
<td>PMIX_QUERY</td>
<td>{'keys': [keys], 'qualifiers': [info]}</td>
<td><em>keys is a Python list of Python strings, and qualifiers is a Python list of info values</em></td>
</tr>
<tr>
<td>pmix_regattr_t</td>
<td>PMIX_REGATTR</td>
<td>{'name': name, 'key': key, 'type': type, 'info': [info], 'description': [desc]}</td>
<td><em>name and string are Python strings; type is the PMIx datatype for the attribute’s value; info is a Python list of info values; and description is a list of Python strings describing the attribute</em></td>
</tr>
<tr>
<td>pmix_job_state_t</td>
<td>PMIX_JOB_STATE</td>
<td>integer</td>
<td><em>value shall be limited to the uint8_t range</em></td>
</tr>
<tr>
<td>pmix_link_state_t</td>
<td>PMIX_LINK_STATE</td>
<td>integer</td>
<td><em>value shall be limited to the uint8_t range</em></td>
</tr>
<tr>
<td>pmix_cpuset_t</td>
<td>PMIX_PROC_CPUSER</td>
<td>{'source': source, 'cpus': bitmap}</td>
<td><em>source is a string name of the library that created the cpuset; and cpus is a list of string ranges identifying the PUs to which the process is bound (e.g., [1, 3-5, 7])</em></td>
</tr>
<tr>
<td>pmix_locality_t</td>
<td>PMIX_LOCTYPE</td>
<td>list</td>
<td><em>list of integer values (defined in Section 11.4.2.3) describing the relative locality of the specified local process</em></td>
</tr>
<tr>
<td>pmix_fabric_t</td>
<td>N/A</td>
<td>{'name': name, 'index': idx, 'info': [info]}</td>
<td><em>name is the string name assigned to the fabric; index is the integer ID assigned to the fabric; info is a list of info describing the fabric</em></td>
</tr>
<tr>
<td>pmix_endpoint_t</td>
<td>PMIX_ENDPOINT</td>
<td>{'uuid': uuid, 'osname': osname, 'endpt': endpt}</td>
<td><em>uuid is the string system-unique identifier assigned to the device; osname is the operating system name assigned to the device; endpt is a byteobject containing the endpoint information</em></td>
</tr>
<tr>
<td>C-Definition</td>
<td>PMIx Name</td>
<td>Python Definition</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>pmix_device_distance_t</code></td>
<td>PMIX_DEVICE_DIST</td>
<td><code>{‘uuid’: uuid, ’osname’: osname, ’mindist’: mindist, ’maxdist’: maxdist}</code></td>
<td><em>uuid</em> is the string system-unique identifier assigned to the device; <em>osname</em> is the operating system name assigned to the device; and <em>mindist</em> and <em>maxdist</em> are Python integers</td>
</tr>
<tr>
<td><code>pmix_coord_t</code></td>
<td>PMIX_COORD</td>
<td><code>{‘view’: view, ‘coord’: [coords]}</code></td>
<td><em>view</em> is the <code>pmix_coord_view_t</code> of the coordinate; and <em>coord</em> is a list of integer coordinates, one for each dimension of the fabric</td>
</tr>
<tr>
<td><code>pmix_geometry_t</code></td>
<td>PMIX_GEOMETRY</td>
<td><code>{‘fabric’: idx, ’uuid’: uuid, ’osname’: osname, coordinates’: [coords]}</code></td>
<td><em>fabric</em> is the Python integer index of the fabric; <em>uuid</em> is the string system-unique identifier assigned to the device; <em>osname</em> is the operating system name assigned to the device; and <em>coordinates</em> is a list of <em>coord</em> containing the coordinates for the device across all views</td>
</tr>
<tr>
<td><code>pmix_device_type_t</code></td>
<td>PMIX_DEVTYPE</td>
<td>list</td>
<td>list of integer values (defined in Section 11.4.8)</td>
</tr>
<tr>
<td><code>pmix_bind_envelope_t</code></td>
<td>N/A</td>
<td>integer</td>
<td>one of the values defined in Section 11.4.4.1</td>
</tr>
</tbody>
</table>
A.2.1 Example

Converting a C-based program to its Python equivalent requires translation of the relevant datatypes as well as use of the appropriate API form. An example small program may help illustrate the changes. Consider the following C-based program snippet:

```c
#include <pmix.h>
...

pmix_info_t info[2];
PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "TEST", PMIX_STRING)
PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "PMIX", PMIX_STRING)
rc = PMIx_Init(&myproc, info, 2);
PMIX_INFO_DESTRUCT(&info[0]); // free the copied string
PMIX_INFO_DESTRUCT(&info[1]); // free the copied string
```

Moving to the Python version requires that the `pmix_info_t` be translated to the Python `info` equivalent, and that the returned information be captured in the return parameters as opposed to a pointer parameter in the function call, as shown below:

```python
import pmix
...

myclient = PMIxClient()
info = [{'key':PMIX_PROGRAMMING_MODEL, 
     'value':'TEST', 'val_type':PMIX_STRING},
    {'key':PMIX_MODEL_LIBRARY_NAME, 
     'value':'PMIX', 'val_type':PMIX_STRING}]
(rc,myproc) = myclient.init(info)
```

Note the use of the `PMIX_STRING` identifier to ensure the Python bindings interpret the provided string value as a PMIx "string" and not an array of bytes.
A.3 Callback Function Definitions

A.3.1 IOF Delivery Function

Summary
Callback function for delivering forwarded IO to a process

Format

```
PMIx v4.0

def iofcbfunc(iofhdlr:integer, channel:bitarray,
source:dict, payload:dict, info:list)
```

IN `iofhdlr`
Registration number of the handler being invoked (integer)

IN `channel`
Python `channel` 16-bit bitarray identifying the channel the data arrived on (bitarray)

IN `source`
Python `proc` identifying the namespace/rank of the process that generated the data (dict)

IN `payload`
Python `byteobject` containing the data (dict)

IN `info`
List of Python `info` provided by the source containing metadata about the payload. This
could include `PMIX_IOF_COMPLETE` (list)

Returns: nothing

See `pmix_iof_cbfunc_t` for details

A.3.2 Event Handler

Summary
Callback function for event handlers

Format

```
PMIx v4.0
```
def evhandler(evhdlr: integer, status: integer,
source: dict, info: list, results: list)

IN iofhdlr
Registration number of the handler being invoked (integer)

IN status
Status associated with the operation (integer)

IN source
Python proc identifying the namespace/rank of the process that generated the event (dict)

IN info
List of Python info provided by the source containing metadata about the event (list)

IN results
List of Python info containing the aggregated results of all prior evhandlers (list)

Returns:
• rc - Status returned by the event handler’s operation (integer)
• results - List of Python info containing results from this event handler’s operation on the event (list)

See pmix_notification_fn_t for details

A.3.3 Server Module Functions

The following definitions represent functions that may be provided to the PMIx server library at
time of initialization for servicing of client requests. Module functions that are not provided default
to returning "not supported" to the caller.

A.3.3.1 Client Connected

Summary
Notify the host server that a client connected to this server.

Format
def clientconnected2(proc:dict is not None, info:list):
    IN proc
    Python proc identifying the namespace/rank of the process that connected (dict)
    IN info
    list of Python info containing information about the process (list)

    Returns:
    • rc - PMIX_SUCCESS or a PMIx error code indicating the connection should be rejected (integer)

    See pmix_server_client_connected2_fn_t for details

A.3.3.2 Client Finalized

Summary
Notify the host environment that a client called PMIx_Finalize.

Format
PMIx v4.0

def clientfinalized(proc:dict is not None):
    IN proc
    Python proc identifying the namespace/rank of the process that finalized (dict)

    Returns: nothing

    See pmix_server_client_finalized_fn_t for details

A.3.3.3 Client Aborted

Summary
Notify the host environment that a local client called PMIx_Abort.
def client_aborted(args: dict is not None)

IN args
Python dictionary containing:
- 'caller': Python process identifying the namespace/rank of the process calling abort (dict)
- 'status': PMIx status to be returned on exit (integer)
- 'msg': Optional string message to be printed (string)
- 'targets': Optional list of Python processes identifying the namespace/rank of the processes to be aborted (list)

Returns:
- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
See pmix_server_abort_fn_t for details

A.3.3.4 Fence

Summary
At least one client called either PMIx_Fence or PMIx_Fence_nb

def fence(args: dict is not None)

IN args
Python dictionary containing:
- 'procs': List of Python processes identifying the namespace/rank of the participating processes (list)
- 'directives': Optional list of Python containing directives controlling the operation (list)
- 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)

Returns:
- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
- data - Python bytearray containing the aggregated data from all participants (bytearray)
See pmix_server_fencenb_fn_t for details

PMIx Standard – Version 4.0 – December 2020
A.3.3.5 Direct Modex

Summary
Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc.

Format

```python
PMIx v4.0
Python

```  
```python
def dmodex(args: dict is not None)
```  
**IN** args
Python dictionary containing:
- 'proc': Python `proc` of process whose data is being requested (dict)
- 'directives': Optional list of Python `info` containing directives controlling the operation (list)

Returns:
- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)
- `data` - Python bytearray containing the data for the specified process (bytearray)

See `pmix_server_dmodex_req_fn_t` for details

A.3.3.6 Publish

Summary
Publish data per the PMIx API specification.

Format

```python
PMIx v4.0
Python

```  
```python
def publish(args: dict is not None)
```  
**IN** args
Python dictionary containing:
- 'proc': Python `proc` dictionary of process publishing the data (dict)
- 'directives': List of Python `info` containing data and directives (list)

Returns:
- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

See `pmix_server_publish_fn_t` for details
**A.3.3.7 Lookup**

**Summary**
Lookup published data.

**Format**

```python
PMIx v4.0

def lookup(args: dict is not None)

IN args
Python dictionary containing:
- 'proc': Python proc of process seeking the data (dict)
- 'keys': List of Python strings (list)
- 'directives': Optional list of Python info containing directives (list)

Returns:
- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
- pdata - List of pdata containing the returned results (list)

See pmix_server_lookup_fn_t for details
```

**A.3.3.8 Unpublish**

**Summary**
Delete data from the data store.

**Format**

```python
PMIx v4.0

def unpublish(args: dict is not None)

IN args
Python dictionary containing:
- 'proc': Python proc of process unpublishing data (dict)
- 'keys': List of Python strings (list)
- 'directives': Optional list of Python info containing directives (list)

Returns:
- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See pmix_server_unpublish_fn_t for details
```
A.3.3.9  Spawn

Summary
Spawn a set of applications/processes as per the PMIx_Spawn API.

Format
PMIx v4.0

```python
def spawn(args: dict is not None)
```

IN  args
Python dictionary containing:

- 'proc': Python `proc` of process making the request (dict)
- 'jobinfo': Optional list of Python `info` job-level directives and information (list)
- 'apps': List of Python `app` describing applications to be spawned (list)

Returns:

- `rc` - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
- `namespace` - Python string containing namespace of the spawned job (str)

See `pmix_server_spawn_fn_t` for details

A.3.3.10  Connect

Summary
Record the specified processes as connected.

Format
PMIx v4.0

```python
def connect(args: dict is not None)
```

IN  args
Python dictionary containing:

- 'procs': List of Python `proc` identifying the namespace/rank of the participating processes (list)
- 'directives': Optional list of Python `info` containing directives controlling the operation (list)

Returns:

- `rc` - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See `pmix_server_connect_fn_t` for details
A.3.3.11 Disconnect

Summary
Disconnect a previously connected set of processes.

Format
PMIx v4.0

```python
def disconnect(args: dict is not None)
```

IN args
Python dictionary containing:

- 'procs': List of Python proc identifying the namespace/rank of the participating processes (list)
- 'directives': Optional list of Python info containing directives controlling the operation (list)

Returns:
- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See `pmix_server_disconnect_fn_t` for details

A.3.3.12 Register Events

Summary
Register to receive notifications for the specified events.

Format
PMIx v4.0

```python
def register_events(args: dict is not None)
```

IN args
Python dictionary containing:

- 'codes': List of Python integers (list)
- 'directives': Optional list of Python info containing directives controlling the operation (list)

Returns:
- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See `pmix_server_register_events_fn_t` for details
A.3.3.13 Deregister Events

Summary
Deregister to receive notifications for the specified events.

Format

PMIx v4.0

```python
def deregister_events(args: dict is not None)
```

IN args
Python dictionary containing:

• 'codes': List of Python integers (list)

Returns:

• rc - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

See `pmix_server_deregister_events_fn_t` for details

A.3.3.14 Notify Event

Summary
Notify the specified range of processes of an event.

Format

PMIx v4.0

```python
def notify_event(args: dict is not None)
```

IN args
Python dictionary containing:

• 'code': Python integer `pmix_status_t` (integer)
• 'source': Python `proc` of process that generated the event (dict)
• 'range': Python `range` in which the event is to be reported (integer)
• 'directives': Optional list of Python `info` directives (list)

Returns:

• rc - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

See `pmix_server_notify_event_fn_t` for details

A.3.3.15 Query

Summary
Query information from the resource manager.
### Format

```
PMIx v4.0
```

```python
def query(args:dict is not None)
    # IN args
    # Python dictionary containing:
    #   - 'source': Python proc of requesting process (dict)
    #   - 'queries': List of Python query directives (list)

    Returns:
    #   - rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
    #   - info - List of Python info containing the returned results (list)

    See pmix_server_query_fn_t for details
```

### A.3.3.16 Tool Connected

#### Summary

Register that a tool has connected to the server.

```
PMIx v4.0
```

```python
def tool_connected(args:dict is not None)
    # IN args
    # Python dictionary containing:
    #   - 'directives': Optional list of Python info info on the connecting tool (list)

    Returns:
    #   - rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
    #   - proc - Python proc containing the assigned namespace:rank for the tool (dict)

    See pmix_server_tool_connection_fn_t for details
```

### A.3.3.17 Log

#### Summary

Log data on behalf of a client.
Format

PMIx v4.0

```python
def log(args: dict is not None)
```

IN  `args`
Python dictionary containing:

- `'source'`: Python `proc` of requesting process (dict)
- `'data'`: Optional list of Python `info` containing data to be logged (list)
- `'directives'`: Optional list of Python `info` containing directives (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

See `pmix_server_log_fn_t` for details.

A.3.3.18 Allocate Resources

Summary
Request allocation operations on behalf of a client.

Format

PMIx v4.0

```python
def allocate(args: dict is not None)
```

IN  `args`
Python dictionary containing:

- `'source'`: Python `proc` of requesting process (dict)
- `'action'`: Python `allocdir` specifying requested action (integer)
- `'directives'`: Optional list of Python `info` containing directives (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)
- `refarginfo` - List of Python `info` containing results of requested operation (list)

See `pmix_server_alloc_fn_t` for details.

A.3.3.19 Job Control

Summary
Execute a job control action on behalf of a client.
def job_control(args:dict is not None)

IN  args
    Python dictionary containing:
    • 'source': Python proc of requesting process (dict)
    • 'targets': List of Python proc specifying target processes (list)
    • 'directives': Optional list of Python info containing directives (list)

Returns:
    • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See pmix_server_job_control_fn_t for details.

A.3.3.20    Monitor

Summary
    Request that a client be monitored for activity.

def monitor(args:dict is not None)

IN  args
    Python dictionary containing:
    • 'source': Python proc of requesting process (dict)
    • 'monitor': Python info attribute indicating the type of monitor being requested (dict)
    • 'error': Status code to be used when generating an event notification (integer) alerting that
        the monitor has been triggered.
    • 'directives': Optional list of Python info containing directives (list)

Returns:
    • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See pmix_server_monitor_fn_t for details.

A.3.3.21    Get Credential

Summary
    Request a credential from the host environment.
Format

PMIx v4.0

```python
def get_credential(args:dict is not None)
```

**IN** `args`
- Python dictionary containing:
  - 'source': Python `proc` of requesting process (dict)
  - 'directives': Optional list of Python `info` containing directives (list)

Returns:
- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)
- `cred` - Python `byteobject` containing returned credential (dict)
- `info` - List of Python `info` containing any additional info about the credential (list)

See `pmix_server_get_cred_fn_t` for details.

### A.3.3.22 Validate Credential

**Summary**
- Request validation of a credential

Format

PMIx v4.0

```python
def validate_credential(args:dict is not None)
```

**IN** `args`
- Python dictionary containing:
  - 'source': Python `proc` of requesting process (dict)
  - 'credential': Python `byteobject` containing credential (dict)
  - 'directives': Optional list of Python `info` containing directives (list)

Returns:
- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)
- `info` - List of Python `info` containing any additional info from the credential (list)

See `pmix_server_validate_cred_fn_t` for details.
A.3.3.23  IO Forward

**Summary**
Request the specified IO channels be forwarded from the given array of processes.

**Format**

```python
def iof_pull(args: dict is not None)
```

**IN** args
Python dictionary containing:
- 'sources': List of Python `proc` of processes whose IO is being requested (list)
- 'channels': Bitmask of Python `channel` identifying IO channels to be forwarded (integer)
- 'directives': Optional list of Python `info` containing directives (list)

Returns:
- `rc` - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See `pmix_server_iof_fn_t` for details.

A.3.3.24  IO Push

**Summary**
Pass standard input data to the host environment for transmission to specified recipients.

**Format**

```python
def iof_push(args: dict is not None)
```

**IN** args
Python dictionary containing:
- 'source': Python `proc` of process whose input is being forwarded (dict)
- 'payload': Python `byteobject` containing input bytes (dict)
- 'targets': List of `proc` of processes that are to receive the payload (list)
- 'directives': Optional list of Python `info` containing directives (list)

Returns:
- `rc` - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)

See `pmix_server_stdin_fn_t` for details.
A.3.3.25  Group Operations

Summary
Request group operations (construct, destruct, etc.) on behalf of a set of processes.

Format

```python
PMIx v4.0
def group(args:dict is not None)
```

```python
IN
args
``` Python

Python dictionary containing:

- 'op': Operation host is to perform on the specified group (integer)
- 'group': String identifier of target group (str)
- 'procs': List of Python proc of participating processes (dict)
- 'directives': Optional list of Python info containing directives (list)

Returns:

- rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python info containing results of requested operation (list)

See `pmix_server_grp_fn_t` for details.

A.3.3.26  Fabric Operations

Summary
Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

Format

```python
PMIx v4.0
def fabric(args:dict is not None)
```

```python
IN
args
``` Python

Python dictionary containing:

- 'source': Python proc of requesting process (dict)
- 'index': Identifier of the fabric being operated upon (integer)
- 'op': Operation host is to perform on the specified fabric (integer)
- 'directives': Optional list of Python info containing directives (list)

Returns:
- `rc` - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- `refarginfo` - List of Python `info` containing results of requested operation (list)

See `pmix_server_fabric_fn_t` for details.

## A.4 PMIxClient

The client Python class is by far the richest in terms of APIs as it houses all the APIs that an application might utilize. Due to the datatype translation requirements of the C-Python interface, only the blocking form of each API is supported – providing a Python callback function directly to the C interface underlying the bindings was not a supportable option.

### A.4.1 Client.init

**Summary**

Initialize the PMIx client library after obtaining a new PMIxClient object.

**Format**

```
PMIx v4.0

rc, proc = myclient.init(info:list)
```

**IN** `info`  
List of Python `info` dictionaries (list)

**Returns:**

- `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- `proc` - a Python `proc` dictionary (dict)

See `PMIx_Init` for description of all relevant attributes and behaviors.

### A.4.2 Client.initialized

**Format**

```
PMIx v4.0

rc = myclient.initialized()
```

**Returns:**

- `rc` - a value of 1 (true) will be returned if the PMIx library has been initialized, and 0 (false) otherwise (integer)

See `PMIx_Initialized` for description of all relevant attributes and behaviors.
A.4.3  Client.get_version

Format

PMIx v4.0  Python

vers = myclient.get_version()

Returns:

- vers - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)

See PMIx_Get_version for description of all relevant attributes and behaviors.

A.4.4  Client.finalize

Summary

Finalize the PMIx client library.

Format

PMIx v4.0  Python

rc = myclient.finalize(info:list)

IN  info
    List of Python info dictionaries (list)

Returns:

- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Finalize for description of all relevant attributes and behaviors.

A.4.5  Client.abort

Summary

Request that the provided list of processes be aborted.
1  Format
2  PMIx v4.0  Python
3  rc = myclient.abort(status:integer, msg:str, targets:list)

4  IN  status
5  PMIx status to be returned on exit (integer)
6  IN  msg
7  String message to be printed (string)
8  IN  targets
9  List of Python proc dictionaries (list)

Returns:

- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Abort for description of all relevant attributes and behaviors.

A.4.6 Client.store_internal

Summary
Store some data locally for retrieval by other areas of the process

Format
PMIx v4.0  Python
rc = myclient.store_internal(proc:dict, key:str, value:dict)

IN  proc
Python proc dictionary of the process being referenced (dict)
IN  key
String key of the data (string)
IN  value
Python value dictionary (dict)

Returns:

- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Store_internal for details.

A.4.7 Client.put

Summary
Push a key/value pair into the client’s namespace.
Format

```
PMIx v4.0
```

```
rc = myclient.put(scope:integer, key:str, value:dict)
```

```Python

IN scope
Scope of the data being posted (integer)

IN key
String key of the data (string)

IN value
Python value dictionary (dict)
```

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_Put` for description of all relevant attributes and behaviors.

## A.4.8 Client.commit

**Summary**

Push all previously `PMIxClient.put` values to the local PMIx server.

**Format**

```
PMIx v4.0
```

```
rc = myclient.commit()
```

```

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_Commit` for description of all relevant attributes and behaviors.

## A.4.9 Client.fence

**Summary**

Execute a blocking barrier across the processes identified in the specified list.
A.4.10 Client.get

Summary
Retrieve a key/value pair.

Format

rc, val = myclient.get(proc:dict, key:str, directives:list)

IN proc
Python proc whose data is being requested (dict)

IN key
Python string key of the data to be returned (str)

IN directives
List of Python info dictionaries (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

• val - Python value containing the returned data (dict)

See PMIx_Get for description of all relevant attributes and behaviors.

A.4.11 Client.publish

Summary
Publish data for later access via PMIx_Lookup.
Format

PMIx v4.0

rc = myclient.publish(directives:list)

IN directives
    List of Python info dictionaries containing data to be published and directives (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Publish for description of all relevant attributes and behaviors.

A.4.12 Client.lookup

Summary
Lookup information published by this or another process with PMIx_Publish.

Format

PMIx v4.0

rc,info = myclient.lookup(pdata:list, directives:list)

IN pdata
    List of Python pdata dictionaries identifying data to be retrieved (list)

IN directives
    List of Python info dictionaries (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

• info - Python list of info containing the returned data (list)

See PMIx_Lookup for description of all relevant attributes and behaviors.

A.4.13 Client.unpublish

Summary
Delete data published by this process with PMIx_Publish.
Format

```
rc = myclient.unpublish(keys:list, directives:list)
```

**IN**  
`keys`  
List of Python string keys identifying data to be deleted (list)

**IN**  
`directives`  
List of Python `info` dictionaries (list)

Returns:

- `rc` - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See [PMIx Unpublish](#) for description of all relevant attributes and behaviors.

### A.4.14 Client.spawn

**Summary**

Spawn a new job.

Format

```
rc,nspace = myclient.spawn(jobinfo:list, apps:list)
```

**IN**  
`jobinfo`  
List of Python `info` dictionaries (list)

**IN**  
`apps`  
List of Python `app` dictionaries (list)

Returns:

- `rc` - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- `nspace` - Python `nspace` of the new job (dict)

See [PMIx Spawn](#) for description of all relevant attributes and behaviors.

### A.4.15 Client.connect

**Summary**

Connect namespaces.
rc = myclient.connect(peers:list, directives:list)

IN peers
List of Python proc dictionaries (list)

IN directives
List of Python info dictionaries (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Connect for description of all relevant attributes and behaviors.

A.4.16 Client.disconnect

Summary
Disconnect namespaces.

rc = myclient.disconnect(peers:list, directives:list)

IN peers
List of Python proc dictionaries (list)

IN directives
List of Python info dictionaries (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Disconnect for description of all relevant attributes and behaviors.

A.4.17 Client.resolve_peers

Summary
Return list of processes within the specified nspace on the given node.
rc, procs = myclient.resolve_peers(node:str, nspace:str)

**IN**
- **node**
  Name of node whose processes are being requested (str)

**IN**
- **nspace**
  Python nspace whose processes are to be returned (str)

Returns:

- **rc** - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- **procs** - List of Python proc dictionaries (list)

See PMIx Resolve_peers for description of all relevant attributes and behaviors.

### A.4.18 Client.resolve_nodes

**Summary**
Return list of nodes hosting processes within the specified nspace.

**Format**

rc, nodes = myclient.resolve_nodes(nspace:str)

**IN**
- **nspace**
  Python nspace (str)

Returns:

- **rc** - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- **nodes** - List of Python string node names (list)

See PMIx Resolve_nodes for description of all relevant attributes and behaviors.

### A.4.19 Client.query

**Summary**
Query information about the system in general.
Format

PMIx v4.0

rc, info = myclient.query(queries:list)

IN  queries
    List of Python query dictionaries (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
• info - List of Python info containing results of the query (list)

See PMIx_Query_info for description of all relevant attributes and behaviors.

A.4.20 Client.log

Summary
Log data to a central data service/store.

Format

PMIx v4.0

rc = myclient.log(data:list, directives:list)

IN  data
    List of Python info (list)

IN  directives
    Optional list of Python info (list)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_Log for description of all relevant attributes and behaviors.

A.4.21 Client.allocation_request

Summary
Request an allocation operation from the host resource manager.
A.4.22 Client.job_ctrl

Summary
Request a job control action.

Format
```
PMIx v4.0
rc, info = myclient.job_ctrl(targets: list, directives: list)
```

IN  targets
List of Python proc specifying targets of requested operation (integer)

IN  directives
List of Python info describing operation to be performed (list)

Returns:
- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- info - List of Python info containing results of the request (list)

See PMIx_Job_control for description of all relevant attributes and behaviors.

A.4.23 Client.monitor

Summary
Request that something be monitored.
### Client.get_credential

**Summary**
Request a credential from the PMIx server/SMS.

**Format**

```python
rc, cred = myclient.get_credential(directives:list)
```

**IN**

- `directives`:
  Optional list of Python `info` describing request (list)

**Returns:**

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `cred` - Python `byteobject` containing returned credential (dict)

See `PMIx_Get_credential` for description of all relevant attributes and behaviors.

### Client.validate_credential

**Summary**
Request validation of a credential by the PMIx server/SMS.

**Format**

```python
rc, info = myclient.monitor(monitor:dict, error_code:integer, directives:list)
```

**IN**

- `monitor`:
  Python `info` specifying specifying the type of monitor being requested (dict)
- `error_code`:
  Status code to be used when generating an event notification alerting that the monitor has been triggered (integer)
- `directives`:
  List of Python `info` describing request (list)

**Returns:**

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `info` - List of Python `info` containing results of the request (list)

See `PMIx_Process_monitor` for description of all relevant attributes and behaviors.
rc, info = myclient.validate_credential(cred: dict, directives: list)

IN  cred
    Python `byteobject` containing credential (dict)

IN  directives
    Optional list of Python `info` describing request (list)

Returns:
• rc - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
• info - List of Python `info` containing additional results of the request (list)

See `PMIx.Validate.credential` for description of all relevant attributes and behaviors.

A.4.26  Client.group_construct

Summary
Construct a new group composed of the specified processes and identified with the provided group identifier.

rc, info = myclient.construct_group(grp: string, members: list, directives: list)

IN  grp
    Python string identifier for the group (str)

IN  members
    List of Python `proc` dictionaries identifying group members (list)

IN  directives
    Optional list of Python `info` describing request (list)

Returns:
• rc - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
• info - List of Python `info` containing results of the request (list)

See `PMIx.Group.construct` for description of all relevant attributes and behaviors.
A.4.27 Client.group_invite

**Summary**
Explicitly invite specified processes to join a group.

**Format**

```python
PMIx v4.0
rc, info = myclient.group_invite(grp:string,
    members:list, directives:list)
```

**IN**  
- `grp`  
  Python string identifier for the group (str)

**IN**  
- `members`  
  List of Python `proc` dictionaries identifying processes to be invited (list)

**IN**  
- `directives`  
  Optional list of Python `info` describing request (list)

*Returns:*

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `info` - List of Python `info` containing results of the request (list)

See `PMIx_Group_invite` for description of all relevant attributes and behaviors.

A.4.28 Client.group_join

**Summary**
Respond to an invitation to join a group that is being asynchronously constructed.

**Format**

```python
PMIx v4.0
rc, info = myclient.group_join(grp:string,
    leader:dict, opt:integer,
    directives:list)
```

**IN**  
- `grp`  
  Python string identifier for the group (str)

**IN**  
- `leader`  
  Python `proc` dictionary identifying process leading the group (dict)

**IN**  
- `opt`  
  One of the `pmix_group_opt_t` values indicating decline/accept (integer)

**IN**  
- `directives`  
  Optional list of Python `info` describing request (list)
Returns:
- \texttt{rc} - \texttt{PMIX_SUCCESS} or a negative value corresponding to a PMIx error constant (integer)
- \texttt{info} - List of Python \texttt{info} containing results of the request (list)

See \texttt{PMIx\_Group\_join} for description of all relevant attributes and behaviors.

### A.4.29 Client.group.leave

**Summary**

Leave a PMIx Group.

**Format**

\begin{verbatim}
PMIx v4.0
rc = myclient.group_leave(grp:string, directives:list)
\end{verbatim}

**IN** \texttt{grp}
Python string identifier for the group (str)

**IN** \texttt{directives}
Optional list of Python \texttt{info} describing request (list)

Returns:
- \texttt{rc} - \texttt{PMIX_SUCCESS} or a negative value corresponding to a PMIx error constant (integer)

See \texttt{PMIx\_Group\_leave} for description of all relevant attributes and behaviors.

### A.4.30 Client.group.destruct

**Summary**

Destruct a PMIx Group.

**Format**

\begin{verbatim}
PMIx v4.0
rc = myclient.group_destruct(grp:string, directives:list)
\end{verbatim}

**IN** \texttt{grp}
Python string identifier for the group (str)

**IN** \texttt{directives}
Optional list of Python \texttt{info} describing request (list)

Returns:
- \texttt{rc} - \texttt{PMIX_SUCCESS} or a negative value corresponding to a PMIx error constant (integer)

See \texttt{PMIx\_Group\_destruct} for description of all relevant attributes and behaviors.
A.4.31  **Client.register_event_handler**

**Summary**
Register an event handler to report events.

**Format**

```python
rc, id = myclient.register_event_handler(codes:list,
                                        directives:list, cbfunc)
```

- **codes**
  - List of Python integer status codes that should be reported to this handler (list)
- **directives**
  - Optional list of Python info describing request (list)
- **cbfunc**
  - Python evhandler to be called when event is received (func)

**Returns:**
- `rc` - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- `id` - PMIx reference identifier for handler (integer)

See **PMIx_Register_event_handler** for description of all relevant attributes and behaviors.

A.4.32  **Client.deregister_event_handler**

**Summary**
Deregister an event handler.

**Format**

```python
myclient.deregister_event_handler(id:integer)
```

- **id**
  - PMIx reference identifier for handler (integer)

**Returns:** None

See **PMIx_Deregister_event_handler** for description of all relevant attributes and behaviors.
A.4.33 Client.notify_event

Summary
Report an event for notification via any registered handler.

Format

```
PMIx v4.0
rc = myclient.notify_event(status:integer, source:dict,
range:integer, directives:list)
```

IN  `status`
PMIx status code indicating the event being reported (integer)

IN  `source`
Python `proc` of the process that generated the event (dict)

IN  `range`
Python `range` in which the event is to be reported (integer)

IN  `directives`
Optional list of Python `info` dictionaries describing the event (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_Notify_event` for description of all relevant attributes and behaviors.

A.4.34 Client.fabric_register

Summary
Register for access to fabric-related information, including communication cost matrix.

Format

```
PMIx v4.0
rc,idx,fabricinfo = myclient.fabric_register(directives:list)
```

IN  `directives`
Optional list of Python `info` containing directives (list)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `idx` - Index of the registered fabric (integer)
- `fabricinfo` - List of Python `info` containing fabric info (list)

See `PMIx_Fabric_register` for details.
A.4.35 Client.fabric_update

**Summary**
Update fabric-related information, including communication cost matrix.

**Format**

```python
PMIx v4.0
rc, fabricinfo = myclient.fabric_update(idx:integer)
```

**IN**
idx
Index of the registered fabric (list)

Returns:
- `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- `fabricinfo` - List of Python **info** containing updated fabric info (list)

See **PMIx_Fabric_update** for details.

A.4.36 Client.fabric_deregister

**Summary**
Deregister fabric.

**Format**

```python
PMIx v4.0
rc = myclient.fabric_deregister(idx:integer)
```

**IN**
idx
Index of the registered fabric (list)

Returns:
- `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

See **PMIx_Fabric_deregister** for details.

A.4.37 Client.load_topology

**Summary**
Load the local hardware topology into the PMIx library.
A.4.38 Client.get_relative_locality

**Summary**
Get the relative locality of two local processes.

**Format**
```
rc, locality = myclient.get_relative_locality(loc1:str, loc2:str)
```

<table>
<thead>
<tr>
<th>IN</th>
<th>loc1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Locality string of a process (str)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN</th>
<th>loc2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Locality string of a process (str)</td>
</tr>
</tbody>
</table>

**Returns:**
- `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- `locality` - list containing the relative locality of the two processes (list)

See **PMIx_Get_relative_locality** for details.

A.4.39 Client.get_cpuset

**Summary**
Get the PU binding bitmap of the current process.
A.40 Client.parse_cpuset_string

Summary
Parse the PU binding bitmap from its string representation.

Format

rc, cpuset = myclient.parse_cpuset_string(cpuset:string)

IN  cpuset
String returned by PMIxServer.generate_cpuset_string (string)

Returns:
• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
• cpuset - cpuset containing the source and bitmap of the cpuset (dict)

See PMIx_Parse_cpuset_string for details.

A.41 Client.compute_distances

Summary
Compute distances from specified process location to local devices.
rc, distances = myclient.compute_distances(cpuset: dict, info: list)

IN cpuset
describing the location of the process (dict)

IN info
List of info dictionaries describing the devices whose distance is to be computed (list)

Returns:
- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- distances - List of devdist structures containing the distances from the caller to the specified devices (list)

See PMIx_Compute_distances for details. Note that distances can only be computed against the local topology.

A.4.42 Client.error_string

Summary
Pretty-print string representation of pmix_status_t.

Format

rep = myclient.error_string(status: integer)

IN status
PMIx status code (integer)

Returns:
- rep - String representation of the provided status code (str)

See PMIx_Error_string for further details.

A.4.43 Client.proc_state_string

Summary
Pretty-print string representation of pmix_proc_state_t.
**A.4.44 Client.scope_string**

**Summary**
Pretty-print string representation of `pmix_scope_t`.

**Format**

```python
rep = myclient.scope_string(scope:integer)
```

**IN**

PMIx scope value (integer)

**Returns:**

- `rep` - String representation of the provided scope (str)

See `PMIx_Scope_string` for further details.

**A.4.45 Client.persistence_string**

**Summary**
Pretty-print string representation of `pmix_persistence_t`.

**Format**

```python
rep = myclient.persistence_string(persistence:integer)
```

**IN**

PMIx persistence value (integer)

**Returns:**

- `rep` - String representation of the provided persistence (str)

See `PMIx_Persistence_string` for further details.
A.4.46 Client.data_range_string

Summary
Pretty-print string representation of \texttt{pmix\_data\_range\_t}.

Format

\begin{verbatim}
PMIx v4.0
rep = myclient.data_range_string(range:integer)
\end{verbatim}

\textbf{IN} \hspace{1cm} \texttt{range}
PMIx data range value (integer)

Returns:
- \texttt{rep} - String representation of the provided data range (str)

See \texttt{PMIx\_Data\_range\_string} for further details.

A.4.47 Client.info_directives_string

Summary
Pretty-print string representation of \texttt{pmix\_info\_directives\_t}.

Format

\begin{verbatim}
PMIx v4.0
rep = myclient.info_directives_string(directives:bitarray)
\end{verbatim}

\textbf{IN} \hspace{1cm} \texttt{directives}
PMIx \texttt{info directives} value (bitarray)

Returns:
- \texttt{rep} - String representation of the provided info directives (str)

See \texttt{PMIx\_Info\_directives\_string} for further details.

A.4.48 Client.data_type_string

Summary
Pretty-print string representation of \texttt{pmix\_data\_type\_t}. 

PMIx Standard – Version 4.0 – December 2020
1 Format

PMIx v4.0 Python

2 rep = myclient.data_type_string(dtype:integer)

IN dtype
PMIx datatype value (integer)

Returns:

• rep - String representation of the provided datatype (str)

See PMIx_Data_type_string for further details.

A.4.49 Client.allocDirective_string

Summary
Pretty-print string representation of pmix_alloc_directive_t.

Format

PMIx v4.0 Python

rep = myclient.alloc_directive_string(adir:integer)

IN adir
PMIx allocation directive value (integer)

Returns:

• rep - String representation of the provided allocation directive (str)

See PMIx_Alloc_directive_string for further details.

A.4.50 Client.iof_channel_string

Summary
Pretty-print string representation of pmix_iof_channel_t.

Format

PMIx v4.0 Python

rep = myclient.iof_channel_string(channel:bitarray)

IN channel
PMIx IOF channel value (bitarray)

Returns:

• rep - String representation of the provided IOF channel (str)

See PMIx_IOF_channel_string for further details.
A.4.51 Client.job_state_string

Summary
Pretty-print string representation of `pmix_job_state_t`.

Format

```
PMIx v4.0
rep = myclient.job_state_string(state:integer)
```

\textbf{IN} state
PMIx job state value (integer)

Returns:

\begin{itemize}
\item \textit{rep} - String representation of the provided job state (str)
\end{itemize}

See \texttt{PMIx_Job_state_string} for further details.

A.4.52 Client.get_attribute_string

Summary
Pretty-print string representation of a PMIx attribute.

Format

```
PMIx v4.0
rep = myclient.get_attribute_string(attribute:str)
```

\textbf{IN} attribute
PMIx attribute name (string)

Returns:

\begin{itemize}
\item \textit{rep} - String representation of the provided attribute (str)
\end{itemize}

See \texttt{PMIx_Get_attribute_string} for further details.

A.4.53 Client.get_attribute_name

Summary
Pretty-print name of a PMIx attribute corresponding to the provided string.
A.4.54 Client.link_state_string

Summary
Pretty-print string representation of \texttt{pmix_link_state_t}.

Format

\begin{verbatim}
PMIx v4.0
rep = myclient.link_state_string(state:integer)
\end{verbatim}

\begin{itemize}
\item \texttt{state} - PMIx link state value (integer)
\end{itemize}

Returns:

\begin{itemize}
\item \texttt{rep} - String representation of the provided link state (str)
\end{itemize}

See \texttt{PMIx_Link_state_string} for further details.

A.4.55 Client.device_type_string

Summary
Pretty-print string representation of \texttt{pmix_device_type_t}.

Format

\begin{verbatim}
PMIx v4.0
rep = myclient.device_type_string(type:bitarray)
\end{verbatim}

\begin{itemize}
\item \texttt{type} - PMIx device type value (bitarray)
\end{itemize}

Returns:

\begin{itemize}
\item \texttt{rep} - String representation of the provided device type (str)
\end{itemize}

See \texttt{PMIx_Device_type_string} for further details.
A.4.56 Client.progress

Summary
Progress the PMIx library.

Format
PMIx v4.0
myclient.progress()

Python
See PMIx_Progress for further details.

A.5 PMIxServer

The server Python class inherits the Python "client" class as its parent. Thus, it includes all client functions in addition to the ones defined in this section.

A.5.1 Server.init

Summary
Initialize the PMIx server library after obtaining a new PMIxServer object.

Format
PMIx v4.0
rc = myserver.init(directives:list, map:dict)

Python
IN directives
List of Python info dictionaries (list)

IN map
Python dictionary key-function pairs that map server module callback functions to provided implementations (see pmix_server_module_t) (dict)

Returns:
• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_server_init for description of all relevant attributes and behaviors.

A.5.2 Server.finalize

Summary
Finalize the PMIx server library.
**A.5.3 Server.generate_regex**

**Summary**
Generate a regular expression representation of the input strings.

**Format**

```plaintext
rc, regex = myserver.generate_regex(input:list)
```

**IN**
- `input` : List of Python strings (e.g., node names) (list)

**Returns:**
- `rc` : PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- `regex` : Python `bytearray` containing regular expression representation of the input list (`bytearray`)

See **PMIx_generate_regex** for details.

**A.5.4 Server.generate_ppn**

**Summary**
Generate a regular expression representation of the input strings.
Format

PMIx v4.0

\[
\text{rc, regex} = \text{myserver.generate_ppn}(\text{input:list})
\]

IN  \text{input}

List of Python strings, each string consisting of a comma-delimited list of ranks on each node, with the strings being in the same order as the node names provided to "generate_regex" (list)

Returns:

• \text{rc} - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

• \text{regex} - Python bytearray containing regular expression representation of the input list (bytearray)

See PMIx_generate_ppn for details.

A.5.5 Server.generate_locality_string

Summary

Generate a PMIx locality string from a given cpuset.

Format

PMIx v4.0

\[
\text{rc, locality} = \text{myserver.generate_locality_string}(\text{cpuset:dict})
\]

IN  \text{cpuset}

containing the bitmap of assigned PUs (dict)

Returns:

• \text{rc} - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

• \text{locality} - String representation of the PMIx locality corresponding to the input bitmap (string)

See PMIx_server_generate_locality_string for details.

A.5.6 Server.generate_cpuset_string

Summary

Generate a PMIx string representation of the provided cpuset.
rc, cpustr = myserver.generate_cpuset_string(cpuset=dict)

IN cset
  cpuset containing the bitmap of assigned PUs (dict)

Returns:
- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- cpustr - String representation of the input bitmap (string)

See PMIx_server_generate_cpuset_string for details.

A.5.7 Server.register_nspace

Summary
Setup the data about a particular namespace.

rc = myserver.register_nspace(nspace=str,
  nlocalprocs=integer,
  directives=list)

IN nspace
  Python string containing the namespace (str)
IN nlocalprocs
  Number of local processes (integer)
IN directives
  List of Python info dictionaries (list)

Returns:
- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_server_register_nspace for description of all relevant attributes and behaviors.

A.5.8 Server.deregister_nspace

Summary
Deregister a namespace.
A.5.9 Server.register_resources

**Summary**
Register non-namespace related information with the local PMIx library

**Format**

```python
myserver.register_resources(directives:list)
```

**IN**
- `directives`: List of Python `info` dictionaries (list)

**Returns**: None

See `PMIx_server_register_resources` for details.

A.5.10 Server.deregister_resources

**Summary**
Remove non-namespace related information from the local PMIx library

**Format**

```python
myserver.deregister_resources(directives:list)
```

**IN**
- `directives`: List of Python `info` dictionaries (list)

**Returns**: None

See `PMIx_server_deregister_resources` for details.
A.5.11 Server.register_client

Summary
Register a client process with the PMIx server library.

Format

\[ rc = \text{myserver.register_client}(\text{proc:dict}, \text{uid:integer}, \text{gid:integer}) \]

IN\ proc
Python proc dictionary identifying the client process (dict)

IN\ uid
Linux uid value for user executing client process (integer)

IN\ gid
Linux gid value for user executing client process (integer)

Returns:
- \( rc \) - \text{PMIX_SUCCESS} or a negative value corresponding to a PMIx error constant (integer)

See \text{PMIx_server_register_client} for details.

A.5.12 Server.deregister_client

Summary
Deregister a client process and purge all data relating to it.

Format

\[ \text{myserver.deregister_client}(\text{proc:dict}) \]

IN\ proc
Python proc dictionary identifying the client process (dict)

Returns: None

See \text{PMIx_server_deregister_client} for details.

A.5.13 Server.setup_fork

Summary
Setup the environment of a child process that is to be forked by the host.
**A.5.14 Server.dmodex_request**

**Summary**
Function by which the host server can request modex data from the local PMIx server.

**Format**

```plaintext
rc, data = myserver.dmodex_request(proc:dict)  
```

**IN**  
Python `proc` dictionary identifying the process whose data is requested (dict)

**Returns:**
- `rc` - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- `data` - Python `byteobject` containing the returned data (dict)

See PMIx_server_dmodex_request for details.

**A.5.15 Server.setup_application**

**Summary**
Function by which the resource manager can request application-specific setup data prior to launch of a job.
A.5.16 Server.register_attributes

**Summary**
Register host environment attribute support for a function.

**Format**

```
PMIx v4.0
rc = myserver.register_attributes(function:str, attrs:list)
```

**IN**
- `function` (str): Name of the function

**IN**
- `attrs` (list of regattr): Python list of regattr describing the supported attributes

**Returns:**
- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `attrs` - Python list of regattr dictionaries containing the returned data (list)

See `PMIx_Register_attributes` for details.

A.5.17 Server.setup_local_support

**Summary**
Function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.
rc = myserver.setup_local_support(nspace:str, info:list)

IN  nspace
    Namespace whose setup information is being requested (str)

IN  info
    Python list of info containing the setup data (list)

Returns:

- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_server_setup_local_support for details.

A.5.18 Server.iof_deliver

Summary
Function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

rc = myserver.iof_deliver(source:dict, channel:integer, data:dict, directives:list)

IN  source
    Python proc dictionary identifying the process who generated the data (dict)

IN  channel
    Python channel bitmask identifying IO channel of the provided data (integer)

IN  data
    Python byteobject containing the data (dict)

IN  directives
    Python list of info containing directives (list)

Returns:

- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_server_IOF_deliver for details.

A.5.19 Server.collect_inventory

Summary
Collect inventory of resources on a node.
**A.5.20 Server.deliver_inventory**

**Summary**
Pass collected inventory to the PMIx server library for storage.

**Format**

```python
PMIx v4.0
rc = myserver.deliver_inventory(info:list, directives:list)
```

**IN**
- `info` - Python list of `info` dictionaries containing the inventory data (list)
- `directives` - Python list of `info` dictionaries containing directives (list)

**Returns:**
- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `info` - Python list of `info` containing the returned data (list)

See `PMIx_server_deliver_inventory` for details.

**A.5.21 Server.define_process_set**

**Summary**
Add members to a PMIx process set.

**Format**

```python
PMIx v4.0
rc, info = myserver.collect_inventory(directives:list)
```

**IN**
- `directives` - Optional Python list of `info` containing directives (list)

**Returns:**
- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `info` - Python list of `info` containing the returned data (list)

See `PMIx_server_collect_inventory` for details.
Format

```python
rc = myserver.define_process_set(members:list, name:str)
```

**IN** `members`
- List of Python `proc` dictionaries identifying the processes to be added to the process set (list)

**IN** `name`
- Name of the process set (str)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_server_define_process_set` for details.

### A.5.22 Server.delete_process_set

**Summary**
Delete a PMIx process set.

```
rc = myserver.delete_process_set(name:str)
```

**IN** `name`
- Name of the process set (str)

Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_server_delete_process_set` for details.

### A.5.23 Server.register_resources

**Summary**
Register non-namespace related information with the local PMIx server library.
A.5.24 Server.deregister_resources

Summary
Deregister non-namespace related information with the local PMIx server library.

Format

```python
rc = myserver.deregister_resources(info:list)
```

IN info
- List of Python info dictionaries list)

Returns:
- `rc` - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See `PMIx_server_deregister_resources` for details.

A.6 PMIxTool

The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and server functions in addition to the ones defined in this section.

A.6.1 Tool.init

Summary
Initialize the PMIx tool library after obtaining a new PMIxTool object.
A.6.2 Tool.finalize

Summary
Finalize the PMIx tool library, closing the connection to the server.

Format

rc = mytool.finalize()

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_tool_final for description of all relevant attributes and behaviors.

A.6.3 Tool.disconnect

Summary
Disconnect the PMIx tool from the specified server connection while leaving the tool library initialized.

Format

rc = mytool.disconnect(server=dict)

IN server
Process identifier of server from which the tool is to be disconnected (proc)

Returns:

• rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_tool_disconnect for details.
A.6.4 Tool.attach_to_server

Summary
Establish a connection to a PMIx server.

Format

PMIx v4.0
rc, proc, server = mytool.connect_to_server(info:list)

IN info
List of Python info dictionaries (list)

Returns:
- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- proc - a Python proc containing the tool’s identifier (dict)
- server - a Python proc containing the identifier of the server to which the tool attached (dict)

See PMIx_tool_attach_to_server for details.

A.6.5 Tool.get_servers

Summary
Get a list containing the proc process identifiers of all servers to which the tool is currently connected.

Format

PMIx v4.0
rc, servers = mytool.get_servers()

Returns:
- rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- servers - a list of Python proc containing the identifiers of the servers to which the tool is currently attached (dict)

See PMIx_tool_get_servers for details.

A.6.6 Tool.set_server

Summary
Designate a server as the tool’s primary server.
**A.6.7 Tool.iof_pull**

**Summary**
Register to receive output forwarded from a remote process.

**Format**

```
PMIx v4.0
rc, id = mytool.iof_pull(sources:list, channel: integer,
directives:list, cbfunc)
```

### IN sources
List of Python `proc` dictionaries of processes whose IO is being requested (list)

### IN channel
Python `channel` bitmask identifying IO channels to be forwarded (integer)

### IN directives
List of Python `info` dictionaries describing request (list)

### IN cbfunc
Python `iofcbfunc` to receive IO payloads (func)

**Returns:**

- `rc` - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
- `id` - PMIx reference identifier for request (integer)

See **PMIx_IOF_pull** for description of all relevant attributes and behaviors.
A.6.8 Tool.iof_deregister

**Summary**

Deregister from output forwarded from a remote process.

**Format**

```plaintext
PMIx v4.0

rc = mytool.iof_deregister(id:integer, directives:list)
```

**IN**  
- `id`: PMIx reference identifier returned by pull request (list)
- `directives`: List of Python `info` dictionaries describing request (list)

**Returns:**

- `rc`: PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_IOF_deregister for description of all relevant attributes and behaviors.

A.6.9 Tool.iof_push

**Summary**

Push data collected locally (typically from stdin) to stdin of target recipients.

**Format**

```plaintext
PMIx v4.0

rc = mytool.iof_push(targets:list, data:dict, directives:list)
```

**IN**  
- `sources`: List of Python `proc` of target processes (list)
- `data`: Python `byteobject` containing data to be delivered (dict)
- `directives`: Optional list of Python `info` describing request (list)

**Returns:**

- `rc`: PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

See PMIx_IOF_push for description of all relevant attributes and behaviors.

A.7 Example Usage

The following examples are provided to illustrate the use of the Python bindings.
A.7.1 Python Client

The following example contains a client program that illustrates a fairly common usage pattern. The program instantiates and initializes the PMIxClient class, posts some data that is to be shared across all processes in the job, executes a “fence” that circulates the data, and then retrieves a value posted by one of its peers. Note that the example has been formatted to fit the document layout.

```python
from pmix import *

def main():
    # Instantiate a client object
    myclient = PMIxClient()
    print("Testing PMIx ", myclient.get_version())

    # Initialize the PMIx client library, declaring the programming model
    # as "TEST" and the library name as "PMIX", just for the example
    info = [  
        {'key':PMIX_PROGRAMMING_MODEL,  
        'value':"TEST", 'val_type':PMIX_STRING,  
        'key':PMIX_MODEL_LIBRARY_NAME,  
        'value':"PMIX", 'val_type':PMIX_STRING]

    rc,myname = myclient.init(info)
    if PMIX_SUCCESS != rc:
        print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
        exit(1)

    # try posting a value
    rc = myclient.put(PMIX_GLOBAL, "mykey",  
                      'value':1, 'val_type':PMIX_INT32)
    if PMIX_SUCCESS != rc:
        print("PMIx_Put FAILED WITH ERROR", myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)

    # commit it
    rc = myclient.commit()
    if PMIX_SUCCESS != rc:
        print("PMIx_Commit FAILED WITH ERROR",  
               myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)

```

PMIx Standard – Version 4.0 – December 2020
```python
# execute fence across all processes in my job
procs = []
info = []
rc = myclient.fence(procs, info)
if PMIX_SUCCESS != rc:
    print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
    # cleanly finalize
    myclient.finalize()
    exit(1)

# Get a value from a peer
if 0 != myname['rank']:
    info = []
    rc, get_val = myclient.get('nspace':"testnspace", 'rank': 0,
    "mykey", info)
    if PMIX_SUCCESS != rc:
        print("PMIx_Commit FAILED WITH ERROR",
        myclient.error_string(rc))
        # cleanly finalize
        myclient.finalize()
        exit(1)
    print("Get value returned: ", get_val)

# test a fence that should return not_supported because
# we pass a required attribute that the server is known
# not to support
procs = []
info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
    'value':10, 'val_type':PMIX_INT]
rc = myclient.fence(procs, info)
if PMIX_SUCCESS == rc:
    print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
    # cleanly finalize
    myclient.finalize()
    exit(1)

# Publish something
info = ['key': 'ARBITRARY', 'value':10, 'val_type':PMIX_INT]
rc = myclient.publish(info)
if PMIX_SUCCESS != rc:
    print("PMIx_Publish FAILED WITH ERROR",
    myclient.error_string(rc))
    # cleanly finalize
```

APPENDIX A. PYTHON BINDINGS
519
```python
myclient.finalize()
exit(1)

# finalize
info = []
myclient.finalize(info)
print("Client finalize complete")

# Python main program entry point
if __name__ == '__main__':
    main()
```

### A.7.2 Python Server

The following example contains a minimum-level server host program that instantiates and initializes the PMIxServer class. The program illustrates passing several server module functions to the bindings and includes code to setup and spawn a simple client application, waiting until the spawned client terminates before finalizing and exiting itself. Note that the example has been formatted to fit the document layout.

```python
from pmix import *
import signal, time
import os
import select
import subprocess

def clientconnected(proc:tuple is not None):
    print("CLIENT CONNECTED", proc)
    return PMIX_OPERATION_SUCCEEDED

def clientfinalized(proc:tuple is not None):
    print("CLIENT FINALIZED", proc)
    return PMIX_OPERATION_SUCCEEDED

def clientfence(procs:list, directives:list, data:bytearray):
    # check directives
    if directives is not None:
        for d in directives:
            # these are each an info dict
            if "pmix" not in d['key']:
                # we do not support such directives - see if
```
# it is required
try:
    if d['flags'] & PMIX_INFO_REQD:
        # return an error
        return PMIX_ERR_NOT_SUPPORTED
except:
    # it can be ignored
    pass
return PMIX_OPERATION_SUCCEEDED

def main():
    try:
        myserver = PMIxServer()
    except:
        print("FAILED TO CREATE SERVER")
        exit(1)
    print("Testing server version ", myserver.get_version())

    args = [{'key':PMIX_SERVER_SCHEDULER,
             'value':'T', 'val_type':PMIX_BOOL}
            map = {'clientconnected': clientconnected,
                   'clientfinalized': clientfinalized,
                   'fencenb': clientfence}
    my_result = myserver.init(args, map)

    # get our environment as a base
    env = os.environ.copy()

    # register an nspace for the client app
    (rc, regex) = myserver.generate_regex("test000,test001,test002")
    (rc, ppn) = myserver.generate_ppn("0")
    kvals = [{'key':PMIX_NODE_MAP,
              'value':regex, 'val_type':PMIX_STRING,
              'key':PMIX_PROC_MAP,
              'value':ppn, 'val_type':PMIX_STRING,
              'key':PMIX_UNIV_SIZE,
              'value':1, 'val_type':PMIX_UINT32,
              'key':PMIX_JOB_SIZE,
              'value':1, 'val_type':PMIX_UINT32}
    rc = foo.register_nspace("testnspace", 1, kvals)
    print("RegNspace ", rc)

    # register a client
uid = os.getuid()
gid = os.getgid()
rc = myserver.register_client('nspace':"testnspace", 'rank':0,
        uid, gid)
print("RegClient ", rc)
# setup the fork
rc = myserver.setup_fork('nspace':"testnspace", 'rank':0, env)
print("SetupFrk", rc)

# setup the client argv
args = ['./client.py']
# open a subprocess with stdout and stderr
# as distinct pipes so we can capture their
# output as the process runs
p = subprocess.Popen(args, env=env,
        stdout=subprocess.PIPE, stderr=subprocess.PIPE)
# define storage to catch the output
stdout = []
stderr = []
# loop until the pipes close
while True:
    reads = [p.stdout.fileno(), p.stderr.fileno()]
    ret = select.select(reads, [], [])

    stdout_done = True
    stderr_done = True

    for fd in ret[0]:
        # if the data
        if fd == p.stdout.fileno():
            read = p.stdout.readline()
            if read:
                read = read.decode('utf-8').rstrip()
                print('stdout: ' + read)
                stdout_done = False
        elif fd == p.stderr.fileno():
            read = p.stderr.readline()
            if read:
                read = read.decode('utf-8').rstrip()
                print('stderr: ' + read)
                stderr_done = False

    if stdout_done and stderr_done:
break
print("FINALIZING")
myserver.finalize()

if __name__ == '__main__':
    main()
B.1 Version 1.0: June 12, 2015

The PMIx version 1.0 ad hoc standard was defined in a set of header files as part of the v1.0.0 release of the OpenPMIx library prior to the creation of the formal PMIx 2.0 standard. Below are a summary listing of the interfaces defined in the 1.0 headers.

- **Client APIs**
  - PMIx_Init, PMIx_Initialized, PMIx_Abort, PMIx_Finalize
  - PMIx_Put, PMIx_Commit
  - PMIx_Fence, PMIx_Fence_nb
  - PMIx_Get, PMIx_Get_nb
  - PMIx_Publish, PMIx_Publish_nb
  - PMIx_Lookup, PMIx_Lookup_nb
  - PMIx_Unpublish, PMIx_Unpublish_nb
  - PMIx_Spawn, PMIx_Spawn_nb
  - PMIx_Connect, PMIx_Connect_nb
  - PMIx_Disconnect, PMIx_Disconnect_nb
  - PMIx_Resolve_nodes, PMIx_Resolve_peers

- **Server APIs**
  - PMIx_server_init, PMIx_server_finalize
  - PMIx_generate_regex, PMIx_generate_ppn
  - PMIx_server_register_nspace, PMIx_server_deregister_nspace
  - PMIx_server_register_client, PMIx_server_deregister_client
  - PMIx_server_setup_fork, PMIx_server_dmodex_request

- **Common APIs**
  - PMIx_Get_version, PMIx_Store_internal, PMIx_Error_string
  - PMIx_Register_errhandler, PMIx_Deregister_errhandler, PMIx_Notify_error

The **PMIx_Init** API was subsequently modified in the v1.1.0 release of that library.
The following APIs were introduced in v2.0 of the PMIx Standard:

- **Client APIs**
  - PMIx_Query_info_nb, PMIx_Log_nb
  - PMIx_Allocation_request_nb, PMIx_Job_control_nb,
    PMIx_Process_monitor_nb, PMIx_Heartbeat

- **Server APIs**
  - PMIx_server_setup_application, PMIx_server_setup_local_support

- **Tool APIs**
  - PMIx_tool_init, PMIx_tool_finalize

- **Common APIs**
  - PMIx_Register_event_handler, PMIx_Deregister_event_handler
  - PMIx_Notify_event
  - PMIx_Proc_state_string, PMIx_Scope_string
  - PMIx_Persistence_string, PMIx_Data_range_string
  - PMIx_Info_directives_string, PMIx_Data_type_string
  - PMIx_Alloc_directive_string
  - PMIx_Data_pack, PMIx_Data_unpack, PMIx_Data_copy
  - PMIx_Data_print, PMIx_Data_copy_payload

### B.2.1 Removed/Modified APIs

The PMIx_Init API was modified in v2.0 of the standard from its *ad hoc* v1.0 signature to include passing of a pmix_info_t array for flexibility and “future-proofing” of the API. In addition, the PMIx_Notify_error, PMIx_Register_errhandler, and PMIx_Deregister_errhandler APIs were replaced. This pre-dated official adoption of PMIx as a Standard.

### B.2.2 Deprecated constants

The following constants were deprecated in v2.0:

- PMIX_MODEX
- PMIX_INFO_ARRAY
B.2.3 Deprecated attributes

The following attributes were deprecated in v2.0:

- `PMIX_ERROR_NAME` "pmix.errname" (pmix_status_t)
  Specific error to be notified
- `PMIX_ERROR_GROUP_COMM` "pmix.errgroup.comm" (bool)
  Set true to get comm errors notification
- `PMIX_ERROR_GROUP_ABORT` "pmix.errgroup.abort" (bool)
  Set true to get abort errors notification
- `PMIX_ERROR_GROUP_MIGRATE` "pmix.errgroup.migrate" (bool)
  Set true to get migrate errors notification
- `PMIX_ERROR_GROUP_RESOURCE` "pmix.errgroup.resource" (bool)
  Set true to get resource errors notification
- `PMIX_ERROR_GROUP_SPAWN` "pmix.errgroup.spawn" (bool)
  Set true to get spawn errors notification
- `PMIX_ERROR_GROUP_NODE` "pmix.errgroup.node" (bool)
  Set true to get node status notification
- `PMIX_ERROR_GROUP_LOCAL` "pmix.errgroup.local" (bool)
  Set true to get local errors notification
- `PMIX_ERROR_GROUP_GENERAL` "pmix.errgroup.gen" (bool)
  Set true to get notified of generic errors
- `PMIX_ERROR_HANDLER_ID` "pmix.errhandler.id" (int)
  Errhandler reference id of notification being reported

B.3 Version 2.1: Dec. 2018

The v2.1 update includes clarifications and corrections from the v2.0 document, plus addition of examples:

- Clarify description of `PMIx_Connect` and `PMIx_Disconnect` APIs.
- Explain that values for the `PMIX_COLLECTIVE_ALGO` are environment-dependent
- Identify the namespace/rank values required for retrieving attribute-associated information using the `PMIx_Get` API
- Provide definitions for `session`, `job`, `application`, and other terms used throughout the document
- Clarify definitions of `PMIX_UNIV_SIZE` versus `PMIX_JOB_SIZE`
- Clarify server module function return values
- Provide examples of the use of `PMIx_Get` for retrieval of information
- Clarify the use of `PMIx_Get` versus `PMIx_Query_info_nb`
- Clarify return values for non-blocking APIs and emphasize that callback functions must not be invoked prior to return from the API
- Provide detailed example for construction of the `PMIx_server_register_nspace` input information array
• Define information levels (e.g., session vs job) and associated attributes for both storing and retrieving values
• Clarify roles of PMIx server library and host environment for collective operations
• Clarify definition of PMIX_UNIV_SIZE

B.4 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

B.5 Version 3.0: Dec. 2018

The following APIs were introduced in v3.0 of the PMIx Standard:

• Client APIs
  - PMIx_Log, PMIx_Job_control
  - PMIx_Allocation_request, PMIx_Process_monitor
  - PMIx_Get_credential, PMIx.Validate.credential

• Server APIs
  - PMIx_server_IOF_deliver
  - PMIx_server_collect_inventory, PMIx_server_deliver_inventory

• Tool APIs
  - PMIx_IOF_pull, PMIx_IOF_push, PMIx_IOF_deregister
  - PMIx_tool_connect_to_server

• Common APIs
  - PMIx_IOF_channel_string
The document added a chapter on security credentials, a new section for 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.

**B.5.1 Removed constants**

The following constants were removed in v3.0:

- PMIX_MODEX
- PMIX_INFO_ARRAY

**B.5.2 Deprecated attributes**

The following attributes were deprecated in v3.0:

- PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)
  
  If true, indicates that the requested choice of algorithm is mandatory.

- PMIX_ERROR_NAME "pmix.errname" (pmix_status_t)
  
  Specific error to be notified

- PMIX_ERROR_GROUP_COMM "pmix.errgroup.comm" (bool)
  
  Set true to get comm errors notification

- PMIX_ERROR_GROUP_ABORT "pmix.errgroup.abort" (bool)
  
  Set true to get abort errors notification

- PMIX_ERROR_GROUP_MIGRATE "pmix.errgroup.migrate" (bool)
  
  Set true to get migrate errors notification

- PMIX_ERROR_GROUP_RESOURCE "pmix.errgroup.resource" (bool)
  
  Set true to get resource errors notification

- PMIX_ERROR_GROUP_SPAWN "pmix.errgroup.spawn" (bool)
  
  Set true to get spawn errors notification

- PMIX_ERROR_GROUP_NODE "pmix.errgroup.node" (bool)
  
  Set true to get node status notification

- PMIX_ERROR_GROUP_LOCAL "pmix.errgroup.local" (bool)
  
  Set true to get local errors notification

- PMIX_ERROR_GROUP_GENERAL "pmix.errgroup.gen" (bool)
  
  Set true to get notified of generic errors

- PMIX_ERROR_HANDLER_ID "pmix.errhandler.id" (int)
  
  Errhandler reference id of notification being reported
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_STD_DIAG 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 6.1).
- Add new attributes to assemble information by its level for storage where ambiguity may exist (see 16.2.3.1).
- 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.

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`

• Update the `PMIx_generate_regex` and `PMIx_generate_ppn` descriptions to clarify that the output from these generator functions may not be a NULL-terminated string, but instead could be a byte array of arbitrary binary content.

• Add a new `PMIX_REGEX` constant that represents a regular expression data type.

### B.7.1 Deprecated constants

The following constants were deprecated in v3.2:

- `PMIX_ERR_DATA_VALUE_NOT_FOUND` Data value not found
- `PMIX_ERR_HANDSHAKE_FAILED` Connection handshake failed
- `PMIX_ERR_IN_ERRNO` Error defined in `errno`
- `PMIX_ERR_INVALID_ARG` Invalid argument
- `PMIX_ERR_INVALID_ARGS` Invalid arguments
- `PMIX_ERR_INVALID_KEY` Invalid key
- `PMIX_ERR_INVALID_KEY_LENGTH` Invalid key length
- `PMIX_ERR_INVALID_KEYVALP` Invalid key/value pair
- `PMIX_ERR_INVALID_LENGTH` Invalid argument length
- `PMIX_ERR_INVALID_NAMESPACE` Invalid namespace
- `PMIX_ERR_INVALID_NUM_ARGS` Invalid number of arguments
- `PMIX_ERR_INVALID_NUM_PARSED` Invalid number parsed
- `PMIX_ERR_INVALID_SIZE` Invalid size
- `PMIX_ERR_INVALID_VAL` Invalid value
- `PMIX_ERR_INVALID_VAL_LENGTH` Invalid value length
- `PMIX_ERR_NOT_IMPLEMENTED` Not implemented
- `PMIX_ERR_PACK_MISMATCH` Pack mismatch
- `PMIX_ERR_PROC_ENTRY_NOT_FOUND` Process not found
- `PMIX_ERR_PROC_REQUESTED_ABORT` Process is already requested to abort
- `PMIX_ERR_READY_FOR_HANDSHAKE` Ready for handshake
- `PMIX_ERR_SERVER_FAILED_REQUEST` Failed to connect to the server
- `PMIX_ERR_SERVER_NOT_AVAIL` Server is not available
- `PMIX_ERR_SILENT` Silent error
- `PMIX_GDS_ACTION_COMPLETE` The Global Data Storage (GDS) action has completed
- `PMIX_NOTIFY_ALLOC_COMPLETE` Notify that a requested allocation operation is complete

- the result of the request will be included in the `info` array
B.7.2 Deprecated attributes

The following attributes were deprecated in v3.2:

**PMIX_ARCH** "pmix.arch" (uint32_t)
Architecture flag.

**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_DSTPATH** "pmix.dstpath" (char*)
Path to shared memory data storage (dstore) files. Deprecated from Standard as being implementation specific.

**PMIX_HWLOC_HOLE_KIND** "pmix.hwlocholek" (char*)
Kind of VM “hole” HWLOC should use for shared memory

**PMIX_HWLOC_SHARE_TOPO** "pmix.hwlocsh" (bool)
Share the HWLOC topology via shared memory

**PMIX_HWLOC_SHMEM_ADDR** "pmix.hwlocaddr" (size_t)
Address of the HWLOC shared memory segment.

**PMIX_HWLOC_SHMEM_FILE** "pmix.hwlocfile" (char*)
Path to the HWLOC shared memory file.

**PMIX_HWLOC_SHMEM_SIZE** "pmix.hwlocsize" (size_t)
Size of the HWLOC shared memory segment.

**PMIX_HWLOC_XML_V1** "pmix.hwlocxml1" (char*)
XML representation of local topology using HWLOC’s v1.x format.

**PMIX_HWLOC_XML_V2** "pmix.hwlocxml2" (char*)
XML representation of local topology using HWLOC’s v2.x format.

**PMIX_LOCAL_TOPO** "pmix.ltopo" (char*)
XML representation of local node topology.

**PMIX_MAPPER** "pmix.mapper" (char*)
Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

**PMIX_MAP_BLOB** "pmix.mblob" (pmix_byte_object_t)
Packed blob of process location.

**PMIX_NON_PMI** "pmix.nonpmi" (bool)
Spawned processes will not call **PMIx_Init**.

**PMIX_PROC_BLOB** "pmix.pblob" (pmix_byte_object_t)
Packed blob of process data.

**PMIX_PROC_URI** "pmix.puri" (char*)
URI containing contact information for the specified process.

**PMIX_TOPOLOGY_FILE** "pmix.topo.file" (char*)
Full path to file containing XML topology description
PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
  Topology signature string.
PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*)
  XML-based description of topology

B.8 Version 4.0: Dec. 2020

NOTE: The PMIx Standard document has undergone significant reorganization in an effort to
become more user-friendly. Highlights include:

- Moving all added, deprecated, and removed items to this revision log section to make them more
  visible
- Co-locating constants and attribute definitions with the primary API that uses them - citations
  and hyperlinks are retained elsewhere
- Splitting the Key-Value Management chapter into separate chapters on the use of reserved keys,
  non-reserved keys, and non-process-related key-value data exchange
- Creating a new chapter on synchronization and data access methods
- Removing references to specific implementations of PMIx and to implementation-specific
  features and/or behaviors

In addition to the reorganization, the following changes were introduced in v4.0 of the PMIx
Standard:

- Clarified that the PMIX_Fence_nb operation can immediately return
  PMIX_OPERATION_SUCCEEDED in lieu of passing the request to a PMIx server if only the
  calling process is involved in the operation
- Added the PMIX_Register_attributes API by which a host environment can register the
  attributes it supports for each server-to-host operation
- Added the ability to query supported attributes from the PMIx tool, client and server libraries, as
  well as the host environment via the new pmix_regattr_t structure. Both human-readable
  and machine-parsable output is supported. New attributes to support this operation include:

  - PMIX_CLIENT_ATTRIBUTES, PMIX_SERVER_ATTRIBUTES,
    PMIX_TOOL_ATTRIBUTES, and PMIX_HOST_ATTRIBUTES to identify which library
    supports the attribute; and
  - PMIX_MAX_VALUE, PMIX_MIN_VALUE, and PMIX_ENUM_VALUE to provide
    machine-parsable description of accepted values
- Add PMIX_APP_WILDCARD to reference all applications within a given job
- Fix signature of blocking APIs PMIx_Allocation_request, PMIx_Job_control,
  PMIx_Process_monitor, PMIx_Get_credential, and
  PMIx_Validate_credential to allow return of results
- Update description to provide an option for blocking behavior of the
  PMIx_Register_event_handler, PMIx_Deregister_event_handler,
  PMIx_Notify_event, PMIx_IOF_pull, PMIx_IOF_deregister, and
  PMIx_IOF_push APIs. The need for blocking forms of these functions was not initially
anticipated but has emerged over time. For these functions, the return value is sufficient to
provide the caller with information otherwise returned via callback. Thus, use of a **NULL** value
as the callback function parameter was deemed a minimal disruption method for providing the
desired capability

- Added a chapter on fabric support that includes new APIs, datatypes, and attributes
- Added a chapter on process sets and groups that includes new APIs and attributes
- Added APIs and a new datatypes to support generation and parsing of PMIx locality and cpuset
  strings
- Added a new chapter on tools that provides deeper explanation on their operation and collecting
  all tool-relevant definitions into one location. Also introduced two new APIs and removed
  restriction that limited tools to being connected to only one server at a time.
- Extended behavior of **PMIx_server_init** to scalably expose the topology description to the
  local clients. This includes creating any required shared memory backing stores and/or XML
  representations, plus ensuring that all necessary key-value pairs for clients to access the
  description are included in the job-level information provided to each client.
- Added a new API by which the host can manually progress the PMIx library in lieu of the
  library’s own progress thread.

The above changes included introduction of the following APIs and data types:

**Client APIs**

- **PMIx_Group_construct**, **PMIx_Group_construct_nb**
- **PMIx_Group_destruct**, **PMIx_Group_destruct_nb**
- **PMIx_Group_invite**, **PMIx_Group_invite_nb**
- **PMIx_Group_join**, **PMIx_Group_join_nb**
- **PMIx_Group_leave**, **PMIx_Group_leave_nb**
- **PMIx_Get_relative_locality**, **PMIx_Load_topology**
- **PMIx_Parse_cpuset_string**, **PMIx_Get_cpuset**
- **PMIx_Link_state_string**, **PMIx_Job_state_string**
- **PMIx_Device_type_string**
- **PMIx_Fabric_register**, **PMIx_Fabric_register_nb**
- **PMIx_Fabric_update**, **PMIx_Fabric_update_nb**
- **PMIx_Fabric_deregister**, **PMIx_Fabric_deregister_nb**
- **PMIx_Compute_distances**, **PMIx_Compute_distances_nb**
- **PMIx_Get_attribute_string**, **PMIx_Get_attribute_name**
- **PMIx_Progress**

**Server APIs**

- **PMIx_server_generate_locality_string**
- **PMIx_Register_attributes**
- **PMIx_server_define_process_set**, **PMIx_server_delete_process_set**
- **pmix_server_grp_fn_t**, **pmix_server_fabric_fn_t**
- **pmix_server_client_connected2_fn_t**
- `PMIx_server_generate_cpuset_string`
- `PMIx_server_register_resources`, `PMIx_server_deregister_resources`

• Tool APIs
  - `PMIx_tool_disconnect`
  - `PMIx_tool_set_server`
  - `PMIx_tool_attach_to_server`
  - `PMIx_tool_get_servers`

• Data types
  - `pmix_regattr_t`
  - `pmix_cpuset_t`
  - `pmix_topology_t`
  - `pmix_locality_t`
  - `pmix_bind_envelope_t`
  - `pmix_group_opt_t`
  - `pmix_group_operation_t`
  - `pmix_fabric_t`
  - `pmix_device_distance_t`
  - `pmix_coord_t`
  - `pmix_coord_view_t`
  - `pmix_geometry_t`
  - `pmix_link_state_t`
  - `pmix_job_state_t`
  - `pmix_device_type_t`

• Callback functions
  - `pmix_device_dist_cbfunc_t`

B.8.1 Added Constants

General error constants
- `PMIX_ERR_EXISTS_OUTSIDE_SCOPE`
- `PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED`
- `PMIX_ERR_EMPTY`
Data type constants
PMIX_COORD
PMIX_REGATTR
PMIX_REGEX
PMIX_JOB_STATE
PMIX_LINK_STATE
PMIX_PROC_CPUSET
PMIX_GEOMETRY
PMIX_DEVICE_DIST
PMIX_ENDPOINT
PMIX_TOPO
PMIX_DEVTYPE
PMIX_LOCTYPE
PMIX_DATA_TYPE_MAX
PMIX_COMPRESSED_BYTE_OBJECT

Info directives
PMIX_INFO_REQD_PROCESSED

Server constants
PMIX_ERR_REPEAT_ATTR_REGISTRATION

Job-Mgmt constants
PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES

Publish constants
PMIX_ERR_DUPLICATE_KEY

Tool constants
PMIX_LAUNCHER_READY
PMIX_ERR_IOF_FAILURE
PMIX_ERR_IOF_COMPLETE
PMIX_EVENT_JOB_START
PMIX_LAUNCH_COMPLETE
PMIX_EVENT_JOB_END
PMIX_EVENT_SESSION_START
PMIX_EVENT_SESSION_END
PMIX_ERR_PROC_TERM_WO_SYNC
PMIX_ERR_JOB_CANCELED
PMIX_ERR_JOB_ABORTED
PMIX_ERR_JOB_KILLED_BY_CMD
PMIX_ERR_JOB_ABORTED_BY_SIG
PMIX_ERR_JOB_TERM_WO_SYNC
PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED
PMIX_ERR_JOB_NON_ZERO_TERM
PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT
PMIX_DEBUG_WAITING_FOR_NOTIFY
PMIX_DEBUGGER_RELEASE

Fabric constants
PMIX_FABRIC_UPDATE_PENDING
PMIX_FABRIC_UPDATED
PMIX_FABRIC_UPDATE_ENDPOINTS
PMIX_COORD_VIEW_UNDEF
PMIX_COORD_LOGICAL_VIEW
PMIX_COORD_PHYSICAL_VIEW
PMIX_LINK_STATE_UNKNOWN
PMIX_LINK_DOWN
PMIX_LINK_UP
PMIX_FABRIC_REQUEST_INFO
PMIX_FABRIC_UPDATE_INFO

Sets-Groups constants
PMIX_PROCESS_SET_DEFINE
PMIX_PROCESS_SET_DELETE
PMIX_GROUP_INVITED
PMIX_GROUP_LEFT
PMIX_GROUP_MEMBER_FAILED
PMIX_GROUP_INVITE_ACCEPTED
PMIX_GROUP_INVITE_DECLINED
PMIX_GROUP_INVITE_FAILED
PMIX_GROUP_MEMBERSHIP_UPDATE
PMIX_GROUP_CONSTRUCT_ABORT
PMIX_GROUP_CONSTRUCT_COMPLETE
PMIX_GROUP_LEADER_FAILED
PMIX_GROUP_LEADER_SELECTED
PMIX_GROUP_CONTEXT_ID_ASSIGNED

Process-Mgmt constants
PMIX_ERR_JOB_ALLOC_FAILED
PMIX_ERR_JOB_APP_NOT_EXECUTABLE
PMIX_ERR_JOB_NO_EXE_SPECIFIED
PMIX_ERR_JOB_FAILED_TO_MAP
PMIX_ERR_JOB_FAILED_TO_LAUNCH
PMIX_LOCALITY_UNKNOWN
PMIX_LOCALITY_NONLOCAL
PMIX_LOCALITY_SHARE_HWTHREAD
PMIX_LOCALITY_SHARE_CORE
PMIX_LOCALITY_SHARE_L1CACHE
PMIX_LOCALITY_SHARE_L2CACHE
PMIX_LOCALITY_SHARE_L3CACHE
PMIX_LOCALITY/share/package
PMIX_LOCALITY_SHARE_NUMA
PMIX_LOCALITY_SHARE_NODE

Events
PMIX_EVENT_SYS_BASE
PMIX_EVENT_NODE_DOWN
PMIX_EVENT_NODE_OFFLINE
PMIX_EVENT_SYS_OTHER

B.8.2 Added Attributes

Sync-Access attributes
PMIX_COLLECT_GENERATED_JOB_INFO "pmix.collect.gen" (bool)
Collect all job-level information (i.e., reserved keys) that was locally generated by PMIx servers. Some job-level information (e.g., distance between processes and fabric devices) is best determined on a distributed basis as it primarily pertains to local processes. Should remote processes need to access the information, it can either be obtained collectively using the PMix_Fence operation with this directive, or can be retrieved one peer at a time using PMIX_Get without first having performed the job-wide collection.

PMIX_ALL_CLONES_PARTICIPATE "pmix.clone.part" (bool)
All clones of the calling process must participate in the collective operation.

PMIX_GET_POINTER_VALUES "pmix.get.pntrs" (bool)
Request that any pointers in the returned value point directly to values in the key-value store. The user must not release any returned data pointers.

PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)
Request that the data be returned in the provided storage location. The caller is responsible for destructing the pmix_value_t using the PMIX_VALUE_DESTRUCT macro when done.

PMIX_GET_REFRESH_CACHE "pmix.get.refresh" (bool)
When retrieving data for a remote process, refresh the existing local data cache for the process in case new values have been put and committed by the process since the last refresh. Local process information is assumed to be automatically updated upon posting by the process. A **NULL** key will cause all values associated with the process to be refreshed - otherwise, only the indicated key will be updated. A process rank of **PMIX_RANK_WILDCARD** can be used to update job-related information in dynamic environments. The user is responsible for subsequently updating refreshed values they may have cached in their own local memory.

**PMIX_QUERY_RESULTS** "pmix.qry.res" (pmix_data_array_t)
Contains an array of query results for a given **pmix_query_t** passed to the **PMIx_Query_info** APIs. If qualifiers were included in the query, then the first element of the array shall be the **PMIX_QUERY_QUALIFIERS** attribute containing those qualifiers. Each of the remaining elements of the array is a **pmix_info_t** containing the query key and the corresponding value returned by the query. This attribute is solely for reporting purposes and cannot be used in **PMIx_Get** or other query operations.

**PMIX_QUERY_QUALIFIERS** "pmix.qry.quals" (pmix_data_array_t)
Contains an array of qualifiers that were included in the query that produced the provided results. This attribute is solely for reporting purposes and cannot be used in **PMIx_Get** or other query operations.

**PMIX_QUERY_SUPPORTED_KEYS** "pmix.qry.keys" (char*)
Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS.

**PMIX_QUERY_SUPPORTED_QUALIFIERS** "pmix.qry.quals" (char*)
Return comma-delimited list of qualifiers supported by a query on the provided key, instead of actually performing the query on the key. NO QUALIFIERS.

**PMIX_QUERY_NAMESPACE_INFO** "pmix.qry.nsinfo" (pmix_data_array_t*)
Return an array of active namespace information - each element will itself contain an array including the namespace plus the command line of the application executing within it. OPTIONAL QUALIFIERS: **PMIX_NSPACE** of specific namespace whose info is being requested.

**PMIX_QUERY_ATTRIBUTE_SUPPORT** "pmix.qry.attrs" (bool)
Query list of supported attributes for specified APIs. REQUIRED QUALIFIERS: one or more of **PMIX_CLIENT_FUNCTIONS**, **PMIX_SERVER_FUNCTIONS**, **PMIX_TOOL_FUNCTIONS**, and **PMIX_HOST_FUNCTIONS**.

**PMIX_QUERY_AVAIL_SERVERS** "pmix.qry.asrvrs" (pmix_data_array_t*)
Return an array of **pmix_info_t**, each element itself containing a **PMIX_SERVER_INFO_ARRAY** entry holding all available data for a server on this node to which the caller might be able to connect.

**PMIX_SERVER_INFO_ARRAY** "pmix.srv.arr" (pmix_data_array_t)
Array of `pmix_info_t` about a given server, starting with its `PMIX_NSPACE` and including at least one of the rendezvous-required pieces of information.

**PMIX_CLIENT_FUNCTIONS** "pmix.client.fns" (bool)
Request a list of functions supported by the PMIx client library.

**PMIX_CLIENT_ATTRIBUTES** "pmix.client.attrs" (bool)
Request attributes supported by the PMIx client library.

**PMIX_SERVER_FUNCTIONS** "pmix.srvr.fns" (bool)
Request a list of functions supported by the PMIx server library.

**PMIX_SERVER_ATTRIBUTES** "pmix.srvr.attrs" (bool)
Request attributes supported by the PMIx server library.

**PMIX_HOST_FUNCTIONS** "pmix.srvr.fns" (bool)
Request a list of functions supported by the host environment.

**PMIX_HOST_ATTRIBUTES** "pmix.host.attrs" (bool)
Request attributes supported by the host environment.

**PMIX_TOOL_FUNCTIONS** "pmix.tool.fns" (bool)
Request a list of functions supported by the PMIx tool library.

**PMIX_TOOL_ATTRIBUTES** "pmix.setup.env" (bool)
Request attributes supported by the PMIx tool library functions.

**Server attributes**

**PMIX_TOPOLOGY2** "pmix.topo2" (pmix_topology_t)
Provide a pointer to an implementation-specific description of the local node topology.

**PMIX_SERVER_SHARE_TOPOLOGY** "pmix.srvr.share" (bool)
The PMIx server is to share its copy of the local node topology (whether given to it or self-discovered) with any clients.

**PMIX_SERVER_SESSION_SUPPORT** "pmix.srvr.sess" (bool)
The host RM wants to declare itself as being the local session server for PMIx connection requests.

**PMIX_SERVER_START_TIME** "pmix.srvr.strtime" (char*)
Time when the server started - i.e., when the server created it’s rendezvous file (given in ctime string format).

**PMIX_SERVER_SCHEDULER** "pmix.srv.sched" (bool)
Server is supporting system scheduler and desires access to appropriate WLM-supporting features. Indicates that the library is to be initialized for scheduler support.

**PMIX_JOB_INFO_ARRAY** "pmix.job.arr" (pmix_data_array_t)
Provide an array of `pmix_info_t` containing job-realm information. The `PMIX_SESSION_ID` attribute of the session containing the job is required to be included in the array whenever the PMIx server library may host multiple sessions (e.g., when executing with a host RM daemon). As information is registered one job (aka namespace) at a time via the `PMIx_server_register_nspace` API, there is no requirement that the array contain either the `PMIX_NSPACE` or `PMIX_JOBID` attributes when used in that context (though either or both of them may be included). At least one of the job identifiers must be provided in all other contexts where the job being referenced is ambiguous.

**PMIX_APP_INFO_ARRAY** "pmix.app.arr" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing application-realm information. The `PMIX_NSPACE` or `PMIX_JOBID` attributes of the job containing the application, plus its `PMIX_APPNUM` attribute, must to be included in the array when the array is not included as part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the application is ambiguous. The job identification is otherwise optional.

**PMIX_PROC_INFO_ARRAY** "pmix.pdata" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing process-realm information. The `PMIX_RANK` and `PMIX_NSPACE` attributes, or the `PMIX_PROCID` attribute, are required to be included in the array when the array is not included as part of a call to `PMIx_server_register_nspace` - i.e., when the job containing the process is ambiguous. All three may be included if desired. When the array is included in some broader structure that identifies the job, then only the `PMIX_RANK` or the `PMIX_PROCID` attribute must be included (the others are optional).

**PMIX_NODE_INFO_ARRAY** "pmix.node.arr" (pmix_data_array_t)

Provide an array of `pmix_info_t` containing node-realm information. At a minimum, either the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute is required to be included in the array, though both may be included.

**PMIX_MAX_VALUE** "pmix.descr.maxval" (varies)

Used in `pmix_regattr_t` to describe the maximum valid value for the associated attribute.

**PMIX_MIN_VALUE** "pmix.descr.minval" (varies)

Used in `pmix_regattr_t` to describe the minimum valid value for the associated attribute.

**PMIX_ENUM_VALUE** "pmix.descr.enum" (char*)

Used in `pmix_regattr_t` to describe accepted values for the associated attribute. Numerical values shall be presented in a form convertible to the attribute’s declared data type. Named values (i.e., values defined by constant names via a typical C-language enum declaration) must be provided as their numerical equivalent.

**PMIX_HOMOGENEOUS_SYSTEM** "pmix.homo" (bool)

The nodes comprising the session are homogeneous - i.e., they each contain the same number of identical packages, fabric interfaces, GPUs, and other devices.
**PMIX_REQUIRED_KEY** "pmix.req.key" (char*)

Identifies a key that must be included in the requested information. If the specified key is not already available, then the PMIx servers are required to delay response to the dmodex request until either the key becomes available or the request times out.

**Job-Mgmt attributes**

**PMIX_ALLOC_ID** "pmix.alloc.id" (char*)

A string identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to PMIx_Spawn.

**PMIX_ALLOC_QUEUE** "pmix.alloc.queue" (char*)

Name of the WLM queue to which the allocation request is to be directed, or the queue being referenced in a query.

**Publish attributes**

**PMIX_ACCESS_PERMISSIONS** "pmix.aperms" (pmix_data_array_t)

Define access permissions for the published data. The value shall contain an array of pmix_info_t structs containing the specified permissions.

**PMIX_ACCESS_USERIDS** "pmix.auids" (pmix_data_array_t)

Array of effective UIDs that are allowed to access the published data.

**PMIX_ACCESS_GRPIDS** "pmix.agids" (pmix_data_array_t)

Array of effective GIDs that are allowed to access the published data.

**Reserved keys**

**PMIX_NUMALLOCATED_NODES** "pmix.num.anodes" (uint32_t)

Number of nodes in the specified realm regardless of whether or not they currently host processes. Defaults to the job realm.

**PMIX_NUM_NODES** "pmix.num.nodes" (uint32_t)

Number of nodes currently hosting processes in the specified realm. Defaults to the job realm.

**PMIX_CMD_LINE** "pmix.cmd.line" (char*)

Command line used to execute the specified job (e.g., "mpirun -n 2 -map-by foo ./myapp : -n 4 ./myapp2").

**PMIX_APP_ARGV** "pmix.app.argv" (char*)

Consolidated argv passed to the spawn command for the given application (e.g., "./myapp arg1 arg2 arg3").

**PMIX_PACKAGE_RANK** "pmix.pkgrank" (uint16_t)

Rank of the specified process on the package where this process resides - refers to the numerical location (starting from zero) of the process on its package when counting only those processes from the same job that share the package, ordered by their overall rank within that job. Note that processes that are not bound to PUs within a single specific package cannot have a package rank.
PMIX_REINCARNATION "pmix.reinc" (uint32_t)
Number of times this process has been re-instantiated - i.e, a value of zero indicates that the
process has never been restarted. 5

PMIX_HOSTNAME_ALIASES "pmix.alias" (char*)
Comma-delimited list of names by which the target node is known.

PMIX_HOSTNAME_KEEP_FQDN "pmix.fqdn" (bool)
FQDNs are being retained by the PMIx library.

PMIX_CPUSET_BITMAP  "pmix.bitmap" (pmix_cpuset_t*)
Bitmap applied to the process upon launch.

PMIX_EXTERNAL_PROGRESS "pmix.evext" (bool)
The host shall progress the PMIx library via calls to PMIx_Progress

PMIX_NODE_MAP_RAW  "pmix.nmap.raw" (char*)
Comma-delimited list of nodes containing procs within the specified realm. Defaults to the
job realm.

PMIX_PROC_MAP_RAW  "pmix.pmap.raw" (char*)
Semi-colon delimited list of strings, each string containing a comma-delimited list of ranks
on the corresponding node within the specified realm. Defaults to the job realm.

Tool attributes

PMIX_TOOL_CONNECT_OPTIONAL  "pmix.tool.conopt" (bool)
The tool shall connect to a server if available, but otherwise continue to operate
unconnected.

PMIX_TOOL_ATTACHMENT_FILE  "pmix.tool.attach" (char*)
Pathname of file containing connection information to be used for attaching to a specific
server.

PMIX_LAUNCHER_RENDEVOUS_FILE  "pmix.tool.lncrnd" (char*)
Pathname of file where the launcher is to store its connection information so that the
spawning tool can connect to it.

PMIX_PRIMARY_SERVER  "pmix.pri.srvr" (bool)
The server to which the tool is connecting shall be designated the primary server once
connection has been accomplished.

PMIX_NOHUP  "pmix.nohup" (bool)
Any processes started on behalf of the calling tool (or the specified namespace, if such
specification is included in the list of attributes) should continue after the tool disconnects
from its server.

PMIX_LAUNCHER_DAEMON  "pmix.lncrn.dmn" (char*)
Path to executable that is to be used as the backend daemon for the launcher. This replaces the launcher’s own daemon with the specified executable. Note that the user is therefore responsible for ensuring compatibility of the specified executable and the host launcher.

PMIX_FORKEXEC_AGENT "pmix.frkex.agnt" (char*)
Path to executable that the launcher’s backend daemons are to fork/exec in place of the actual application processes. The fork/exec agent shall connect back (as a PMIx tool) to the launcher’s daemon to receive its spawn instructions, and is responsible for starting the actual application process it replaced. See Section 17.4.3 for details.

PMIX_EXEC_AGENT "pmix.exec.agnt" (char*)
Path to executable that the launcher’s backend daemons are to fork/exec in place of the actual application processes. The launcher’s daemon shall pass the full command line of the application on the command line of the exec agent, which shall not connect back to the launcher’s daemon. The exec agent is responsible for exec’ing the specified application process in its own place. See Section 17.4.3 for details.

PMIX_IOF_PUSH_STDIN "pmix.iof.stdin" (bool)
Requests that the PMIx library collect the stdin of the requester and forward it to the processes specified in the PMIX_IOF_push call. All collected data is sent to the same targets until stdin is closed, or a subsequent call to PMIX_IOF_push is made that includes the PMIX_IOF_COMPLETE attribute indicating that forwarding of stdin is to be terminated.

PMIX_IOF_COPY "pmix.iof.cpy" (bool)
Requests that the host environment deliver a copy of the specified output stream(s) to the tool, letting the stream(s) continue to also be delivered to the default location. This allows the tool to tap into the output stream(s) without redirecting it from its current final destination.

PMIX_IOF_REDIRECT "pmix.iof.redir" (bool)
Requests that the host environment intercept the specified output stream(s) and deliver it to the requesting tool instead of its current final destination. This might be used, for example, during a debugging procedure to avoid injection of debugger-related output into the application’s results file. The original output stream(s) destination is restored upon termination of the tool.

PMIX_DEBUG_TARGET "pmix.dbg.tgt" (pmix_proc_t*)
Identifier of process(es) to be debugged - a rank of PMIX_RANK_WILDCARD indicates that all processes in the specified namespace are to be included.

PMIX_DEBUG_DAEMONS_PER_PROC "pmix.dbg.dpproc" (uint16_t)
Number of debugger daemons to be spawned per application process. The launcher is to pass the identifier of the namespace to be debugged by including the PMIX_DEBUG_TARGET attribute in the daemon’s job-level information. The debugger daemons spawned on a given node are responsible for self-determining their specific target process(es) - e.g., by referencing their own PMIX_LOCAL_RANK in the daemon debugger job versus the corresponding PMIX_LOCAL_RANK of the target processes on the node.
**PMIX_DEBUG_DAEMONS_PER_NODE** "pmix.dbg.dpnd" (uint16_t)
Number of debugger daemons to be spawned on each node where the target job is executing. The launcher is to pass the identifier of the namespace to be debugged by including the **PMIX_DEBUG_TARGET** attribute in the daemon’s job-level information. The debugger daemons spawned on a given node are responsible for self-determining their specific target process(es) - e.g., by referencing their own **PMIX_LOCAL_RANK** in the daemon debugger job versus the corresponding **PMIX_LOCAL_RANK** of the target processes on the node.

**PMIX_WAIT_FOR_CONNECTION** "pmix.wait.conn" (bool)
Wait until the specified process has connected to the requesting tool or server, or the operation times out (if the **PMIX_TIMEOUT** directive is included in the request).

**PMIX_LAUNCH_DIRECTIVES** "pmix.lnch.dirs" (pmix_data_array_t*)
Array of **pmix_info_t** containing directives for the launcher - a convenience attribute for retrieving all directives with a single call to **PMIx_Get**.

**Fabric attributes**
**PMIX_SERVER_SCHEDULER** "pmix.srv.sched" (bool)
Server is supporting system scheduler and desires access to appropriate WLM-supporting features. Indicates that the library is to be initialized for scheduler support.

**PMIX_FABRIC_COST_MATRIX** "pmix.fab.cm" (pointer)
Pointer to a two-dimensional square 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 overall 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)

**PMIX_FABRIC_VENDOR** "pmix.fab.vndr" (string)
Name of the vendor (e.g., Amazon, Mellanox, HPE, Intel) for the specified fabric.

**PMIX_FABRIC_IDENTIFIER** "pmix.fab.id" (string)
An identifier for the specified fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1).

**PMIX_FABRIC_INDEX** "pmix.fab.idx" (size_t)
The index of the fabric as returned in **pmix_fabric_t**.

**PMIX_FABRIC_NUM_DEVICES** "pmix.fab.nverts" (size_t)
Total number of fabric devices in the overall system - corresponds to the number of rows or columns in the cost matrix.

**PMIX_FABRIC_COORDINATES** "pmix.fab.coords" (pmix_data_array_t)
Array of \texttt{pmix\_geometry\_t} fabric coordinates for devices on the specified node. The
array will contain the coordinates of all devices on the node, including values for all
supported coordinate views. The information for devices on the local node shall be provided
if the node is not specified in the request.

\textbf{PMIX\_FABRIC\_DIMS} "pmix.fab.dims" (uint32\_t)
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 overall system will be returned as a
\texttt{pmix\_data\_array\_t} containing an array of \texttt{uint32\_t} values. Default is to provide
dimensions in \textit{logical} view.

\textbf{PMIX\_FABRIC\_ENDPT} "pmix.fab.endpt" (pmix\_data\_array\_t)
Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given
process (e.g., in the case where multiple devices are associated with a package to which the
process is bound), the returned values will be provided in a \texttt{pmix\_data\_array\_t} of
\texttt{pmix\_endpoint\_t} elements.

\textbf{PMIX\_FABRIC\_SHAPE} "pmix.fab.shape" (pmix\_data\_array\_t\*)
The size of each dimension in the specified fabric plane/view, returned in a
\texttt{pmix\_data\_array\_t} containing an array of \texttt{uint32\_t} values. The size is defined as
the number of elements present in that dimension - e.g., the number of devices in one
dimension of a physical view of a fabric plane. If no plane is specified, then the shape of
each plane in the overall system will be returned in a \texttt{pmix\_data\_array\_t} array where
each element is itself a two-element array containing the \textbf{PMIX\_FABRIC\_PLANE} followed
by that plane’s fabric shape. Default is to provide the shape in \textit{logical} view.

\textbf{PMIX\_FABRIC\_SHAPE\_STRING} "pmix.fab.shapestr" (string)
Network shape expressed as a string (e.g., "10x12x2"). If no plane is specified, then the
shape of each plane in the overall system will be returned in a \texttt{pmix\_data\_array\_t} array
where each element is itself a two-element array containing the \textbf{PMIX\_FABRIC\_PLANE}
followed by that plane’s fabric shape string. Default is to provide the shape in \textit{logical} view.

\textbf{PMIX\_SWITCH\_PEERS} "pmix.speers" (pmix\_data\_array\_t)
Peer ranks that share the same switch as the process specified in the call to \textbf{PMIx\_Get}.
Returns a \texttt{pmix\_data\_array\_t} array of \texttt{pmix\_info\_t} results, each element
containing the \textbf{PMIX\_SWITCH\_PEERS} key with a three-element \texttt{pmix\_data\_array\_t}
array of \texttt{pmix\_info\_t} containing the \textbf{PMIX\_DEVICE\_ID} of the local fabric device, the
\textbf{PMIX\_FABRIC\_SWITCH} identifying the switch to which it is connected, and a
comma-delimited string of peer ranks sharing the switch to which that device is connected.

\textbf{PMIX\_FABRIC\_PLANE} "pmix.fab.plane" (string)
ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
for information, specifies the plane whose information is to be returned. When used directly
as a key in a request, returns a \texttt{pmix\_data\_array\_t} of string identifiers for all fabric
planes in the overall system.

\textbf{PMIX\_FABRIC\_SWITCH} "pmix.fab.switch" (string)
ID string of a fabric switch. When used as a modifier in a request for information, specifies
the switch whose information is to be returned. When used directly as a key in a request,
returns a PMIX_data_array_t of string identifiers for all fabric switches in the overall
system.

PMIX_FABRIC_DEVICE  "pmix.fabdev" (pmix_data_array_t)
An array of pmix_info_t describing a particular fabric device using one or more of the
attributes defined below. The first element in the array shall be the PMIX_DEVICE_ID of
the device.

PMIX_FABRIC_DEVICE_INDEX  "pmix.fabdev.idx" (uint32_t)
Index of the device within an associated communication cost matrix.

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.

PMIX_FABRIC_DEVICE_VENDOR  "pmix.fabdev.vndr" (string)
Indicates the name of the vendor that distributes the device.

PMIX_FABRIC_DEVICE_BUS_TYPE  "pmix.fabdev.btyp" (string)
The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

PMIX_FABRIC_DEVICE_VENDORID  "pmix.fabdev.vendid" (string)
This is a vendor-provided identifier for the device or product.

PMIX_FABRIC_DEVICE_DRIVER  "pmix.fabdev.driver" (string)
The name of the driver associated with the device.

PMIX_FABRIC_DEVICE_FIRMWARE  "pmix.fabdev.fmwr" (string)
The device’s firmware version.

PMIX_FABRIC_DEVICE_ADDRESS  "pmix.fabdev.addr" (string)
The primary link-level address associated with the device, such as a MAC address. If
multiple addresses are available, only one will be reported.

PMIX_FABRIC_DEVICE_COORDINATES  "pmix.fab.coord" (pmix_geometry_t)
The pmix_geometry_t fabric coordinates for the device, including values for all
supported coordinate views.

PMIX_FABRIC_DEVICE_MTU  "pmix.fabdev.mtu" (size_t)
The maximum transfer unit of link level frames or packets, in bytes.

PMIX_FABRIC_DEVICE_SPEED  "pmix.fabdev.speed" (size_t)
The active link data rate, given in bits per second.

PMIX_FABRIC_DEVICE_STATE  "pmix.fabdev.state" (pmix_link_state_t)
The last available physical port state for the specified device. Possible values are
PMIX_LINK_STATE_UNKNOWN, PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate
if the port state is unknown or not applicable (unknown), inactive (down), or active (up).
PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
  Specifies the type of fabric interface currently active on the device, such as Ethernet or
  InfiniBand.

PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
  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
  zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
  combination of node identifier (PMIX_HOSTNAME or PMIX_NODEID) and
  PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the overall system.

Device attributes
PMIX_DEVICE_DISTANCES "pmix.dev.dist" (pmix_data_array_t)
  Return an array of pmix_device_distance_t containing the minimum and maximum
  distances of the given process location to all devices of the specified type on the local node.

PMIX_DEVICE_TYPE "pmix.dev.type" (pmix_device_type_t)
  Bitmask specifying the type(s) of device(s) whose information is being requested. Only used
  as a directive/qualifier.

PMIX_DEVICE_ID "pmix.dev.id" (string)
  System-wide UUID or node-local OS name of a particular device.

Sets-Groups attributes
PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
  Return the number of process sets defined in the specified range (defaults to
  PMIX_RANGE_SESSION).

PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (pmix_data_array_t*)
  Return a pmix_data_array_t containing an array of strings of the process set names
  defined in the specified range (defaults to PMIX_RANGE_SESSION).

PMIX_QUERY_PSET_MEMBERSHIP "pmix.qry.pmems" (pmix_data_array_t*)
  Return an array of pmix_proc_t containing the members of the specified process set.

PMIX_PSET_NAME "pmix.pset.nm" (char*)
  The name of the newly defined process set.

PMIX_PSET_MEMBERS "pmix.pset.mems" (pmix_data_array_t*)
  An array of pmix_proc_t containing the members of the newly defined process set.

PMIX_PSET_NAMES "pmix.pset.nms" (pmix_data_array_t*)
  Returns an array of char* string names of the process sets in which the given process is a
  member.

PMIX_QUERY_NUM_GROUPS "pmix.qry.pgrpnum" (size_t)
  Return the number of process groups defined in the specified range (defaults to session).

OPTIONAL QUALIFIERS: PMIX_RANGE.
PMIX_QUERY_GROUP_NAMES "pmix.qry.pgrp" (pmix_data_array_t*)
Return a pmix_data_array_t containing an array of string names of the process groups defined in the specified range (defaults to session). OPTIONAL QUALIFIERS: PMIX_RANGE.

PMIX_QUERY_GROUP_MEMBERSHIP
"pmix.qry.pgrpmems" (pmix_data_array_t*)
Return a pmix_data_array_t of pmix_proc_t containing the members of the specified process group. REQUIRED QUALIFIERS: PMIX_GROUP_ID.

PMIX_GROUP_ID "pmix.grp.id" (char*)
User-provided group identifier - as the group identifier may be used in PMIx operations, the user is required to ensure that the provided ID is unique within the scope of the host environment (e.g., by including some user-specific or application-specific prefix or suffix to the string).

PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)
This process is the leader of the group.

PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)
Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false.

PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)
Notify remaining members when another member terminates without first leaving the group.

PMIX_GROUP_FT_COLLECTIVE "pmix.grp.ftcoll" (bool)
Adjust internal tracking on-the-fly for terminated processes during a PMIx group collective operation.

PMIX_GROUP.Assign_CONTEXT_ID "pmix.grp.actxid" (bool)
Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, size_t value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to PMIX_RANGE_SESSION.

PMIX_GROUP_LOCALONLY "pmix.grp.lcl" (bool)
Group operation only involves local processes. PMIx implementations are required to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan.

PMIX_GROUP_CONTEXT_ID "pmix.grp.ctxid" (size_t)
Context identifier assigned to the group by the host RM.
PMIX_GROUP_ENDPT_DATA "pmix.grp.endpt" (pmix_byte_object_t)
   Data collected during group construction to ensure communication between group members
   is supported upon completion of the operation.

PMIX_GROUP_NAMES "pmix.pgrp.nm" (pmix_data_array_t*)
   Returns an array of char* string names of the process groups in which the given process is
   a member.

Process Mgmt attributes

PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
   Direct output into files of form "<directory>/<jobid>/rank.<rank>/
   stdout[err]" - can be assigned to the entire job (by including attribute in the job_info array) or on a per-application basis in the info array for each pmix_app_t.

PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)
   Include process stacktraces in timeout report from a job.

PMIX_TIMEOUT_REPORT_STATE "pmix.tim.state" (bool)
   Report process states in timeout report from a job.

PMIX_NOTIFY_JOB_EVENTS "pmix.note.jev" (bool)
   Requests that the launcher generate the PMIX_EVENT_JOB_START,
   PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events. Each event is to
   include at least the namespace of the corresponding job and a PMIX_EVENT_TIMESTAMP
   indicating the time the event occurred. Note that the requester must register for these
   individual events, or capture and process them by registering a default event handler instead
   of individual handlers and then process the events based on the returned status code. Another
   common method is to register one event handler for all job-related events, with a separate
   handler for non-job events - see PMIx_Register_event_handler for details.

PMIX_NOTIFY_PROC_TERMINATION "pmix.noteproc" (bool)
   Requests that the launcher generate the PMIX_EVENT_PROC_TERMINATED event
   whenever a process either normally or abnormally terminates.

PMIX_NOTIFY_PROC_ABNORMAL_TERMINATION "pmix.noteabproc" (bool)
   Requests that the launcher generate the PMIX_EVENT_PROC_TERMINATED event only
   when a process abnormally terminates.

PMIX_LOG_PROC_TERMINATION "pmix.logproc" (bool)
   Requests that the launcher log the PMIX_EVENT_PROC_TERMINATED event whenever a
   process either normally or abnormally terminates.

PMIX_LOG_PROC_ABNORMAL_TERMINATION "pmix.logabproc" (bool)
   Requests that the launcher log the PMIX_EVENT_PROC_TERMINATED event only when a
   process abnormally terminates.

PMIX_LOG_JOB_EVENTS "pmix.log.jev" (bool)
Requests that the launcher log the PMIX_EVENT_JOB_START, PMIX_LAUNCH_COMPLETE, and PMIX_EVENT_JOB_END events using PMIx_Log, subject to the logging attributes of Section 12.4.3.

**PMIX_LOG_COMPLETION** "pmix.logcomp" (bool)

Requests that the launcher log the PMIX_EVENT_JOB_END event for normal or abnormal termination of the spawned job using PMIx_Log, subject to the logging attributes of Section 12.4.3. The event shall include the returned status code (PMIX_JOB_TERM_STATUS) for the corresponding job; the identity (PMIX_PROCID) and exit status (PMIX_EXIT_CODE) of the first failed process, if applicable; and a PMIX_EVENT_TIMESTAMP indicating the time the termination occurred.

**PMIX_FIRST_ENVAR** "pmix.envvar.first" (pmix_envvar_t*)

Ensure the given value appears first in the specified envar using the separator character, creating the envar if it doesn’t already exist

**Event attributes**

**PMIX_EVENT_TIMESTAMP** "pmix.evtstamp" (time_t)

System time when the associated event occurred.

### B.8.3 Added Environmental Variables

**Tool environmental variables**

- PMIX_LAUNCHER_RNDZ_URI
- PMIX_LAUNCHER_RNDZ_FILE
- PMIX_KEEPALIVE_PIPE

### B.8.4 Added Macros

- PMIX_CHECK_RESERVED_KEY
- PMIX_INFO_WAS_PROCESSED
- PMIX_INFO_PROCESSED
- PMIX_INFO_LIST_START
- PMIX_INFO_LIST_ADD
- PMIX_INFO_LIST_XFER
- PMIX_INFO_LIST_CONVERT
- PMIX_INFO_LIST_RELEASE

### B.8.5 Deprecated APIs

- pmix_evhdlr_reg_cbfunc_t Renamed to pmix_hdlr_reg_cbfunc_t

The pmix_server_client_connected_fn_t server module entry point has been deprecated in favor of pmix_server_client_connected2_fn_t

- PMIx_tool_connect_to_server Replaced by PMIx_tool_attach_to_server to allow return of the process identifier of the server to which the tool has attached.
B.8.6 Deprecated constants

The following constants were deprecated in v4.0:

- `PMIX_ERR_DEBUGGER_RELEASE` Renamed to `PMIX_DEBUGGER_RELEASE`
- `PMIX_ERR_JOB_TERMINATED` Renamed to `PMIX_EVENT_JOB_END`
- `PMIX_EXISTS` Renamed to `PMIX_ERR_EXISTS`
- `PMIX_ERR_PROC_ABORTED` Consolidated with `PMIX_EVENT_PROC_TERMINATED`
- `PMIX_ERR_PROC_ABORTING` Consolidated with `PMIX_EVENT_PROC_TERMINATED`
- `PMIX_ERR_LOST_CONNECTION_TO_SERVER` Consolidated into `PMIX_ERR_LOST_CONNECTION`
- `PMIX_ERR_LOST_PEER_CONNECTION` Consolidated into `PMIX_ERR_LOST_CONNECTION`
- `PMIX_ERR_LOST_CONNECTION_TO_CLIENT` Consolidated into `PMIX_ERR_LOST_CONNECTION`
- `PMIX_ERR_INVALID_TERMINATION` Renamed to `PMIX_ERR_JOB_TERM_WO_SYNC`
- `PMIX_PROC_TERMINATED` Renamed to `PMIX_EVENT_PROC_TERMINATED`
- `PMIX_ERR_NODE_DOWN` Renamed to `PMIX_EVENT_NODE_DOWN`
- `PMIX_ERR_NODE_OFFLINE` Renamed to `PMIX_EVENT_NODE_OFFLINE`
- `PMIX_ERR_SYS_OTHER` Renamed to `PMIX_EVENT_SYS_OTHER`
- `PMIX_CONNECT_REQUESTED` Connection has been requested by a PMIx-based tool - deprecated as not required.
- `PMIX_PROC_HAS_CONNECTED` A tool or client has connected to the PMIx server - deprecated in favor of the new `pmix_server_client_connected2_fn_t` server module API

B.8.7 Removed constants

The following constants were removed from the PMIx Standard in v4.0 as they are internal to a particular PMIx implementation.

- `PMIX_ERR_HANDSHAKE_FAILED` Connection handshake failed
- `PMIX_ERR_READY_FOR_HANDSHAKE` Ready for handshake
- `PMIX_ERR_IN_ERRNO` Error defined in `errno`
- `PMIX_ERR_INVALID_VAL_LENGTH` Invalid value length
- `PMIX_ERR_INVALID_LENGTH` Invalid argument length
- `PMIX_ERR_INVALID_NUM_ARGS` Invalid number of arguments
- `PMIX_ERR_INVALID_ARGS` Invalid arguments
- `PMIX_ERR_INVALID_NUM_PARSED` Invalid number parsed
- `PMIX_ERR_INVALID_KEYVALP` Invalid key/value pair
- `PMIX_ERR_INVALID_SIZE` Invalid size
- `PMIX_ERR_PROC_REQUESTED_ABORT` Process is already requested to abort
- `PMIX_ERR_SERVER_FAILED_REQUEST` Failed to connect to the server
- `PMIX_ERR_PROC_ENTRY_NOT_FOUND` Process not found
PMIX_ERR_INVALID_ARG       Invalid argument
PMIX_ERR_INVALID_KEY       Invalid key
PMIX_ERR_INVALID_KEY_LENGTH Invalid key length
PMIX_ERR_INVALID_VAL       Invalid value
PMIX_ERR_INVALID_NAMESPACE Invalid namespace
PMIX_ERR_SERVER_NOT_AVAIl  Server is not available
PMIX_ERR_SILENT            Silent error
PMIX_ERR_PACK_MISMATCH     Pack mismatch
PMIX_ERR_DATA_VALUE_NOT_FOUND Data value not found
PMIX_ERR_NOT_IMPLEMENTED   Not implemented
PMIX_GDS_ACTION_COMPLETE   The GDS action has completed
PMIX_NOTIFY_ALLOC_COMPLETE Notify that a requested allocation operation is complete
- the result of the request will be included in the info array

B.8.8 Deprecated attributes

The following attributes were deprecated in v4.0:

PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
  Renamed to PMIX_TOPOLOGY2.
PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
  Renamed to PMIX_DEBUG_TARGET
PMIX_RECONNECT_SERVER "pmix.tool.recon" (bool)
  Renamed to the PMIx_tool_connect_to_server API
PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)
  Renamed to PMIX_ALLOC_FABRIC
PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
  Renamed to PMIX_ALLOC_FABRIC_ID
PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
  Renamed to PMIX_ALLOC_FABRIC_QOS
PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)
  Renamed to PMIX_ALLOC_FABRIC_TYPE
PMIX_ALLOC_NETWORK_PLANe "pmix.alloc.netplane" (char*)
  Renamed to PMIX_ALLOC_FABRIC_PLANe
PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)
  Renamed to PMIX_ALLOC_FABRIC_ENDPTS
PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
  Renamed to PMIX_ALLOC_FABRIC_ENDPTS_NODE
PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
  Renamed to PMIX_ALLOC_FABRIC_SEC_KEY
PMIX_PROC_DATA "pmix.pdata" (pmix_data_array_t)
  Renamed to PMIX_PROC_INFO_ARRAY
PMIX_LOCALITY "pmix.loc" (pmix_locality_t)
Relative locality of the specified process to the requester, expressed as a bitmask as per the
description in the pmix_locality_t section. This value is unique to the requesting
process and thus cannot be communicated by the server as part of the job-level information.
Its use has been replaced by the PMIx_Get_relative_locality function.

B.8.9 Removed attributes

The following attributes were removed from the PMIx Standard in v4.0 as they are internal to a
particular PMIx implementation. Users are referred to the PMIx_Load_topology API for
obtaining the local topology description.

PMIX_LOCAL_TOPO "pmix.ltopo" (char*)
XML representation of local node topology.

PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*)
XML-based description of topology

PMIX_TOPOLOGY_FILE "pmix.topo.file" (char*)
Full path to file containing XML topology description

PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
Topology signature string.

PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
Address of the HWLOC shared memory segment.

PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
Size of the HWLOC shared memory segment.

PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
Path to the HWLOC shared memory file.

PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
XML representation of local topology using HWLOC’s v1.x format.

PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
XML representation of local topology using HWLOC’s v2.x format.

PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)
Share the HWLOC topology via shared memory

PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)
Kind of VM “hole” HWLOC should use for shared memory

PMIX_DSTPATH "pmix.dstpath" (char*)
Path to shared memory data storage (dstore) files. Deprecated from Standard as being
implementation specific.

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_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.

**PMIX_MAPPER** "pmix.mapper" (char*)

Mapping mechanism to use for placing spawned processes - when accessed using `PMIx_Get`, use the `PMIX_RANK_WILDCARD` value for the rank to discover the mapping mechanism used for the provided namespace.

**PMIX_NON_PMI** "pmix.nonpmi" (bool)

Spawned processes will not call `PMIx_Init`.

**PMIX_PROC_URI** "pmix.puri" (char*)

URI containing contact information for the specified process.

**PMIX_ARCH** "pmix.arch" (uint32_t)

Architecture flag.
APPENDIX C

Acknowledgements

This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.

C.1 Version 4.0

The following list includes some of the active participants in the PMIx v4 standardization process.

- Ralph H. Castain and Danielle Sikich
- Joshua Hursey and David Solt
- Dirk Schubert
- John DelSignore
- Aurelien Bouteiller
- Michael A Raymond
- Howard Pritchard and Nathan Hjelm
- Brice Goglin
- Kathryn Mohror and Stephen Herbein
- Thomas Naughton and Swaroop Pophale
- William E. Allcock and Paul Rich
- Michael Karo
- Artem Polyakov

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- Allinea (ARM)
• Perforce
• University of Tennessee, Knoxville
• The Exascale Computing Project, an initiative of the US Department of Energy
• National Science Foundation
• HPE Co.
• Los Alamos National Laboratory
• INRIA
• Lawrence Livermore National Laboratory
• Oak Ridge National Laboratory
• Argonne National Laboratory
• Altair
• NVIDIA

### C.2 Version 3.0

The following list includes some of the active participants in the PMIx v3 standardization process.

• Ralph H. Castain, Andrew Friedley, Brandon Yates
• Joshua Hursey and David Solt
• Aurelien Bouteiller and George Bosilca
• Dirk Schubert
• Kevin Harms
• Artem Polyakov

The following institutions supported this effort through time and travel support for the people listed above.

• Intel Corporation
• IBM, Inc.
• University of Tennessee, Knoxville
• The Exascale Computing Project, an initiative of the US Department of Energy
• National Science Foundation
• Argonne National Laboratory
• Allinea (ARM)
• NVIDIA

C.3 Version 2.0

The following list includes some of the active participants in the PMIx v2 standardization process.
• Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm and Terry Wilmarth
• Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
• Aurelien Bouteiller and George Bosilca
• Artem Polyakov, Igor Ivanov and Boris Karasev
• Gilles Gouaillardet
• Michael A Raymond and Jim Stoffel
• Dirk Schubert
• Moe Jette
• Takahiro Kawashima and Shinji Sumimoto
• Howard Pritchard
• David Beer
• Brice Goglin
• Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
• Adam Moody and Martin Schulz
• Ryan Grant and Stephen Olivier
• Michael Karo

The following institutions supported this effort through time and travel support for the people listed above.
• Intel Corporation
• IBM, Inc.
• University of Tennessee, Knoxville
• The Exascale Computing Project, an initiative of the US Department of Energy
• National Science Foundation
• Mellanox, Inc.
C.4 Version 1.0

The following list includes some of the active participants in the PMIx v1 standardization process.

- Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- Joshua Hursey and David Solt
- Aurelien Bouteiller and George Bosilca
- Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- Gilles Gouaillardet
- Gary Brown
- Moe Jette

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- Mellanox, Inc.
- Research Organization for Information Science and Technology
• Adaptive Solutions, Inc.

• SchedMD, Inc.
Bibliography


Index

General terms and other items not induced in the other indices.

application, 6, 95, 107, 305, 307, 311, 526, 529
attribute, 8

client, 7, 59
clients, 7
clone, 7
clones, 7, 68, 71, 180, 182, 183, 185, 208, 211, 537

data realm, 98, 269, 270
data realms, 98
device, 8
devices, 8
Direct Modex, 257, 321

fabric, 7
fabric device, 8
fabric devices, 8
fabric plane, 8, 165, 170, 200, 203, 204, 274, 324
fabric planes, 8
fabrics, 7

host environment, 7

instant on, 8, 112, 256

job, 6, 7, 95, 100–104, 107, 295–299, 301, 305–307, 309, 311, 322, 324, 325, 508, 526, 527, 529, 540–542

key, 8

namespace, 6
node, 7, 95, 107, 165, 170, 200, 203, 204, 305, 324

package, 7, 105, 303, 541
peer, 7, 106, 301
peers, 7
process, 7, 95, 107, 165, 170, 200, 203, 204, 305, 324
processing unit, 7
rank, 7, 311
realm, 98
realms, 98
resource manager, 7
RM, 7
scheduler, 7, 273
thread, 7
threads, 7
workflow, 7
workflows, 7, 393
Index of APIs

PMIx_Abort, 24, 159, 160, 346, 348, 463, 478, 524
   PMIxClient.abort (Python), 477
PMIx_Aloc_directive_string, 57, 499, 525
   PMIxClient.alloc_directive_string (Python), 499
PMIx_Allocation_request, 96, 197, 197, 203, 486, 527, 529, 532
   PMIxClient.allocation_request (Python), 485
PMIx_Allocation_request_nb, 200, 203, 205, 525
   PMIxClient.commit (Python), 479
PMIx_Compute_distances, 191, 193, 496, 533
   PMIxClient.compute_distances (Python), 495
PMIx_Compute_distances_nb, 192, 533
PMIx_Connect, 179, 180, 182, 183, 185, 229–231, 436, 437, 483, 524, 526
   PMIxClient.connect (Python), 482
PMIx_Connect_nb, 181, 181, 524
pmix_connection_cbf, 374, 374
pmix_credential_cbf, 281, 391, 392
PMIx_Data_copy, 156, 525
PMIx_Data_copy_payload, 157, 525
PMIx_Data_pack, 152, 153, 293, 525
PMIx_Data_print, 156, 525
PMIx_Data_range_string, 56, 498, 525
   PMIxClient.data_range_string (Python), 498
PMIx_Data_type_string, 56, 499, 525
   PMIxClient.data_type_string (Python), 498
PMIx_Data_unpack, 154, 525
PMIx_Deregister_event_handler, 144, 491, 525, 532
   PMIxClient.deregister_event_handler (Python), 491
pmix_device_dist_cbf, 193, 193, 534
PMIx_Device_type_string, 58, 501, 533
   PMIxClient.device_type_string (Python), 501
PMIx_Disconnect, 182, 183–185, 231, 436, 437, 483, 524, 526
   PMIxClient.disconnect (Python), 483
PMIx_Disconnect_nb, 184, 185, 231, 524
pmix_dmodex_response_fn, 320, 321
PMIx_Error_string, 55, 496, 524
   PMIxClient.error_string (Python), 496
pmix_event_notification_cbf, 143, 149

PMIx_Fabric_deregister, 277, 278, 493, 533
  PMIxClient.fabric_deregister (Python), 493
PMIx_Fabric_deregister_nb, 277, 533
PMIx_Fabric_register, 266, 273, 275, 492, 533
  PMIxClient.fabric_register (Python), 492
PMIx_Fabric_register_nb, 275, 533
PMIx_Fabric_update, 274, 275, 493, 533
  PMIxClient.fabric_update (Python), 493
PMIx_Fabric_update_nb, 276, 533
  PMIxClient.fence (Python), 479
PMIx_Fence_nb, 53, 69, 348, 350, 464, 524, 532
PMIx_Finalize, 24, 63, 65, 65, 178, 345, 346, 436, 437, 463, 477, 524
  PMIxClient.finalize (Python), 477
PMIx_generate_ppn, 294, 504, 524, 530
  PMIxServer.generate_ppn (Python), 503
PMIx_generate_regex, 292, 294, 306, 503, 524, 530
  PMIxServer.generate_regex (Python), 503
  PMIxClient.get (Python), 480
PMIx_Get_attribute_name, 57, 501, 533
  PMIxClient.get_attribute_name (Python), 500
PMIx_Get_attribute_string, 57, 500, 533
  PMIxClient.get_attribute_string (Python), 500
PMIx_Get_cpuset, 190, 495, 533
  PMIxClient.get_cpuset (Python), 494
PMIx_Get_credential, 280, 282, 392, 487, 527, 532
  PMIxClient.get_credential (Python), 487
PMIx_Get_credential_nb, 281
PMIx_Get_nb, 53, 54, 74, 524
PMIx_Get_relative_locality, 187, 189, 302, 337, 494, 533, 553
  PMIxClient.get_relative_locality (Python), 494
PMIx_Get_version, 10, 60, 477, 524
  PMIxClient.get_version (Python), 477
PMIx_Group_construct, 228, 229, 234, 236, 237, 240, 488, 533
  PMIxClient.group_construct (Python), 488
PMIx_Group_construct_nb, 237, 240, 533
PMIx_Group_destruct, 231, 240, 241, 243, 253, 490, 533
  PMIxClient.group_destruct (Python), 490
PMIx_Group_destruct_nb, 241, 243, 533
PMIx_Group_invite, 230, 243, 245, 246, 248, 489, 533
  PMIxClient.group_invite (Python), 489
PMIx_Group_invite_nb, 246, 533
PMIx_Group_join, 230, 245, 246, 248, 249, 250–252, 490, 533
  PMIxClient.group_join (Python), 489
PMIx_Group_join_nb, 248, 251, 252, 533
PMIx_Group_leave, 231, 252, 253–255, 490, 533
  PMIxClient.group_leave (Python), 490
PMIx_Group_leave_nb, 254, 533
pmix_hdlr_reg_cbfnc_t, 55, 138, 448, 449, 550
pmix_info_cbfnc_t, 52, 54, 85, 192, 201, 209, 211, 216, 218, 238, 247, 251, 334, 375, 383, 385, 386, 388, 389, 402, 405
PMIx_Info_directives_string, 56, 498, 525
  PMIxClient.info_directives_string (Python), 498
PMIx_Init, 7, 59, 60, 63, 82, 87, 105, 344, 417, 420, 430, 431, 438, 476, 525, 531, 554
  PMIxClient.init (Python), 476
PMIx_Initialized, 59, 476, 524
  PMIxClient.initialized (Python), 476
pmix_iоф_cbfnc_t, 399, 448, 461
  iofcbfunc (Python), 461
PMIx_IOF_channel_string, 57, 499, 527
  PMIxClient.iof_channel_string (Python), 499
PMIx_IOF_deregister, 449, 517, 527, 532
  PMIxTool.iof_deregister (Python), 517
PMIx_IOF_pull, 363, 379, 414, 415, 419, 423, 425, 447, 449, 516, 527, 532
  PMIxTool.iof_pull (Python), 516
PMIx_IOF_push, 363, 379, 414, 418, 423, 425, 426, 429, 450, 452, 517, 527, 532, 543
  PMIxTool.iof_push (Python), 517
PMIx_Job_control, 197, 206, 207, 210–212, 387, 433, 486, 527, 532
  PMIxClient.job_ctrl (Python), 486
PMIx_Job_control_nb, 78, 205, 208, 304, 525
PMIx_Job_state_string, 57, 500, 533
  PMIxClient.job_state_string (Python), 500
PMIx_Link_state_string, 58, 501, 533
  PMIxClient.link_state_string (Python), 501
PMIx_Load_topology, 186, 494, 533, 553
  PMIxClient.load_topology (Python), 493
PMIx_Log, 174, 219, 222, 225, 416, 435, 485, 527, 550
  PMIxClient.log (Python), 485
PMIx_Log_nb, 222, 225, 525
PMIx_Lookup, 118, 123, 125–127, 480, 481, 524
  PMIxClient.lookup (Python), 481
pmix_lookup_cbfunc_t, 130, 130, 356
PMIx_Lookup_nb, 125, 130, 524
pmix_modex_cbfunc_t, 52, 348, 351, 351, 352
pmix_notification_fn_t, 138, 143, 143, 462
evhandler (Python), 461
PMIx_Notify_event, 145, 373, 492, 525, 532
PMIxClient.notify_event (Python), 492
pmix_op_cbfunc_t, 53, 53, 120, 133, 145, 146, 149, 181, 184, 222, 242, 254, 275, 276, 278, 295,
381, 397, 400, 451
PMIx_Parse_cpuset_string, 189, 337, 495, 533
PMIxClient.parse_cpuset_string (Python), 495
PMIx_Persistence_string, 56, 497, 525
PMIxClient.persistence_string (Python), 497
PMIx_Proc_state_string, 55, 497, 525
PMIxClient.proc_state_string (Python), 496
PMIx_Process_monitor, 197, 213, 218, 487, 527, 532
PMIxClient.monitor (Python), 486
PMIx_Process_monitor_nb, 216, 218, 525
PMIx_Progress, 62, 65, 290, 292, 442, 502, 533, 542
PMIxClient.progress (Python), 502
PMIx_Publish, 118, 120–123, 355, 481, 524
PMIxClient.publish (Python), 480
PMIx_Publish_nb, 120, 123, 524
PMIxClient.put (Python), 478
PMIx_Query_info, 8, 80, 85, 89, 90, 92, 95, 227, 229, 269, 270, 406, 410, 431, 485, 538
PMIxClient.query (Python), 484
PMIx_Query_info_nb, 78, 85, 96, 178, 306, 327, 525, 526
PMIx_Register_attributes, 326, 509, 532, 533
PMIxServer.register_attributes (Python), 509
PMIx_Register_event_handler, 78, 137, 174, 416, 436, 491, 525, 532, 549
PMIxClient.register_event_handler (Python), 491
pmix_release_cbfunc_t, 52, 52
PMIx_Resolve_nodes, 79, 484, 524
PMIxClient.resolve_nodes (Python), 484
PMIx_Resolve_peers, 79, 106, 301, 484, 524
PMIxClient.resolve_peers (Python), 483
PMIx_Scope_string, 56, 497, 525
PMIxClient.scope_string (Python), 497
pmix_server_abort_fn_t, 346, 464
clientaborted (Python), 463
pmix_server_alloc_fn_t, 382, 471
allocate (Python), 471

`pmix_server_client_connected2_fn_t`, 53, 279, 318, 342, 343, 343–345, 463, 533, 550, 551
`clientconnected2 (Python)`, 462

`pmix_server_client_finalized_fn_t`, 345, 346, 463
`clientfinalized (Python)`, 463

PMIx_server_collect_inventory, 334, 336, 511, 527
   PMIxServer.collect_inventory (Python), 510

`pmix_server_connect_fn_t`, 178, 365, 366, 368, 467
`connect (Python)`, 467

PMIx_server_define_process_set, 227, 339, 512, 533
   PMIxServer.define_process_set (Python), 511

PMIx_server_delete_process_set, 227, 340, 512, 533
   PMIxServer.delete_process_set (Python), 512

PMIx_server_deliver_inventory, 335, 511, 527
   PMIxServer.deliver_inventory (Python), 511

PMIx_server_deregister_client, 319, 507, 524
   PMIxServer.deregister_client (Python), 507

`pmix_server_deregister_events_fn_t`, 370, 469
   deregister_events (Python), 469

PMIx_server_deregister_nspace, 315, 319, 506, 524
   PMIxServer.deregister_nspace (Python), 505

PMIx_server_deregister_resources, 316, 506, 513, 534
   PMIxServer.deregister_resources (Python), 506, 513

`pmix_server_disconnect_fn_t`, 366, 368, 468
   disconnect (Python), 468

`pmix_server_dmodex_req_fn_t`, 107, 116, 117, 351, 351, 465, 527, 529
   dmodex (Python), 465

PMIx_server_dmodex_request, 320, 321, 322, 508, 524
   PMIxServer.dmodex_request (Python), 508

`pmix_server_fabric_fn_t`, 266, 273, 404, 476, 533
   fabric (Python), 475

`pmix_server_fencebn_fn_t`, 348, 350, 351, 464, 529
   fence (Python), 464

PMIx_server_finalize, 291, 503, 524
   PMIxServer.finalize (Python), 502

PMIx_server_generate_cpuset_string, 190, 337, 505, 534
   PMIxServer.generate_cpuset_string (Python), 504

PMIx_server_generate_locality_string, 186, 187, 336, 504, 533
   PMIxServer.generate_locality_string (Python), 504

`pmix_server_get_cred_fn_t`, 391, 395, 473
   get_credential (Python), 472

`pmix_server_grp_fn_t`, 401, 475, 533
   group (Python), 475
PMIx_server_init, 59, 287, 291, 327, 341, 407, 408, 412, 502, 524, 533
PMIxServer.init (Python), 502
PMIx_server_IOF_deliver, 333, 424, 510, 527
PMIxServer.iof_deliver (Python), 510
pmix_server_iof_fn_t, 396, 474
iof_pull (Python), 474
pmix_server_job_control_fn_t, 385, 472
job_control (Python), 471
pmix_server_listener_fn_t, 373
pmix_server_log_fn_t, 380, 471
log (Python), 470
pmix_server_lookup_fn_t, 355, 466
lookup (Python), 466
pmix_server_module_t, 288, 290, 327, 328, 341, 341, 342, 502
pmix_server_monitor_fn_t, 388, 472
monitor (Python), 472
pmix_server_notify_event_fn_t, 144, 148, 372, 373, 469
notify_event (Python), 469
pmix_server_publish_fn_t, 353, 465
publish (Python), 465
pmix_server_query_fn_t, 375, 470
query (Python), 469
PMIx_server_register_client, 279, 317, 318, 344, 346, 507, 524
PMIxServer.register_client (Python), 507
pmix_server_register_events_fn_t, 368, 468
register_events (Python), 468
PMIxServer.register_nspace (Python), 505
PMIx_server_register_resources, 297, 300, 301, 315, 506, 513, 534
PMIxServer.register_resources (Python), 506, 512
PMIx_server_setup_application, 322, 325, 326, 332, 333, 336, 509, 525, 529
PMIxServer.setup_application (Python), 508
PMIx_server_setup_fork, 319, 508, 524
PMIxServer.setup_fork (Python), 507
PMIx_server_setup_local_support, 331, 510, 525
PMIxServer.setup_local_support (Python), 509
pmix_server_spawn_fn_t, 177, 360, 417, 467
spawn (Python), 467
pmix_server_stdout_fn_t, 400, 474
iof_push (Python), 474
pmix_server_tool_connection_fn_t, 279, 377, 407, 470
tool_connected (Python), 470
INDEX OF APIS 569

pmix_server_unpublish_fn_t, 358, 466
  unpublish (Python), 466
pmix_server_validate_credential_fn_t, 393, 473
  validate_credential (Python), 473
pmix_setup_application_cbfunc_t, 323, 325
  PMIxClient.spawn (Python), 482
pmix_spawn_cbfunc_t, 166, 177, 177, 360
PMIx_Store_internal, 113, 114, 114, 478, 524
  PMIxClient.store_internal (Python), 478
PMIx_tool_attach_to_server, 409, 412, 421, 442, 444, 515, 534, 550
  PMIxTool.attach_to_server (Python), 515
PMIx_tool_connect_to_server, 527, 552
pmix_tool_connection_cbfunc_t, 378, 379, 380
PMIx_tool_disconnect, 443, 514, 534
  PMIxTool.disconnect (Python), 514
PMIx_tool_finalize, 442, 514, 525
  PMIxTool.finalize (Python), 514
PMIx_tool_get_servers, 446, 515, 534
  PMIxTool.get_servers (Python), 515
PMIx_tool_init, 59, 406, 409, 411–413, 421, 422, 424, 439, 442, 514, 525
  PMIxTool.init (Python), 513
PMIx_tool_set_server, 408, 421, 422, 445, 446, 516, 534
  PMIxTool.set_server (Python), 515
PMIx_Unpublish, 131, 133, 134, 482, 524
  PMIxClient.unpublish (Python), 481
PMIx_Unpublish_nb, 133, 524
PMIx_validate_credential, 283, 488, 527, 532
  PMIxClient.validate_credential (Python), 487
PMIx_validate_credential_nb, 284
pmix_validation_cbfunc_t, 285, 394, 395
pmix_value_cbfunc_t, 53, 53

pmix_evhdlr_reg_cbfunc_t
  (Deprecated), 550
pmix_server_client_connected_fn_t
  (Deprecated), 342, 550
PMIx_tool_connect_to_server
  (Deprecated), 550
Index of Support Macros

PMIX_APP_CONSTRUCT, 176
PMIX_APP_CREATE, 176
PMIX_APP_DESTRUCT, 176
PMIX_APP_FREE, 177
PMIX_APP_INFO_CREATE, 177, 527, 529
PMIX_APP_RELEASE, 176
PMIX_ARGV_APPEND, 46
PMIX_ARGV_APPEND_UNIQUE, 47
PMIX_ARGV_COPY, 49
PMIX_ARGV_COUNT, 49
PMIX_ARGV_FREE, 48
PMIX_ARGV_JOIN, 49
PMIX_ARGV_PREPEND, 47
PMIX_ARGV_SPLIT, 48
PMIX_BYTE_OBJECT_CONSTRUCT, 43
PMIX_BYTE_OBJECT_CREATE, 43
PMIX_BYTE_OBJECT_DESTRUCT, 43
PMIX_BYTE_OBJECT_FREE, 44
PMIX_BYTE_OBJECT_LOAD, 44
PMIX_CHECK_KEY, 17
PMIX_CHECK_NSPACE, 18
PMIX_CHECK_PROCID, 22
PMIX_CHECK_RANK, 20
PMIX_CHECK_RESERVED_KEY, 17, 550
PMIX_COORD_CONSTRUCT, 262
PMIX_COORD_CREATE, 262
PMIX_COORD_DESTRUCT, 262
PMIX_COORD_FREE, 262
PMIX_CPUSET_CONSTRUCT, 338
PMIX_CPUSET_CREATE, 338
PMIX_CPUSET_DESTRUCT, 338
PMIX_CPUSET_FREE, 339
PMIX_DATA_ARRAY_CONSTRUCT, 45
PMIX_DATA_ARRAY_CREATE, 45
PMIX_DATA_ARRAY_DESTRUCT, 45
PMIX_DATA_ARRAY_FREE, 46
PMIX_DATA_BUFFER_CONSTRUCT, 151, 153, 155
PMIX_DATA_BUFFER_CREATE, 151, 153, 155
PMIX_DATA_BUFFER_DESTRUCT, 151
PMIX_DATA_BUFFER_LOAD, 152
PMIX_DATA_BUFFER_RELEASE, 151
PMIX_DATA_BUFFER_UNLOAD, 152, 293
PMIX_DEVICE_DIST_CONSTRUCT, 195
PMIX_DEVICE_DIST_CREATE, 195
PMIX_DEVICE_DIST_DESTRUCT, 195
PMIX_DEVICE_DIST_FREE, 195
PMIX_ENDPOINT_CONSTRUCT, 260
PMIX_ENDPOINT_CREATE, 260
PMIX_ENDPOINT_DESTRUCT, 260
PMIX_ENDPOINT_FREE, 261
PMIX_ENVAR_CONSTRUCT, 41
PMIX_ENVAR_CREATE, 42
PMIX_ENVAR_DESTRUCT, 13, 41
PMIX_ENVAR_FREE, 42
PMIX_ENVAR_LOAD, 42
PMIX_FABRIC_CONSTRUCT, 269
PMIX_GEOMETRY_CONSTRUCT, 263
PMIX_GEOMETRY_CREATE, 264
PMIX_GEOMETRY_DESTRUCT, 264
PMIX_GEOMETRY_FREE, 264
PMIx_Heartbeat, 218, 525
PMIX_INFO_CONSTRUCT, 32
PMIX_INFO_CREATE, 33, 38, 40
PMIX_INFO_DESTRUCT, 33
PMIX_INFO_FREE, 33
PMIX_INFO_IS_END, 40, 527, 529
PMIX_INFO_IS_OPTIONAL, 39
PMIX_INFO_IS_REQUIRED, 38, 39
PMIX_INFO_LIST_ADD, 35, 550
PMIX_INFO_LIST_CONVERT, 37, 550
PMIX_INFO_LIST_RELEASE, 37, 550
PMIX_INFO_LIST_START, 35, 36, 37, 550
PMIX_INFO_LIST_XFER, 36, 550
PMIX_INFO_LOAD, 33
PMIX_INFO_OPTIONAL, 39
PMIX_INFO_PROCESSED, 40, 550
PMIX_INFO_REQUIRED, 37, 39
PMIX_INFO_TRUE, 35
PMIX_INFO_WAS_PROCESSED, 40, 550
PMIX_INFO_XFER, 34, 306
PMIX_LOAD_KEY, 17
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMIX_LOAD_NSPACE</td>
<td>19</td>
</tr>
<tr>
<td>PMIX_LOAD_PROCID</td>
<td>22, 22</td>
</tr>
<tr>
<td>PMIX_MULTICLUSTER_NSPACE_CONSTRUCT</td>
<td>23</td>
</tr>
<tr>
<td>PMIX_MULTICLUSTER_NSPACEPARSE</td>
<td>23</td>
</tr>
<tr>
<td>PMIX_PDATA_CONSTRUCT</td>
<td>127</td>
</tr>
<tr>
<td>PMIX_PDATA_CREATE</td>
<td>128</td>
</tr>
<tr>
<td>PMIX_PDATA_DESTRUCT</td>
<td>127</td>
</tr>
<tr>
<td>PMIX_PDATA_FREE</td>
<td>128</td>
</tr>
<tr>
<td>PMIX_PDATA_LOAD</td>
<td>128</td>
</tr>
<tr>
<td>PMIX_PDATA_RELEASE</td>
<td>128</td>
</tr>
<tr>
<td>PMIX_PDATA_XFER</td>
<td>129</td>
</tr>
<tr>
<td>PMIX_PROC_CONSTRUCT</td>
<td>20</td>
</tr>
<tr>
<td>PMIX_PROC_CREATE</td>
<td>21</td>
</tr>
<tr>
<td>PMIX_PROC_DESTRUCT</td>
<td>21</td>
</tr>
<tr>
<td>PMIX_PROC_FREE</td>
<td>21, 79</td>
</tr>
<tr>
<td>PMIX_PROC_INFO_CONSTRUCT</td>
<td>25</td>
</tr>
<tr>
<td>PMIX_PROC_INFO_CREATE</td>
<td>26</td>
</tr>
<tr>
<td>PMIX_PROC_INFO_DESTRUCT</td>
<td>25</td>
</tr>
<tr>
<td>PMIX_PROC_INFO_FREE</td>
<td>26</td>
</tr>
<tr>
<td>PMIX_PROC_INFO_RELEASE</td>
<td>26</td>
</tr>
<tr>
<td>PMIX_PROC_LOAD</td>
<td>22</td>
</tr>
<tr>
<td>PMIX_PROC_RELEASE</td>
<td>21</td>
</tr>
<tr>
<td>PMIX_QUERY_CONSTRUCT</td>
<td>93</td>
</tr>
<tr>
<td>PMIX_QUERY_CREATE</td>
<td>94</td>
</tr>
<tr>
<td>PMIX_QUERY_DESTRUCT</td>
<td>93</td>
</tr>
<tr>
<td>PMIX_QUERY_FREE</td>
<td>94</td>
</tr>
<tr>
<td>PMIX_QUERY_QUALIFIERS_CREATE</td>
<td>94, 527, 529</td>
</tr>
<tr>
<td>PMIX_QUERY_RELEASE</td>
<td>94</td>
</tr>
<tr>
<td>PMIX_REGATTR_CONSTRUCT</td>
<td>329</td>
</tr>
<tr>
<td>PMIX_REGATTR_CREATE</td>
<td>330</td>
</tr>
<tr>
<td>PMIX_REGATTR_DESTRUCT</td>
<td>330</td>
</tr>
<tr>
<td>PMIX_REGATTR_FREE</td>
<td>330</td>
</tr>
<tr>
<td>PMIX_REGATTR_LOAD</td>
<td>331</td>
</tr>
<tr>
<td>PMIX_REGATTR_XFER</td>
<td>331</td>
</tr>
<tr>
<td>PMIX_SETENV</td>
<td>50</td>
</tr>
<tr>
<td>PMIX_SYSTEM_EVENT</td>
<td>141</td>
</tr>
<tr>
<td>PMIX_TOPOLOGY_CONSTRUCT</td>
<td>188</td>
</tr>
<tr>
<td>PMIX_TOPOLOGY_CREATE</td>
<td>188</td>
</tr>
<tr>
<td>PMIX_TOPOLOGY_DESTRUCT</td>
<td>188</td>
</tr>
<tr>
<td>PMIX_TOPOLOGY_FREE</td>
<td>188</td>
</tr>
<tr>
<td>PMIX_VALUE_CONSTRUCT</td>
<td>29</td>
</tr>
<tr>
<td>PMIX_VALUE_CREATE</td>
<td>29</td>
</tr>
<tr>
<td>PMIX_VALUE_DESTRUCT</td>
<td>29, 73, 77, 537</td>
</tr>
</tbody>
</table>
PMIX_VALUE_FREE, 30
PMIX_VALUE_GET_NUMBER, 32
PMIX_VALUE_LOAD, 30
PMIX_VALUE_RELEASE, 29
PMIX_VALUE_UNLOAD, 31
PMIX_VALUE_XFER, 31
Index of Data Structures

pmix_alloc_directive_t, 52, 57, 198, 201, 205, 205, 383, 457, 499
pmix_bind_envelope_t, 190, 190, 190, 534
pmix_byte_object_t, 43, 43, 43, 51, 280, 281, 283, 285, 333, 393, 394, 400, 450, 456
pmix_coord_t, 52, 261, 261–263, 459, 534
pmix_coord_view_t, 265, 459, 534
pmix_data_buffer_t, 150, 150–154, 158
pmix_data_range_t, 51, 56, 122, 122, 146, 372, 456, 498
pmix_data_type_t, 30, 32, 34, 36, 45, 50, 50, 51, 56, 129, 153, 155–157, 331, 455, 498
pmix_device_distance_t, 52, 191, 193, 194, 194–196, 303, 459, 534, 547
pmix_device_type_t, 52, 58, 193, 193, 194, 272, 459, 501, 534
pmix_endpoint_t, 52, 259, 259–261, 272, 458, 545
pmix_envvar_t, 13, 41, 41, 42, 52, 457
pmix_fabric_operation_t, 266, 266, 404
pmix_fabric_t, 259, 266, 266, 269, 270, 273–278, 405, 458, 534, 544
pmix_geometry_t, 52, 258, 263, 263, 264, 271, 459, 534, 545, 546
pmix_group_operation_t, 402, 404, 404, 534
pmix_group_opt_t, 249, 251, 252, 252, 489, 534
pmix_info_directives_t, 37, 37, 38, 51, 56, 457, 498
pmix_iof_channel_t, 52, 57, 333, 397, 399, 428, 428, 447, 457, 499
pmix_job_state_t, 27, 27, 52, 57, 458, 500, 534
pmix_key_t, 8, 16, 16, 72, 113, 331, 455
pmix_link_state_t, 52, 58, 259, 265, 265, 268, 272, 458, 501, 534, 546
pmix_locality_t, 52, 187, 189, 189, 458, 534, 552, 553
pmix_nspace_t, 18, 18, 19, 22, 23, 177, 455, 456
pmix_pdata_t, 123, 124, 127, 127–130, 457
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmix_persistence_t</td>
<td>51, 56, <strong>122</strong>, 122, 456, 497</td>
</tr>
<tr>
<td>pmix_proc_info_t</td>
<td>24, 24–26, 51, 82, 83, 87, 88, 90, 91, 376, 431, 439, 456</td>
</tr>
<tr>
<td>pmix_proc_state_t</td>
<td><strong>23</strong>, 23, 51, 55, 456, 496</td>
</tr>
<tr>
<td>pmix_query_t</td>
<td>52, 82, 87, 90, <strong>92</strong>, 92–95, 375, 377, 458, 527, 529, 538</td>
</tr>
<tr>
<td>pmix_rank_t</td>
<td><strong>19</strong>, 19, 20, 22, 51, 456</td>
</tr>
<tr>
<td>pmix_regattr_t</td>
<td>52, 96, 327, <strong>328</strong>, 328–331, 458, 532, 534, 540</td>
</tr>
<tr>
<td>pmix_scope_t</td>
<td>51, 56, <strong>114</strong>, 114, 456, 497</td>
</tr>
<tr>
<td>pmix_status_t</td>
<td><strong>14</strong>, 14, 31, 32, 36, 37, 46, 47, 50, 51, 54, 55, 138, 141, 143, 146, 149, 193, 322, 326, 369, 371, 372, 380, 393, 396, 455, 469, 496</td>
</tr>
<tr>
<td>pmix_topology_t</td>
<td>52, 186, <strong>187</strong>, 187, 188, 191, 192, 534</td>
</tr>
<tr>
<td>pmix_value_t</td>
<td>8, <strong>27</strong>, 27–32, 51, 54, 73, 74, 77, 113, 457, 537</td>
</tr>
</tbody>
</table>
Index of Constants

PMIX_ALLOC_DIRECTIVE, 52
PMIX_ALLOC_EXTEND, 205
PMIX_ALLOC_EXTERNAL, 205
PMIX_ALLOC_NEW, 205
PMIX_ALLOC_REAQUIRE, 205
PMIX_ALLOC_RELEASE, 205
PMIX_APP, 51
PMIX_APP_WILDCARD, 13
PMIX_Bool, 51
PMIX_BUFFER, 51
PMIX_BYTE, 51
PMIX_BYTE_OBJECT, 51
PMIX_COMMAND, 51
PMIX_COMPRESSED_BYTE_OBJECT, 52
PMIX_COMPRESSED_STRING, 52
PMIX_COORD, 52
PMIX_COORD_LOGICAL_VIEW, 265
PMIX_COORD_PHYSICAL_VIEW, 265
PMIX_COORD_VIEW_UNDEF, 265
PMIX_CPUBIND_PROCESS, 190
PMIX_CPUBIND_THREAD, 190
PMIX_DATA_ARRAY, 51
PMIX_DATA_RANGE, 51
PMIX_DATA_TYPE, 51
PMIX_DATA_TYPE_MAX, 52
PMIX_DEBUG_WAITING_FOR_NOTIFY, 437
PMIX_DEBUGGER_RELEASE, 437
PMIX_DEVICE_DIST, 52
PMIX_DEVTYPE, 52
PMIX_DEVTYPE_BLOCK, 194
PMIX_DEVTYPE_COPROC, 194
PMIX_DEVTYPE_DMA, 194
PMIX_DEVTYPE_GPU, 194
PMIX_DEVTYPE_NETWORK, 194
PMIX_DEVTYPE_OPENFABRICS, 194
PMIX_DEVTYPE_UNKNOWN, 194
PMIX_DOUBLE, 51
PMIX_ENDPOINT, 52
PMIX_ENVAR, 52
PMIX_ERR_BAD_PARAM, 15
PMIX_ERR_COMM_FAILURE, 15
PMIX_ERR_CONFLICTING_CLEANUP_DIRECTIVES, 211
PMIX_ERR_DUPLICATE_KEY, 121
PMIX_ERR_EMPTY, 15
PMIX_ERR_EVENT_REGISTRATION, 140
PMIX_ERR_EXISTS, 14
PMIX_ERR_EXISTS_OUTSIDE_SCOPE, 14
PMIX_ERR_INIT, 15
PMIX_ERR_INVALID_CRED, 14
PMIX_ERR_INVALID_OPERATION, 15
PMIX_ERR_IOF_COMPLETE, 428
PMIX_ERR_IOF_FAILURE, 428
PMIX_ERR_JOB_ABORTED, 436
PMIX_ERR_JOB_ABORTED_BY_SIG, 436
PMIX_ERR_JOB_ABORTED_BY_SYS_EVENT, 437
PMIX_ERR_JOB_ALLOC_FAILED, 171
PMIX_ERR_JOB_APP_NOT_EXECUTABLE, 171
PMIX_ERR_JOB_CANCELED, 436
PMIX_ERR_JOB_FAILED_TO_LAUNCH, 171
PMIX_ERR_JOB_FAILED_TO_MAP, 171
PMIX_ERR_JOB_KILLED_BY_CMD, 436
PMIX_ERR_JOB_NO_EXE_SPECIFIED, 171
PMIX_ERR_JOB_NON_ZERO_TERM, 437
PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED, 437
PMIX_ERR_JOB_TERM_WO_SYNC, 437
PMIX_ERR_LOST_CONNECTION, 15
PMIX_ERR_NO_PERMISSIONS, 14
PMIX_ERR_NOMEM, 15
PMIX_ERR_NOT_FOUND, 15
PMIX_ERR_NOT_SUPPORTED, 15
PMIX_ERR_OUT_OF_RESOURCE, 15
PMIX_ERR_PACK_FAILURE, 14
PMIX_ERR_PARAM_VALUE_NOT_SUPPORTED, 15
PMIX_ERR_PARTIAL_SUCCESS, 15
PMIX_ERR_PROC_CHECKPOINT, 212
PMIX_ERR_PROC_MIGRATE, 212
PMIX_ERR_PROC_RESTART, 212
PMIX_ERR_PROC_TERM_WO_SYNC, 436
PMIX_ERR_REPEAT_ATTR_REGISTRATION, 328
PMIX_ERR_RESOURCE_BUSY, 15
PMIX_ERR_TIMEOUT, 14
PMIX_GROUP_INVITE_ACCEPTED, 232
PMIX_GROUP_INVITE_DECLINED, 232
PMIX_GROUP_INVITE_FAILED, 232
PMIX_GROUP_INVITED, 231
PMIX_GROUP_LEADER_FAILED, 232
PMIX_GROUP_LEADER_SELECTED, 232
PMIX_GROUP_LEFT, 231
PMIX_GROUP_MEMBER_FAILED, 231
PMIX_GROUP_MEMBERSHIP_UPDATE, 232
PMIX_INFO, 51
PMIX_INFO_ARRAY_END, 38
PMIX_INFO_DIR_RESERVED, 38
PMIX_INFO_DIRECTIVES, 51
PMIX_INFO_REQD, 38
PMIX_INFO_REQD_PROCESSED, 38
PMIX_INT, 51
PMIX_INT16, 51
PMIX_INT32, 51
PMIX_INT64, 51
PMIX_INT8, 51
PMIX_INTERNAL, 114
PMIX_IOF_CHANNEL, 52
PMIX_JCTRL_CHECKPOINT, 212
PMIX_JCTRL_CHECKPOINT_COMPLETE, 212
PMIX_JCTRL_PREEMPT_ALERT, 212
PMIX_JOB_STATE, 52
PMIX_JOB_STATE_AWAITING_ALLOC, 27
PMIX_JOB_STATE_CONNECTED, 27
PMIX_JOB_STATE_LAUNCH_UNDERWAY, 27
PMIX_JOB_STATE_RUNNING, 27
PMIX_JOB_STATE_SUSPENDED, 27
PMIX_JOB_STATE_TERMINATED, 27
PMIX_JOB_STATE_TERMINATED_WITH_ERROR, 27
PMIX_JOB_STATE_UNDEF, 27
PMIX_JOB_STATE_UNTERMINATED, 27
PMIX_KVAL, 51
PMIX_LAUNCH_COMPLETE, 436
PMIX_LAUNCHER_READY, 424
PMIX_LINK_DOWN, 265
PMIX_LINK_STATE, 52
PMIX_LINK_STATE_UNKNOWN, 265
PMIX_LINK_UP, 265
PMIX_LOCAL, 114
PMIX_LOCALITY_NONLOCAL, 189
PMIX_LOCALITY_SHARE_CORE, 189
PMIX_LOCALITY_SHARE_HWTHREAD, 189
PMIX_LOCALITY_SHARE_L1CACHE, 189
PMIX_LOCALITY_SHARE_L2CACHE, 189
PMIX_LOCALITY_SHARE_L3CACHE, 189
PMIX_LOCALITY_SHARE_NODE, 189
PMIX_LOCALITY_SHARE_NUMA, 189
PMIX_LOCALITY_SHARE_PACKAGE, 189
PMIX_LOCALITY_UNKNOWN, 189
PMIX_LOCTYPE, 52
PMIX_MAX_KEYLEN, 13
PMIX_MAX_NSLEN, 13
PMIX_MODEL_DECLARED, 63
PMIX_MODEL_RESOURCES, 63
PMIX_MONITOR_FILE_ALERT, 218
PMIX_MONITOR_HEARTBEAT_ALERT, 218
PMIX_OPENMP_PARALLEL_ENTERED, 63
PMIX_OPENMP_PARALLEL_EXITED, 63
PMIX_OPERATION_IN_PROGRESS, 15
PMIX_OPERATION_SUCCEEDED, 15
PMIX_PDATA, 51
PMIX_PERSIST, 51
PMIX_PERSIST_APP, 122
PMIX_PERSIST_FIRST_READ, 122
PMIX_PERSIST_INDEF, 122
PMIX_PERSIST_INVALID, 122
PMIX_PERSIST_PROC, 122
PMIX_PERSIST_SESSION, 122
PMIX_PID, 51
PMIX_POINTER, 51
PMIX_PROC, 51
PMIX_PROC_CPUSSET, 52
PMIX_PROC_INFO, 51
PMIX_PROC_RANK, 51
PMIX_PROC_STATE, 51
PMIX_PROC_STATE_ABORTED, 24
PMIX_PROC_STATE_ABORTED_BY_SIG, 24
PMIX_PROC_STATE_CALLED_ABORT, 24
PMIX_PROC_STATE_CANNOT_RESTART, 24
PMIX_PROC_STATE_COMM_FAILED, 24
PMIX_PROC_STATE_CONNECTED, 24
PMIX_PROC_STATE_ERROR, 24
PMIX_PROC_STATE_FAILED_TO_LAUNCH, 24
PMIX_PROC_STATE_FAILED_TO_START, 24
PMIX_PROC_STATE_HEARTBEAT_FAILED, 24
PMIX_PROC_STATE_KILLED_BY_CMD, 24
PMIX_PROC_STATE_LAUNCH_UNDERWAY, 24
PMIX_PROC_STATE_MIGRATING, 24
PMIX_PROC_STATE_PREPPED, 24
PMIX_PROC_STATE_RESTART, 24
PMIX_PROC_STATE_RUNNING, 24
PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED, 24
PMIX_PROC_STATE_TERM_NON_ZERO, 24
PMIX_PROC_STATE_TERM_WO_SYNC, 24
PMIX_PROC_STATE_TERMINATE, 24
PMIX_PROC_STATE_TERMINATED, 24
PMIX_PROC_STATE_UNDEF, 24
PMIX_PROC_STATE_UNTERMINATED, 24
PMIX_PROCESS_SET_DEFINE, 228
PMIX_PROCESS_SET_DELETE, 228
PMIX_QUERY, 52
PMIX_QUERY_PARTIAL_SUCCESS, 89
PMIX_RANGE_CUSTOM, 122
PMIX_RANGE_GLOBAL, 122
PMIX_RANGE_INVALID, 122
PMIX_RANGE_LOCAL, 122
PMIX_RANGE_NAMESPACE, 122
PMIX_RANGE_PROC_LOCAL, 122
PMIX_RANGE_RM, 122
PMIX_RANGE_SESSION, 122
PMIX_RANGE_UNDEF, 122
PMIX_RANK_INVALID, 19
PMIX_RANK_LOCAL_NODE, 19
PMIX_RANK_LOCAL_PEERS, 19
PMIX_RANK_UNDEF, 19
PMIX_RANK_VALID, 19
PMIX_RANK_WILDCARD, 19
PMIX_REGATTR, 52
PMIX_REGEX, 52
PMIX_REMOTE, 114
PMIX_SCOPE, 51
PMIX_SCOPE_UNDEF, 114
PMIX_SIZE, 51
PMIX_STATUS, 51
PMIX_STRING, 51
PMIX_SUCCESS, 14
PMIX_TIME, 51
PMIX_TIMEVAL, 51
PMIX_TOPO, 52
PMIX_UINT, 51
PMIX_UINT16, 51
PMIX_UINT32, 51
PMIX_UINT64, 51
PMIX_UINT8, 51
PMIX_UNDEF, 51
PMIX_VALUE, 51

PMIX_CONNECT_REQUESTED
    Deprecated, 551
PMIX_ERR_DATA_VALUE_NOT_FOUND
    Deprecated, 530
    Removed, 552
PMIX_ERR_DEBUGGER_RELEASE
    Deprecated, 551
PMIX_ERR_HANDSHAKE_FAILED
    Deprecated, 530
    Removed, 551
PMIX_ERR_IN_ERRNO
    Deprecated, 530
    Removed, 551
PMIX_ERR_INVALID_ARG
    Deprecated, 530
    Removed, 552
PMIX_ERR_INVALID_ARGS
    Deprecated, 530
    Removed, 551
PMIX_ERR_INVALID_KEY
    Deprecated, 530
    Removed, 552
PMIX_ERR_INVALID_KEY_LENGTH
    Deprecated, 530
    Removed, 552
PMIX_ERR_INVALID_KEYVALP
    Deprecated, 530
    Removed, 551
PMIX_ERR_INVALID_NAMESPACE
    Deprecated, 530
    Removed, 551
PMIX_ERR_IN_ERRNO
Deprecated, 530
Removed, 552

PMIX_ERR_INVALID_NUM_ARGS
Deprecated, 530
Removed, 551

PMIX_ERR_INVALID_NUM_PARSED
Deprecated, 530
Removed, 551

PMIX_ERR_INVALID_SIZE
Deprecated, 530
Removed, 551

PMIX_ERR_INVALID_TERMINATION
Deprecated, 551

PMIX_ERR_INVALID_VAL
Deprecated, 530
Removed, 552

PMIX_ERR_INVALID_VAL_LENGTH
Deprecated, 530
Removed, 551

PMIX_ERR_JOB_TERMINATED
Deprecated, 551

PMIX_ERR_LOST_CONNECTION_TO_CLIENT
Deprecated, 551

PMIX_ERR_LOST_CONNECTION_TO_SERVER
Deprecated, 551

PMIX_ERR_LOST_PEER_CONNECTION
Deprecated, 551

PMIX_ERR_NODE_DOWN
Deprecated, 551

PMIX_ERR_NODE_OFFLINE
Deprecated, 551

PMIX_ERR_NOT_IMPLEMENTED
Deprecated, 530
Removed, 552

PMIX_ERR_PACK_MISMATCH
Deprecated, 530
Removed, 552

PMIX_ERR_PROC_ABORTED
Deprecated, 551

PMIX_ERR_PROC_ABORTING
Deprecated, 551

PMIX_ERR_PROC_ENTRY_NOT_FOUND
Deprecated, 530
PMIX_ERR_PROC_REQUESTED_ABORT
  Deprecated, 530
  Removed, 551
PMIX_ERR_READY_FOR_HANDSHAKE
  Deprecated, 530
  Removed, 551
PMIX_ERR_SERVER_FAILED_REQUEST
  Deprecated, 530
  Removed, 551
PMIX_ERR_SERVER_NOT_AVAIL
  Deprecated, 530
  Removed, 552
PMIX_ERR_SILENT
  Deprecated, 530
  Removed, 552
PMIX_ERR_SYS_OTHER
  Deprecated, 551
PMIX_EXISTS
  Deprecated, 551
PMIX_GDS_ACTION_COMPLETE
  Deprecated, 530
  Removed, 552
PMIX_INFO_ARRAY
  Deprecated, 525
PMIX_MODEX
  Deprecated, 525
PMIX_NOTIFY_ALLOC_COMPLETE
  Deprecated, 530
  Removed, 552
PMIX_PROC_HAS_CONNECTED
  Deprecated, 551
PMIX_PROC_TERMINATED
  Deprecated, 551
Index of Environmental Variables

PMIX_KEEPALIVE_PIPE, 412, 421, 422, 550
PMIX_LAUNCHER_RNDZ_FILE, 408, 412, 550
PMIX_LAUNCHER_RNDZ_URI, 412, 421, 422, 550
Index of Attributes

- **PMIX_ACCESS_GRPIDS**: 122, 541
- **PMIX_ACCESS_PERMISSIONS**: 119, 121, 121, 541
- **PMIX_ACCESS_USERIDS**: 122, 541
- **PMIX_ADD_ENVAR**: 164, 169, 175
- **PMIX_ADD_HOST**: 162, 167, 171, 362
- **PMIX_ADD_HOSTFILE**: 162, 167, 172, 362
- **PMIX_ALL_CLONES_PARTICIPATE**: 68, 71, 71, 180, 182, 183, 185, 537
- **PMIX_ALLOC_BANDWIDTH**: 164, 170, 199, 202, 204, 204, 323, 324, 385
- **PMIX_ALLOC_CPU_LIST**: 164, 170, 199, 202, 204, 384
- **PMIX_ALLOC_FABRIC**: 199, 202, 204, 323, 384, 552
- **PMIX_ALLOC_FABRIC_ENDPTS**: 165, 170, 199, 202, 203, 204, 323, 324, 384, 385, 552
- **PMIX_ALLOC_FABRIC_ID**: 199, 202, 204, 323, 384, 552
- **PMIX_ALLOC_FABRIC_PLANES**: 165, 170, 199, 202, 203, 204, 204, 323, 324, 385, 552
- **PMIX_ALLOC_FABRIC_QOS**: 164, 170, 199, 202, 203, 204, 323, 324, 385, 552
- **PMIX_ALLOC_FABRIC_SECURITY_KEY**: 199, 200, 202, 203, 204, 323, 385, 552
- **PMIX_ALLOC_FABRIC_TYPE**: 165, 170, 199, 202, 203, 204, 323, 384, 385, 552
- **PMIX_ALLOC_ID**: 200, 203, 384, 529, 530, 541
- **PMIX_ALLOC_MEM_SIZE**: 164, 170, 199, 202, 204, 384
- **PMIX_ALLOC_NODELIST**: 164, 170, 199, 202, 203, 384
- **PMIX_ALLOC_NUM_CPU_LIST**: 164, 170, 199, 202, 203, 384
- **PMIX_ALLOC_NUM_CPUS**: 164, 170, 198, 202, 203, 384
- **PMIX_ALLOC_NUM_NODES**: 164, 170, 198, 201, 203, 384
- **PMIX_ALLOC_QUEUE**: 83, 88, 90, 164, 169, 203, 376, 541
- **PMIX_ALLOC_REQ_ID**: 198, 201, 203, 529
- **PMIX_ALLOC_TIME**: 164, 169, 198, 202, 204, 384
- **PMIX_ALLOCATED_NODELIST**: 100, 297
- **PMIX_ANL_MAP**: 101, 102, 298
- **PMIX_APP_ARGV**: 103, 299, 541
- **PMIX_APP_INFO**: 73, 76, 81, 86, 98, 103, 107, 299, 300
- **PMIX_APP_INFO_ARRAY**: 296, 299, 305, 305, 311, 540
- **PMIX_APP_MAP_REGEX**: 104, 299
- **PMIX_APP_MAP_TYPE**: 104, 299
- **PMIX_APP_RANK**: 104, 302
- **PMIX_APP_SIZE**: 103, 299, 310
- **PMIX_APPEND_ENVAR**: 164, 169, 175
- **PMIX_APPLDR**: 103, 299, 311
- **PMIX_APPNUM**: 73, 76, 81, 86, 99, 103, 107, 296, 299–301, 305, 311, 540
INDEX OF ATTRIBUTES  587
PMIX_EVENT_AFFECTED_PROC, 140, 142, 147, 436
PMIX_EVENT_AFFECTED_PROCS, 140, 142, 147, 436
PMIX_EVENT_BASE, 62, 290, 442
PMIX_EVENT_CUSTOM_RANGE, 139, 142, 147
PMIX_EVENT_DO_NOT_CACHE, 142, 147
PMIX_EVENT_HDLR_AFTER, 139, 141
PMIX_EVENT_HDLR_APPEND, 139, 141
PMIX_EVENT_HDLR_BEFORE, 139, 141
PMIX_EVENT_HDLR_FIRST, 139, 141
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, 139, 141
PMIX_EVENT_HDLR_LAST, 139, 141
PMIX_EVENT_HDLR_LAST_IN_CATEGORY, 139, 141
PMIX_EVENT_HDLR_NAME, 139, 141
PMIX_EVENT_HDLR_PREPEND, 139, 141
PMIX_EVENT_NON_DEFAULT, 142, 147
PMIX_EVENT_PROXY, 142, 147
PMIX_EVENT_RETURN_OBJECT, 139, 142
PMIX_EVENT_SILENT_TERMINATION, 165, 170, 174
PMIX_EVENT_TERMINATE_JOB, 142, 147
PMIX_EVENT_TERMINATE_NODE, 142, 147
PMIX_EVENT_TERMINATE_PROC, 142, 147
PMIX_EVENT_TERMINATE_SESSION, 142, 147
PMIX_EVENT_TEXT_MESSAGE, 142, 147
PMIX_EVENT_TIMESTAMP, 142, 174, 416, 417, 435, 436, 549, 550
PMIX_EXEC_AGENT, 420, 423, 434, 543
PMIX_EXIT_CODE, 104, 174, 416, 417, 435, 550
PMIX_EXTERNAL_PROGRESS, 61, 290, 292, 442, 542
PMIX_FABRIC_COORDINATES, 271, 544
PMIX_FABRIC_COST_MATRIX, 267, 269, 544
PMIX_FABRIC_DEVICE, 257, 268, 271, 271, 546
PMIX_FABRIC_DEVICE_ADDRESS, 258, 268, 271, 546
PMIX_FABRIC_DEVICE_BUS_TYPE, 258, 268, 271, 546
PMIX_FABRIC_DEVICE_COORDINATES, 258, 271, 546
PMIX_FABRIC_DEVICE_DRIVER, 258, 268, 271, 271, 546
PMIX_FABRIC_DEVICE_FIRMWARE, 258, 268, 271, 546
PMIX_FABRIC_DEVICE_INDEX, 258, 259, 271, 405, 546
PMIX_FABRIC_DEVICE_MTU, 259, 268, 271, 546
PMIX_FABRIC_DEVICE_NAME, 257, 258, 268, 271, 317, 546
PMIX_FABRIC_DEVICE_PCI_DEVID, 258, 269, 272, 272, 547
PMIX_FABRIC_DEVICE_SPEED, 259, 268, 271, 546
PMIX_FABRIC_DEVICE_STATE, 259, 268, 272, 546
PMIX_FABRIC_DEVICE_TYPE, 259, 268, 272, 547
PMIX_FABRIC_DEVICE_VENDOR, 258, 268, 271, 546
PMIX_FABRIC_DEVICE_VENDORID, 258, 271, 546
PMIX_FABRIC_DEVICES, 257, 271
PMIX_FABRIC_DIMS, 267, 270, 545
PMIX_FABRIC_ENDPT, 272, 545
PMIX_FABRIC_GROUPS, 267, 269, 544
PMIX_FABRIC_IDENTIFIER, 267, 270, 274, 405, 544
PMIX_FABRIC_INDEX, 266, 270, 270, 544
PMIX_FABRIC_NUM_DEVICES, 267, 270, 544
PMIX_FABRIC_PLANE, 267, 268, 270, 271, 274, 405, 545
PMIX_FABRIC_SHAPE, 267, 270, 545
PMIX_FABRIC_SHAPE_STRING, 268, 270, 545
PMIX_FABRIC_SWITCH, 270, 272, 545
PMIX_FABRIC_VENDOR, 267, 270, 274, 405, 544
PMIX_FIRST_ENVAR, 164, 169, 175, 550
PMIX_FORKEXEC_AGENT, 420, 422, 423, 434, 543
PMIX_FWD_STDDIAG, 415, 419, 423, 529
PMIX_FWD_STDERR, 363, 379, 415, 419, 423, 425
PMIX_FWD_STDIN, 362, 379, 414, 418, 422, 425
PMIX_FWD_STDOUT, 363, 379, 414, 418, 422, 423, 425
PMIX_GET_POINTER_VALUES, 73, 74, 76, 77, 537
PMIX_GET_REFRESH_CACHE, 73, 76, 77, 116, 537
PMIX_GET_STATIC_VALUES, 72–74, 77, 77, 537
PMIX_GLOBAL_RANK, 104, 302
PMIX_GROUP_ASSIGN_CONTEXT_ID, 233, 235, 239, 245, 248, 403, 404, 548
PMIX_GROUP_CONTEXT_ID, 233, 403, 548
PMIX_GROUP_ENDPT_DATA, 233, 403, 549
PMIX_GROUP_FT_COLLECTIVE, 233, 235, 239, 245, 248, 548
PMIX_GROUP_ID, 232, 232, 403, 548
PMIX_GROUP_LEADER, 233, 235, 236, 239, 246, 250, 548
PMIX_GROUP_LOCAL_ONLY, 233, 235, 239, 403, 548
PMIX_GROUP_MEMBERSHIP, 233, 236, 403
PMIX_GROUP_NAMES, 233, 549
PMIX_GROUP_NOTIFY_TERMINATION, 233, 235, 236, 239, 241, 245, 248, 548
PMIX_GROUP_OPTIONAL, 233, 235, 236, 239, 244, 248, 403, 548
PMIX_HOMOGENEOUS_SYSTEM, 290, 292, 540
PMIX_HOST, 161, 167, 171, 361
PMIX_HOST_ATTRIBUTES, 82, 87, 92, 96, 431, 532, 539
PMIX_HOST_FUNCTIONS, 82, 87, 91, 92, 96, 538, 539
PMIX_HOSTFILE, 161, 167, 171, 362
PMIX_HOSTNAME, 73, 76, 81, 83, 84, 86, 88, 89, 91, 92, 99, 106, 106, 257, 258, 268, 269, 272,
PMIX_LAUNCHER_DAEMON, 420, 423, 542
PMIX_LAUNCHER_RENDEZVOUS_FILE, 408, 413, 542
PMIX_LOCAL_CPUSETS, 106, 301, 313
PMIX_LOCAL_PEERS, 106, 106, 301, 312
PMIX_LOCAL_PROCS, 106, 301
PMIX_LOCAL_RANK, 105, 302, 432–434, 439, 543, 544
PMIX_LOCAL_SIZE, 107, 300
PMIX_LOCALITY_STRING, 187, 189, 302, 337
PMIX_LOCALLLDR, 106, 300
PMIX_LOG_COMPLETION, 174, 416, 435, 550
PMIX_LOG_EMAIL, 221, 224, 226, 382
PMIX_LOG_EMAIL_ADDR, 221, 224, 226, 382
PMIX_LOG_EMAIL_MSG, 221, 224, 226, 382
PMIX_LOG_EMAIL_SENDER_ADDR, 221, 224, 226
PMIX_LOG_EMAIL_SERVER, 221, 224, 226
PMIX_LOG_EMAIL_SRVR_PORT, 221, 224, 226
PMIX_LOG_EMAIL_SUBJECT, 221, 224, 226, 382
PMIX_LOG_GENERATE_TIMESTAMP, 221, 224, 225
PMIX_LOG_GLOBAL_DATASTORE, 221, 224, 226
PMIX_LOG_GLOBAL_SYSLOG, 220, 223, 225
PMIX_LOG_JOB_EVENTS, 174, 416, 435, 549
PMIX_LOG_JOB_RECORD, 221, 224, 226
PMIX_LOG_LOCAL_SYSLOG, 220, 223, 225
PMIX_LOG_MSG, 226, 382
PMIX_LOG_ONCE, 220, 223, 226
PMIX_LOG_PROC_ABNORMAL_TERMINATION, 174, 549
PMIX_LOG_PROC_TERMINATION, 174, 549
PMIX_LOG_SOURCE, 221, 224, 225
PMIX_LOG STDERR, 220, 223, 225, 381
PMIX_LOG_STDOUT, 220, 223, 225, 381
PMIX_LOG_SYSLOG, 220, 223, 225, 381
PMIX_LOG_SYSLOG_PRI, 220, 223, 225
PMIX_LOG_TAG_OUTPUT, 221, 224, 225
PMIX_LOG_TIMESTAMP, 221, 224, 225
PMIX_LOG_TIMESTAMP_OUTPUT, 221, 224, 225
PMIX_LOG_XML_OUTPUT, 221, 224, 225
PMIX_MAX_PROCS, 100, 100–103, 296–299, 301, 328, 529
PMIX_MAX_RESTARTS, 163, 169, 173, 364
PMIX_MAX_VALUE, 329, 329, 532, 540
PMIX_MAXMERGE STDERR_STDOUT, 163, 168, 172, 298, 362
PMIX_MAX_PROCS, 100, 100–103, 296–299, 301, 328, 529
PMIX_MAX_RESTARTS, 163, 169, 173, 364
PMIX_MAX_VALUE, 329, 329, 532, 540
PMIX_MAX_VALUE, 329, 329, 532, 540
PMIX_MODEL_AFFINITY_POLICY, 62, 64
PMIX_MODEL_CPU_TYPE, 62, 64
PMIX_MODEL_LIBRARY_NAME, 62, 64, 300, 324
PMIX_MODEL_LIBRARY_VERSION, 62, 64, 300, 325
PMIX_MODEL_NUM_CPUS, 62, 64
PMIX_MODEL_NUM_THREADS, 62, 64
PMIX_MODEL_PHASE_NAME, 64, 142
PMIX_MODEL_PHASE_TYPE, 64, 142
PMIX_MONITOR_APP_CONTROL, 214, 217, 218, 390
PMIX_MONITOR_CANCEL, 214, 217, 218, 390
PMIX_MONITOR_FILE, 215, 217, 219, 390
PMIX_MONITOR_FILE_ACCESS, 215, 217, 219, 390
PMIX_MONITOR_FILE_CHECK_TIME, 215, 217, 219, 391
PMIX_MONITOR_FILE_DROPS, 215, 217, 219, 391
PMIX_MONITOR_FILE_MODIFY, 215, 217, 219, 390
PMIX_MONITOR_FILE_SIZE, 215, 217, 219, 390
PMIX_MONITOR_HEARTBEAT, 215, 217, 218, 390
PMIX_MONITOR_HEARTBEAT_DROPS, 215, 217, 219, 390
PMIX_MONITOR_HEARTBEAT_TIME, 215, 217, 219, 390
PMIX_MONITOR_ID, 214, 217, 218, 390
PMIX_NO_OVERSUBSCRIBE, 163, 169, 173, 364
PMIX_NO_PROCS_ON_HEAD, 163, 168, 173, 364
PMIX_NODE_INFO, 73, 76, 81, 86, 99, 106, 301
PMIX_NODE_INFO_ARRAY, 296, 300, 305, 306, 310, 312, 316, 317, 540
PMIX_NODE_LIST, 100, 102, 104
PMIX_NODE_MAP, 100, 102, 103, 298, 309–311, 324, 325, 529
PMIX_NODE_MAP_RAW, 101, 542
PMIX_NODE_RANK, 105, 302, 433
PMIX_NODE_SIZE, 106, 300
PMIX_NODEID, 73, 76, 81, 84, 86, 89, 92, 99, 106, 106, 258, 269, 272, 296, 300–302, 305, 317, 540, 547
PMIX_NOHUP, 415, 420, 423, 542
PMIX_NOTIFY_COMPLETION, 174, 416, 435
PMIX_NOTIFY_JOB_EVENTS, 173, 416, 435, 549
PMIX_NOTIFY_PROC_ABNORMAL_TERMINATION, 174, 549
PMIX_NOTIFY_PROC_TERMINATION, 174, 549
PMIX_NPROC_OFFSET, 101, 298
PMIX_NSDIR, 102, 105, 301, 303
PMIX_NUM_ALLOCATED_NODES, 100, 541
PMIX_NUM_NODES, 98, 100, 102, 103, 309, 310, 541
PMIX_NUM_SLOTS, 100, 101, 103
PMIX_OPTIONAL, 72, 75, 77, 116
PMIX_OUTPUT_TO_DIRECTORY, 173, 549
INDEX OF ATTRIBUTES 593

PMIX_OUTPUT_TO_FILE, 163, 168, 173, 363
PMIX_PACKAGE_RANK, 105, 303, 541
PMIX_PARENT_ID, 104, 361, 421
PMIX_PERSISTENCE, 119, 121, 121, 354, 456
PMIX_PERSONALITY, 162, 167, 171, 362
PMIX_PPR, 162, 167, 172, 362
PMIX_PREFIX, 161, 167, 172, 361
PMIX_PRELOAD_BIN, 162, 167, 172, 362
PMIX_PRELOAD_FILES, 162, 167, 172, 362
PMIX_PREPEND_ENVAR, 164, 169, 175
PMIX_PRIMARY_SERVER, 413, 445, 542
PMIX_PROC_INFO, 81, 86, 99
PMIX_PROC_INFO_ARRAY, 296, 301, 305, 311, 540, 552
PMIX_PROC_MAP, 101, 101, 102, 104, 298, 309, 310, 324, 325, 529
PMIX_PROC_MAP_RAW, 101, 542
PMIX_PROC_PID, 84, 89, 105
PMIX_PROC_STATE_STATUS, 84, 89, 437
PMIX_PROC_TERM_STATUS, 436, 437
PMIX_PROCDIR, 105, 303
PMIX_PROGRAMMING_MODEL, 62, 64, 300, 324
PMIX_PSET_MEMBERS, 228, 229, 547
PMIX_PSET_NAME, 228, 229, 547
PMIX_PSET_NAMES, 227, 229, 299, 547
PMIX_QUERY_ALLOC_STATUS, 83, 88, 91, 377
PMIX_QUERY_ATTRIBUTE_SUPPORT, 82, 87, 91, 95, 431, 538
PMIX_QUERY_AUTHORIZATIONS, 84, 89, 91
PMIX_QUERY_AVAIL_SERVERS, 91, 410, 538
PMIX_QUERY_DEBUG_SUPPORT, 83, 88, 91, 376
PMIX_QUERY_GROUP_MEMBERSHIP, 232, 548
PMIX_QUERY_GROUP_NAMES, 232, 548
PMIX_QUERY_JOB_STATUS, 83, 88, 90, 376
PMIX_QUERY_LOCALONLY, 91, 377
PMIX_QUERY_LOCAL_PROC_TABLE, 83, 88, 90, 376, 431, 439
PMIX_QUERY_MEMORY_USAGE, 83, 88, 91, 377
PMIX_QUERY_NAMESPACE_INFO, 90, 538
PMIX_QUERY_NAMESPACES, 83, 88, 90, 376, 431
PMIX_QUERY_NUM_GROUPS, 232, 547
PMIX_QUERY_NUM_PSETS, 91, 229, 547
PMIX_QUERY_PROC_TABLE, 83, 88, 90, 376, 431, 439
PMIX_QUERY_PSET_MEMBERSHIP, 91, 229, 547
PMIX_QUERY_PSET_NAMES, 91, 229, 547
PMIX_QUERY_QUALIFIERS, 84, 90, 90, 538
PMIX_SESSION_INFO, 73, 76, 81, 86, 98, 100, 108, 296, 298, 324
PMIX_SESSION_INFO_ARRAY, 295, 296, 305, 306, 309, 527
PMIX_SET_ENVAR, 163, 169, 175
PMIX_SET_SESSION_CWD, 161, 167, 172, 361
PMIX_SETUP_APP_ALL, 323, 326
PMIX_SETUP_APP_ENVARS, 323, 326
PMIX_SETUP_APP_NONENVARS, 323, 326
PMIX_SETUP_APP_ALL, 323, 326
PMIX_SETUP_APP_ENVARS, 323, 326
PMIX_SETUP_APP_NONENVARS, 323, 326
PMIX_SINGLE_LISTENER, 61, 289, 291
PMIX_SOCKET_MODE, 61, 289, 291, 441
PMIX_SPAWN_TOOL, 165, 170, 173, 422
PMIX_SPAWNED, 105, 302, 361
PMIX_STDIN_TGT, 162, 168, 172, 362
PMIX_SWITCH_PEERS, 272, 545
PMIX_SYSTEM_TMPDIR, 288, 292, 407, 409
PMIX_TAG_OUTPUT, 162, 168, 172, 363
PMIX_TCP_DISABLE_IPV4, 61, 64, 289, 441
PMIX_TCP_DISABLE_IPV6, 61, 64, 289, 441
PMIX_TCP_IF_EXCLUDE, 61, 64, 289, 441
PMIX_TCP_IF_INCLUDE, 61, 64, 289, 441
PMIX_TCP_IPV4_PORT, 61, 64, 289, 441
PMIX_TCP_IPV6_PORT, 61, 64, 289, 441
PMIX_TCP_REPORT_URI, 61, 63, 289, 441
PMIX_TCP_URI, 63, 409, 410, 440, 444
PMIX_TDIR_RMCLEAN, 100, 299
PMIX_THREADING_MODEL, 62, 64
PMIX_TIME_REMAINING, 78, 84, 89, 91, 377
PMIX_TIMEOUT_REPORT_STATE, 173, 549
PMIX_TIMEOUT_STACKTRACES, 173, 549
PMIX_TIMESTAMP_OUTPUT, 162, 168, 172, 363
PMIX_TMPDIR, 100, 102, 301
PMIX_TOOL_ATTACHMENT_FILE, 409, 410, 413, 440, 444, 542
PMIX_TOOL_ATTRIBUTES, 82, 87, 92, 96, 532, 539
PMIX_TOOL_CONNECT_OPTIONAL, 413, 542
PMIX_TOOL_DO_NOT_CONNECT, 409, 413, 440, 442
PMIX_TOOL_FUNCTIONS, 82, 87, 91, 92, 96, 538, 539
PMIX_TOOL_NSPACE, 378, 408, 411, 440, 442
PMIX_TOOL_RANK, 378, 408, 412, 440, 442
PMIX_TOPOLOGY2, 290, 291, 539, 552
PMIX_UNIV_SIZE, 8, 99, 296, 309, 526, 527, 529
PMIX_UNSET_ENVAR, 164, 169, 175
PMIX_USOCK_DISABLE, 61, 289, 291
PMIX_VERSION_INFO, 379, 380
PMIX_WAIT, 76, 78, 124, 126, 357
PMIX_WAIT_FOR_CONNECTION, 413, 447, 544
PMIX_WDIR, 161, 167, 172, 299, 361

PMIX_ALLOC_NETWORK
    Deprecated, 552
PMIX_ALLOC_NETWORK_ENDPTS
    Deprecated, 552
PMIX_ALLOC_NETWORK_ENDPTS_NODE
    Deprecated, 552
PMIX_ALLOC_NETWORK_ID
    Deprecated, 552
PMIX_ALLOC_NETWORK_PLANE
    Deprecated, 552
PMIX_ALLOC_NETWORK_QOS
    Deprecated, 552
PMIX_ALLOC_NETWORK_SEC_KEY
    Deprecated, 552
PMIX_ALLOC_NETWORK_TYPE
    Deprecated, 552
PMIX_ARCH
    Deprecated, 551
    Removed, 554
PMIX_COLLECTIVE_ALGO
    Deprecated, 551
    Removed, 553
PMIX_COLLECTIVE_ALGO_REQD
    Deprecated, 528
    Removed, 553
PMIX_DEBUG_JOB
    Deprecated, 552
PMIX_DSTPATH
    Deprecated, 551
    Removed, 553
PMIX_ERROR_GROUP_ABORT
    Deprecated, 526
    Removed, 528
PMIX_ERROR_GROUP_COMM
    Deprecated, 526
Removed, 528

PMIX_ERROR_GROUP_GENERAL
  Deprecated, 526
  Removed, 528

PMIX_ERROR_GROUP_LOCAL
  Deprecated, 526
  Removed, 528

PMIX_ERROR_GROUP_MIGRATE
  Deprecated, 526
  Removed, 528

PMIX_ERROR_GROUP_NODE
  Deprecated, 526
  Removed, 528

PMIX_ERROR_GROUP_RESOURCE
  Deprecated, 526
  Removed, 528

PMIX_ERROR_GROUP_SPAWN
  Deprecated, 526
  Removed, 528

PMIX_ERROR_HANDLER_ID
  Deprecated, 526
  Removed, 528

PMIX_ERROR_NAME
  Deprecated, 526
  Removed, 528

PMIX_HWLOC_HOLE_KIND
  Deprecated, 531
  Removed, 553

PMIX_HWLOC_SHARE_TOPO
  Deprecated, 531
  Removed, 553

PMIX_HWLOC_SHMEM_ADDR
  Deprecated, 531
  Removed, 553

PMIX_HWLOC_SHMEM_FILE
  Deprecated, 531
  Removed, 553

PMIX_HWLOC_SHMEM_SIZE
  Deprecated, 531
  Removed, 553

PMIX_HWLOC_XML_V1
  Deprecated, 531
  Removed, 553
PMIX_HWLOC_XML_V2
  Deprecated, 531
  Removed, 553
PMIX_LOCAL_TOPO
  Deprecated, 531
  Removed, 553
PMIX_LOCALITY
  Deprecated, 552
PMIX_MAP_BLOB
  Deprecated, 531
  Removed, 554
PMIX_MAPPER
  Deprecated, 531
  Removed, 554
PMIX_NON_PMI
  Deprecated, 531
  Removed, 554
PMIX_PROC_BLOB
  Deprecated, 531
  Removed, 553
PMIX_PROC_DATA
  Deprecated, 552
PMIX_PROC_URI
  Deprecated, 531
  Removed, 554
PMIX_RECONNECT_SERVER
  Deprecated, 552
PMIX_TOPOLOGY
  Deprecated, 552
PMIX_TOPOLOGY_FILE
  Deprecated, 531
  Removed, 553
PMIX_TOPOLOGY_SIGNATURE
  Deprecated, 532
  Removed, 553
PMIX_TOPOLOGY_XML
  Deprecated, 532
  Removed, 553