

# Process Management Interface for Exascale (PMIx) Standard

**Version 4.0 (Draft)**

*Created on July 29, 2020*

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

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

Copyright © 2018-2019 PMIx Standard Review Board.

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

*This page intentionally left blank*

# Contents

---

<b>1. Introduction</b>	<b>1</b>
1.1. Background . . . . .	1
1.2. PMIx Architecture Overview . . . . .	2
1.3. Portability of Functionality . . . . .	3
1.3.1. Optional Nature of Attributes . . . . .	4
1.4. Organization of this document . . . . .	5
1.5. Version 1.0: June 12, 2015 . . . . .	6
1.6. Version 2.0: Sept. 2018 . . . . .	6
1.7. Version 2.1: Dec. 2018 . . . . .	7
1.8. Version 2.2: Jan 2019 . . . . .	8
1.9. Version 3.0: Dec. 2018 . . . . .	8
1.10. Version 3.1: Jan. 2019 . . . . .	9
1.11. Version 3.2: Oct. 2019 . . . . .	10
1.12. Version 4.0: Sept 2020 . . . . .	10
<b>2. PMIx Terms and Conventions</b>	<b>13</b>
2.1. Notational Conventions . . . . .	15
2.2. Semantics . . . . .	16
2.3. Naming Conventions . . . . .	17
2.4. Procedure Conventions . . . . .	17
2.5. Standard vs Reference Implementation . . . . .	18
<b>3. Data Structures and Types</b>	<b>19</b>
3.1. Constants . . . . .	20
3.1.1. PMIx Error Constants . . . . .	21
3.1.1.1. General Error Constants . . . . .	21
3.1.1.2. Job-Related Error Constants . . . . .	23
3.1.1.3. Operational Error Constants . . . . .	23
3.1.1.4. Job-related constants . . . . .	25

3.1.1.5.	System error constants . . . . .	26
3.1.1.6.	Event handler error constants . . . . .	26
3.1.1.7.	User-Defined Error Constants . . . . .	27
3.1.2.	Macros for use with PMIx constants . . . . .	27
3.1.2.1.	Detect system event constant . . . . .	27
3.2.	Data Types . . . . .	27
3.2.1.	Key Structure . . . . .	27
3.2.1.1.	Check key macro . . . . .	28
3.2.1.2.	Load key macro . . . . .	28
3.2.2.	Namespace Structure . . . . .	29
3.2.2.1.	Check namespace macro . . . . .	29
3.2.2.2.	Load namespace macro . . . . .	30
3.2.3.	Rank Structure . . . . .	30
3.2.4.	Process Structure . . . . .	31
3.2.5.	Process structure support macros . . . . .	31
3.2.5.1.	Initialize the proc structure . . . . .	31
3.2.5.2.	Destruct the proc structure . . . . .	31
3.2.5.3.	Create a proc array . . . . .	32
3.2.5.4.	Free a proc structure . . . . .	32
3.2.5.5.	Free a proc array . . . . .	32
3.2.5.6.	Load a proc structure . . . . .	32
3.2.5.7.	Compare identifiers . . . . .	33
3.2.5.8.	Load a procID structure . . . . .	33
3.2.5.9.	Construct a multi-cluster namespace . . . . .	33
3.2.5.10.	Parse a multi-cluster namespace . . . . .	34
3.2.6.	Process State Structure . . . . .	34
3.2.7.	Job State Structure . . . . .	35
3.2.8.	Process Information Structure . . . . .	36
3.2.9.	Process Information Structure support macros . . . . .	36
3.2.9.1.	Initialize the process information structure . . . . .	36
3.2.9.2.	Destruct the process information structure . . . . .	36
3.2.9.3.	Create a process information array . . . . .	37
3.2.9.4.	Free a process information structure . . . . .	37

3.2.9.5. Free a process information array . . . . .	37
3.2.10. Scope of Put Data . . . . .	38
3.2.11. Job State Structure . . . . .	38
3.2.12. Range of Published Data . . . . .	39
3.2.13. Data Persistence Structure . . . . .	39
3.2.14. Data Array Structure . . . . .	40
3.2.15. Data array structure support macros . . . . .	40
3.2.15.1. Initialize the data array structure . . . . .	40
3.2.15.2. Destruct the data array structure . . . . .	40
3.2.15.3. Create and initialize a data array object . . . . .	41
3.2.15.4. Free a data array object . . . . .	41
3.2.16. Value Structure . . . . .	41
3.2.17. Value structure support macros . . . . .	42
3.2.17.1. Initialize the value structure . . . . .	43
3.2.17.2. Destruct the value structure . . . . .	43
3.2.17.3. Create a value array . . . . .	43
3.2.17.4. Free a value structure . . . . .	43
3.2.17.5. Free a value array . . . . .	44
3.2.17.6. Load a value structure . . . . .	44
3.2.17.7. Unload a value structure . . . . .	45
3.2.17.8. Transfer data between value structures . . . . .	45
3.2.17.9. Retrieve a numerical value from a value struct . . . . .	46
3.2.18. Info Structure . . . . .	46
3.2.19. Info structure support macros . . . . .	47
3.2.19.1. Initialize the info structure . . . . .	47
3.2.19.2. Destruct the info structure . . . . .	47
3.2.19.3. Create an info array . . . . .	47
3.2.19.4. Free an info array . . . . .	47
3.2.19.5. Load key and value data into a info struct . . . . .	48
3.2.19.6. Copy data between info structures . . . . .	48
3.2.19.7. Test a boolean info struct . . . . .	49
3.2.20. Info Type Directives . . . . .	49

3.2.21.	Info Directive support macros . . . . .	50
3.2.21.1.	Mark an info structure as required . . . . .	50
3.2.21.2.	Mark an info structure as optional . . . . .	50
3.2.21.3.	Test an info structure for <i>required</i> directive . . . . .	51
3.2.21.4.	Test an info structure for <i>optional</i> directive . . . . .	51
3.2.21.5.	Test an info structure for <i>end of array</i> directive . . . . .	51
3.2.22.	Job Allocation Directives . . . . .	52
3.2.23.	IO Forwarding Channels . . . . .	52
3.2.24.	Environmental Variable Structure . . . . .	52
3.2.25.	Environmental variable support macros . . . . .	53
3.2.25.1.	Initialize the envar structure . . . . .	53
3.2.25.2.	Destruct the envar structure . . . . .	53
3.2.25.3.	Create an envar array . . . . .	53
3.2.25.4.	Free an envar array . . . . .	54
3.2.25.5.	Load an envar structure . . . . .	54
3.2.26.	Lookup Returned Data Structure . . . . .	54
3.2.27.	Lookup data structure support macros . . . . .	54
3.2.27.1.	Initialize the pdata structure . . . . .	55
3.2.27.2.	Destruct the pdata structure . . . . .	55
3.2.27.3.	Create a pdata array . . . . .	55
3.2.27.4.	Free a pdata structure . . . . .	55
3.2.27.5.	Free a pdata array . . . . .	56
3.2.27.6.	Load a lookup data structure . . . . .	56
3.2.27.7.	Transfer a lookup data structure . . . . .	57
3.2.28.	Application Structure . . . . .	57
3.2.29.	App structure support macros . . . . .	58
3.2.29.1.	Initialize the app structure . . . . .	58
3.2.29.2.	Destruct the app structure . . . . .	58
3.2.29.3.	Create an app array . . . . .	58
3.2.29.4.	Free an app structure . . . . .	58
3.2.29.5.	Free an app array . . . . .	59
3.2.29.6.	Create the info array of application directives . . . . .	59
3.2.30.	Query Structure . . . . .	59

3.2.31.	Query structure support macros . . . . .	59
3.2.31.1.	Initialize the query structure . . . . .	60
3.2.31.2.	Destruct the query structure . . . . .	60
3.2.31.3.	Create a query array . . . . .	60
3.2.31.4.	Free a query structure . . . . .	60
3.2.31.5.	Free a query array . . . . .	61
3.2.31.6.	Create the info array of query qualifiers . . . . .	61
3.2.32.	Attribute registration structure . . . . .	61
3.2.33.	Attribute registration structure support macros . . . . .	62
3.2.33.1.	Initialize the regattr structure . . . . .	63
3.2.33.2.	Destruct the regattr structure . . . . .	63
3.2.33.3.	Create a regattr array . . . . .	63
3.2.33.4.	Free a regattr array . . . . .	63
3.2.33.5.	Load a regattr structure . . . . .	64
3.2.33.6.	Transfer a regattr to another regattr . . . . .	64
3.2.34.	PMIx Group Directives . . . . .	64
3.2.35.	Byte Object Type . . . . .	65
3.2.36.	Byte object support macros . . . . .	65
3.2.36.1.	Initialize the byte object structure . . . . .	65
3.2.36.2.	Destruct the byte object structure . . . . .	65
3.2.36.3.	Create a byte object structure . . . . .	65
3.2.36.4.	Free a byte object array . . . . .	66
3.2.36.5.	Load a byte object structure . . . . .	66
3.2.37.	Data Array Structure . . . . .	66
3.2.38.	Data array support macros . . . . .	66
3.2.38.1.	Initialize a data array structure . . . . .	67
3.2.38.2.	Destruct a data array structure . . . . .	67
3.2.38.3.	Create a data array structure . . . . .	67
3.2.38.4.	Free a data array structure . . . . .	68
3.2.39.	Argument Array Macros . . . . .	68
3.2.39.1.	Argument array extension . . . . .	68
3.2.39.2.	Argument array prepend . . . . .	69
3.2.39.3.	Argument array extension - unique . . . . .	69

3.2.39.4. Argument array release . . . . .	70
3.2.39.5. Argument array split . . . . .	70
3.2.39.6. Argument array join . . . . .	71
3.2.39.7. Argument array count . . . . .	71
3.2.39.8. Argument array copy . . . . .	72
3.2.40. Set Environment Variable . . . . .	72
3.3. Generalized Data Types Used for Packing/Unpacking . . . . .	73
3.4. Reserved attributes . . . . .	74
3.4.1. Initialization attributes . . . . .	75
3.4.2. Identification attributes . . . . .	76
3.4.3. Programming model attributes . . . . .	76
3.4.4. UNIX socket rendezvous socket attributes . . . . .	77
3.4.5. TCP connection attributes . . . . .	77
3.4.6. Global Data Storage (GDS) attributes . . . . .	78
3.4.7. General process-level attributes . . . . .	78
3.4.8. Scratch directory attributes . . . . .	78
3.4.9. Relative Rank Descriptive Attributes . . . . .	79
3.4.10. Information retrieval attributes . . . . .	80
3.4.11. Information storage attributes . . . . .	81
3.4.12. Size information attributes . . . . .	82
3.4.13. Memory information attributes . . . . .	83
3.4.14. Topology information attributes . . . . .	83
3.4.15. Request-related attributes . . . . .	84
3.4.16. Server-to-PMIx library attributes . . . . .	86
3.4.17. Server-to-Client attributes . . . . .	86
3.4.18. Event handler registration and notification attributes . . . . .	86
3.4.19. Fault tolerance attributes . . . . .	87
3.4.20. Spawn attributes . . . . .	88
3.4.21. Query attributes . . . . .	91
3.4.22. Log attributes . . . . .	92
3.4.23. Resource manager attributes . . . . .	94
3.4.24. Environment variable attributes . . . . .	94
3.4.25. Job Allocation attributes . . . . .	94

3.4.26.	Job control attributes . . . . .	96
3.4.27.	Monitoring attributes . . . . .	98
3.4.28.	Security attributes . . . . .	98
3.4.29.	Application setup attributes . . . . .	99
3.4.30.	Attribute support level attributes . . . . .	99
3.4.31.	Descriptive attributes . . . . .	99
3.4.32.	Process group attributes . . . . .	100
3.4.33.	Storage-related attributes . . . . .	100
3.5.	Callback Functions . . . . .	101
3.5.1.	Release Callback Function . . . . .	101
3.5.2.	Modex Callback Function . . . . .	102
3.5.3.	Spawn Callback Function . . . . .	103
3.5.4.	Op Callback Function . . . . .	103
3.5.5.	Lookup Callback Function . . . . .	104
3.5.6.	Value Callback Function . . . . .	104
3.5.7.	Info Callback Function . . . . .	105
3.5.8.	Event Handler Registration Callback Function . . . . .	106
3.5.9.	Notification Handler Completion Callback Function . . . . .	106
3.5.10.	Notification Function . . . . .	107
3.5.11.	Server Setup Application Callback Function . . . . .	109
3.5.12.	Server Direct Modex Response Callback Function . . . . .	110
3.5.13.	PMIx Client Connection Callback Function . . . . .	111
3.5.14.	PMIx Tool Connection Callback Function . . . . .	111
3.5.15.	Credential callback function . . . . .	112
3.5.16.	Credential validation callback function . . . . .	113
3.5.17.	IOF delivery function . . . . .	114
3.5.18.	IOF and Event registration function . . . . .	115
3.6.	Constant String Functions . . . . .	116
<b>4.</b>	<b>Initialization and Finalization</b>	<b>120</b>
4.1.	Query . . . . .	120
4.1.1.	<b>PMIx_Initialized</b> . . . . .	120
4.1.2.	<b>PMIx_Get_version</b> . . . . .	121

4.2.	Client Initialization and Finalization . . . . .	121
4.2.1.	<b>PMIx_Init</b> . . . . .	121
4.2.2.	<b>PMIx_Finalize</b> . . . . .	124
<b>5.</b>	<b>Key-Value Management</b>	<b>125</b>
5.1.	Reserved Keys . . . . .	125
5.1.1.	Required Keys . . . . .	125
5.1.2.	Optional Keys . . . . .	126
5.2.	Non-reserved Keys . . . . .	126
5.3.	Setting and Accessing Key/Value Pairs . . . . .	126
5.3.1.	<b>PMIx_Put</b> . . . . .	126
5.3.2.	<b>PMIx_Get</b> . . . . .	127
5.3.3.	<b>PMIx_Get_nb</b> . . . . .	130
5.3.4.	<b>PMIx_Store_internal</b> . . . . .	133
5.3.5.	Accessing information: examples . . . . .	134
5.3.5.1.	Session-level information . . . . .	134
5.3.5.2.	Job-level information . . . . .	135
5.3.5.3.	Application-level information . . . . .	136
5.3.5.4.	Process-level information . . . . .	137
5.3.5.5.	Node-level information . . . . .	137
5.4.	Exchanging Key/Value Pairs . . . . .	138
5.4.1.	<b>PMIx_Commit</b> . . . . .	138
5.4.2.	<b>PMIx_Fence</b> . . . . .	139
5.4.3.	<b>PMIx_Fence_nb</b> . . . . .	140
5.5.	Publish and Lookup Data . . . . .	143
5.5.1.	<b>PMIx_Publish</b> . . . . .	143
5.5.2.	<b>PMIx_Publish_nb</b> . . . . .	145
5.5.3.	<b>PMIx_Lookup</b> . . . . .	146
5.5.4.	<b>PMIx_Lookup_nb</b> . . . . .	149
5.5.5.	<b>PMIx_Unpublish</b> . . . . .	150
5.5.6.	<b>PMIx_Unpublish_nb</b> . . . . .	152

<b>6. Process Management</b>	<b>154</b>
6.1. Abort . . . . .	154
6.1.1. <b>PMIx_Abort</b> . . . . .	154
6.2. Process Creation . . . . .	155
6.2.1. <b>PMIx_Spawn</b> . . . . .	155
6.2.2. <b>PMIx_Spawn_nb</b> . . . . .	161
6.3. Connecting and Disconnecting Processes . . . . .	166
6.3.1. <b>PMIx_Connect</b> . . . . .	167
6.3.2. <b>PMIx_Connect_nb</b> . . . . .	170
6.3.3. <b>PMIx_Disconnect</b> . . . . .	171
6.3.4. <b>PMIx_Disconnect_nb</b> . . . . .	173
6.4. Process Locality . . . . .	175
6.4.1. <b>PMIx_Load_topology</b> . . . . .	175
6.4.2. <b>PMIx_Get_relative_locality</b> . . . . .	176
6.4.2.1. Topology description . . . . .	177
6.4.2.2. Initialize the topology description structure . . . . .	177
6.4.2.3. Relative locality of two processes . . . . .	178
6.4.2.4. <b>Locality keys</b> . . . . .	178
<b>7. Job Management and Reporting</b>	<b>179</b>
7.1. Query . . . . .	179
7.1.1. <b>PMIx_Resolve_peers</b> . . . . .	179
7.1.2. <b>PMIx_Resolve_nodes</b> . . . . .	180
7.1.3. <b>PMIx_Query_info</b> . . . . .	181
7.1.4. <b>PMIx_Query_info_nb</b> . . . . .	186
7.1.4.1. Using Get vs Query . . . . .	190
7.1.4.2. Accessing attribute support information . . . . .	191
7.2. Allocation Requests . . . . .	191
7.2.1. <b>PMIx_Allocation_request</b> . . . . .	192
7.2.2. <b>PMIx_Allocation_request_nb</b> . . . . .	195
7.3. Job Control . . . . .	198
7.3.1. <b>PMIx_Job_control</b> . . . . .	199
7.3.2. <b>PMIx_Job_control_nb</b> . . . . .	202

7.4.	Process and Job Monitoring . . . . .	205
7.4.1.	<b>PMIx_Process_monitor</b> . . . . .	205
7.4.2.	<b>PMIx_Process_monitor_nb</b> . . . . .	207
7.4.3.	<b>PMIx_Heartbeat</b> . . . . .	209
7.5.	Logging . . . . .	209
7.5.1.	<b>PMIx_Log</b> . . . . .	209
7.5.2.	<b>PMIx_Log_nb</b> . . . . .	212
<b>8.</b>	<b>Event Notification</b>	<b>216</b>
8.1.	Notification and Management . . . . .	216
8.1.1.	<b>PMIx_Register_event_handler</b> . . . . .	218
8.1.2.	<b>PMIx_Deregister_event_handler</b> . . . . .	221
8.1.3.	<b>PMIx_Notify_event</b> . . . . .	222
<b>9.</b>	<b>Data Packing and Unpacking</b>	<b>226</b>
9.1.	Data Buffer Type . . . . .	226
9.2.	Support Macros . . . . .	227
9.2.1.	<b>PMIX_DATA_BUFFER_CREATE</b> . . . . .	227
9.2.2.	<b>PMIX_DATA_BUFFER_RELEASE</b> . . . . .	227
9.2.3.	<b>PMIX_DATA_BUFFER_CONSTRUCT</b> . . . . .	227
9.2.4.	<b>PMIX_DATA_BUFFER_DESTRUCT</b> . . . . .	228
9.2.5.	<b>PMIX_DATA_BUFFER_LOAD</b> . . . . .	228
9.2.6.	<b>PMIX_DATA_BUFFER_UNLOAD</b> . . . . .	229
9.3.	General Routines . . . . .	229
9.3.1.	<b>PMIx_Data_pack</b> . . . . .	229
9.3.2.	<b>PMIx_Data_unpack</b> . . . . .	231
9.3.3.	<b>PMIx_Data_copy</b> . . . . .	233
9.3.4.	<b>PMIx_Data_print</b> . . . . .	233
9.3.5.	<b>PMIx_Data_copy_payload</b> . . . . .	234
<b>10.</b>	<b>Security</b>	<b>236</b>
10.1.	Obtaining Credentials . . . . .	237
10.1.1.	<b>PMIx_Get_credential</b> . . . . .	237
10.1.2.	<b>PMIx_Get_credential_nb</b> . . . . .	238

10.2.	Validating Credentials . . . . .	240
10.2.1.	<b>PMIx_Validate_credential</b> . . . . .	240
10.2.2.	<b>PMIx_Validate_credential_nb</b> . . . . .	242
<b>11. Server-Specific Interfaces</b>		<b>245</b>
11.1.	Server-Specific Attributes . . . . .	245
11.2.	Server Initialization and Finalization . . . . .	245
11.2.1.	<b>PMIx_server_init</b> . . . . .	245
11.2.2.	<b>PMIx_server_finalize</b> . . . . .	248
11.3.	Server Support Functions . . . . .	249
11.3.1.	<b>PMIx_generate_regex</b> . . . . .	249
11.3.2.	<b>PMIx_generate_ppn</b> . . . . .	250
11.3.3.	<b>PMIx_server_register_nspace</b> . . . . .	250
11.3.3.1.	Assembling the registration information . . . . .	259
11.3.4.	<b>PMIx_server_deregister_nspace</b> . . . . .	266
11.3.5.	<b>PMIx_server_register_client</b> . . . . .	268
11.3.6.	<b>PMIx_server_deregister_client</b> . . . . .	269
11.3.7.	<b>PMIx_server_setup_fork</b> . . . . .	270
11.3.8.	<b>PMIx_server_dmodex_request</b> . . . . .	270
11.3.9.	<b>PMIx_server_setup_application</b> . . . . .	272
11.3.10.	<b>PMIx_Register_attributes</b> . . . . .	274
11.3.11.	<b>PMIx_server_setup_local_support</b> . . . . .	276
11.3.12.	<b>PMIx_server_IOF_deliver</b> . . . . .	277
11.3.13.	<b>PMIx_server_collect_inventory</b> . . . . .	278
11.3.14.	<b>PMIx_server_deliver_inventory</b> . . . . .	279
11.3.15.	<b>PMIx_generate_locality_string</b> . . . . .	281
11.3.15.1.	Cpuset Structure . . . . .	281
11.3.16.	Cpuset support macros . . . . .	282
11.3.16.1.	Initialize the cpuset structure . . . . .	282
11.4.	Server Function Pointers . . . . .	282
11.4.1.	<b>pmix_server_module_t</b> Module . . . . .	282
11.4.2.	<b>pmix_server_client_connected_fn_t</b> . . . . .	283
11.4.3.	<b>pmix_server_client_finalized_fn_t</b> . . . . .	285
11.4.4.	<b>pmix_server_abort_fn_t</b> . . . . .	286

11.4.5. <code>pmix_server_fencenb_fn_t</code> . . . . .	287
11.4.6. <code>pmix_server_dmodex_req_fn_t</code> . . . . .	291
11.4.7. <code>pmix_server_publish_fn_t</code> . . . . .	292
11.4.8. <code>pmix_server_lookup_fn_t</code> . . . . .	294
11.4.9. <code>pmix_server_unpublish_fn_t</code> . . . . .	296
11.4.10. <code>pmix_server_spawn_fn_t</code> . . . . .	298
11.4.11. <code>pmix_server_connect_fn_t</code> . . . . .	303
11.4.12. <code>pmix_server_disconnect_fn_t</code> . . . . .	305
11.4.13. <code>pmix_server_register_events_fn_t</code> . . . . .	307
11.4.14. <code>pmix_server_deregister_events_fn_t</code> . . . . .	309
11.4.15. <code>pmix_server_notify_event_fn_t</code> . . . . .	311
11.4.16. <code>pmix_server_listener_fn_t</code> . . . . .	312
11.4.17. <code>pmix_server_query_fn_t</code> . . . . .	313
11.4.18. <code>pmix_server_tool_connection_fn_t</code> . . . . .	316
11.4.19. <code>pmix_server_log_fn_t</code> . . . . .	317
11.4.20. <code>pmix_server_alloc_fn_t</code> . . . . .	319
11.4.21. <code>pmix_server_job_control_fn_t</code> . . . . .	322
11.4.22. <code>pmix_server_monitor_fn_t</code> . . . . .	325
11.4.23. <code>pmix_server_get_cred_fn_t</code> . . . . .	328
11.4.24. <code>pmix_server_validate_cred_fn_t</code> . . . . .	329
11.4.25. <code>pmix_server_iof_fn_t</code> . . . . .	331
11.4.26. <code>pmix_server_stdin_fn_t</code> . . . . .	334
11.4.27. <code>pmix_server_grp_fn_t</code> . . . . .	335
11.4.28. <code>pmix_server_fabric_fn_t</code> . . . . .	337
<b>12. Fabric Support Definitions</b>	<b>340</b>
12.1. Fabric Support Constants . . . . .	340
12.2. Fabric Support Datatypes . . . . .	340
12.2.1. Fabric Coordinate Structure . . . . .	341
12.2.2. Fabric Coordinate Support Macros . . . . .	342
12.2.2.1. Initialize the coord structure . . . . .	342
12.2.2.2. Destruct the coord structure . . . . .	342
12.2.2.3. Create a coord array . . . . .	342
12.2.2.4. Release a coord array . . . . .	342

12.2.3.	Fabric Coordinate Views . . . . .	343
12.2.4.	Fabric Link State . . . . .	343
12.2.5.	Fabric Operation Constants . . . . .	344
12.2.6.	Fabric registration structure . . . . .	344
12.2.6.1.	Initialize the fabric structure . . . . .	347
12.3.	Fabric Support Attributes . . . . .	348
12.4.	Fabric Support Functions . . . . .	350
12.4.1.	<b>PMIx_Fabric_register</b> . . . . .	351
12.4.2.	<b>PMIx_Fabric_register_nb</b> . . . . .	352
12.4.3.	<b>PMIx_Fabric_update</b> . . . . .	353
12.4.4.	<b>PMIx_Fabric_update_nb</b> . . . . .	353
12.4.5.	<b>PMIx_Fabric_deregister</b> . . . . .	354
12.4.6.	<b>PMIx_Fabric_deregister_nb</b> . . . . .	355
12.4.7.	<b>PMIx_Fabric_get_vertex_info</b> . . . . .	355
12.4.8.	<b>PMIx_Fabric_get_vertex_info_nb</b> . . . . .	358
12.4.9.	<b>PMIx_Fabric_get_device_index</b> . . . . .	358
12.4.10.	<b>PMIx_Fabric_get_device_index_nb</b> . . . . .	359
<b>13. Process Sets and Groups</b>		<b>361</b>
13.1.	Process Sets . . . . .	361
13.2.	Process Groups . . . . .	362
13.2.1.	Group Operation Constants . . . . .	364
13.2.2.	<b>PMIx_Group_construct</b> . . . . .	364
13.2.3.	<b>PMIx_Group_construct_nb</b> . . . . .	368
13.2.4.	<b>PMIx_Group_destruct</b> . . . . .	371
13.2.5.	<b>PMIx_Group_destruct_nb</b> . . . . .	373
13.2.6.	<b>PMIx_Group_invite</b> . . . . .	375
13.2.7.	<b>PMIx_Group_invite_nb</b> . . . . .	379
13.2.8.	<b>PMIx_Group_join</b> . . . . .	381
13.2.9.	<b>PMIx_Group_join_nb</b> . . . . .	384
13.2.10.	<b>PMIx_Group_leave</b> . . . . .	386
13.2.11.	<b>PMIx_Group_leave_nb</b> . . . . .	387

<b>14. Tools and Debuggers</b>	<b>389</b>
14.1. Connection Mechanisms . . . . .	389
14.1.1. Rendezvousing with a local server . . . . .	392
14.1.2. Connecting to a remote server . . . . .	393
14.1.3. Attaching to running jobs . . . . .	393
14.1.4. Tool initialization attributes . . . . .	394
14.1.5. Tool initialization environmental variables . . . . .	394
14.1.6. Tool connection attributes . . . . .	395
14.2. Launching Applications with Tools . . . . .	396
14.2.1. Direct launch . . . . .	396
14.2.2. Indirect launch . . . . .	399
14.2.3. Tool spawn-related attributes . . . . .	402
14.2.4. Tool spawn-related constants . . . . .	403
14.3. IO Forwarding . . . . .	403
14.3.1. Forwarding stdout/stderr . . . . .	404
14.3.2. Forwarding stdin . . . . .	406
14.3.3. IO Forwarding attributes . . . . .	407
14.4. Debugger Support . . . . .	408
14.4.1. Co-Location of Debugger Daemons . . . . .	410
14.4.2. Co-Spawn of Debugger Daemons . . . . .	411
14.4.3. Debugger Agents . . . . .	412
14.4.4. Debugger-related constants . . . . .	413
14.4.5. Debugger attributes . . . . .	413
14.5. Tool-Specific APIs . . . . .	414
14.5.1. <b>PMIx_tool_init</b> . . . . .	414
14.5.2. <b>PMIx_tool_finalize</b> . . . . .	417
14.5.3. <b>PMIx_tool_disconnect</b> . . . . .	418
14.5.4. <b>PMIx_tool_connect_to_server</b> . . . . .	418
14.5.5. <b>PMIx_tool_set_server</b> . . . . .	420
14.5.6. <b>PMIx_IOF_pull</b> . . . . .	421
14.5.7. <b>PMIx_IOF_deregister</b> . . . . .	423
14.5.8. <b>PMIx_IOF_push</b> . . . . .	424

<b>A. Python Bindings</b>	<b>427</b>
A.1. Design Considerations . . . . .	427
A.1.1. Error Codes vs Python Exceptions . . . . .	427
A.1.2. Representation of Structured Data . . . . .	427
A.2. Datatype Definitions . . . . .	428
A.2.1. Example . . . . .	433
A.3. Callback Function Definitions . . . . .	433
A.3.1. IOF Delivery Function . . . . .	433
A.3.2. Event Handler . . . . .	434
A.3.3. Server Module Functions . . . . .	435
A.3.3.1. Client Connected . . . . .	435
A.3.3.2. Client Finalized . . . . .	435
A.3.3.3. Client Aborted . . . . .	436
A.3.3.4. Fence . . . . .	436
A.3.3.5. Direct Modex . . . . .	437
A.3.3.6. Publish . . . . .	437
A.3.3.7. Lookup . . . . .	438
A.3.3.8. Unpublish . . . . .	438
A.3.3.9. Spawn . . . . .	439
A.3.3.10. Connect . . . . .	439
A.3.3.11. Disconnect . . . . .	440
A.3.3.12. Register Events . . . . .	440
A.3.3.13. Deregister Events . . . . .	441
A.3.3.14. Notify Event . . . . .	441
A.3.3.15. Query . . . . .	441
A.3.3.16. Tool Connected . . . . .	442
A.3.3.17. Log . . . . .	442
A.3.3.18. Allocate Resources . . . . .	443
A.3.3.19. Job Control . . . . .	443
A.3.3.20. Monitor . . . . .	444
A.3.3.21. Get Credential . . . . .	444
A.3.3.22. Validate Credential . . . . .	445
A.3.3.23. IO Forward . . . . .	445

A.3.3.24. IO Push . . . . .	446
A.3.3.25. Group Operations . . . . .	446
A.3.3.26. Fabric Operations . . . . .	447
A.4. PMIxClient . . . . .	448
A.4.1. Client.init . . . . .	448
A.4.2. Client.initialized . . . . .	448
A.4.3. Client.get_version . . . . .	449
A.4.4. Client.finalize . . . . .	449
A.4.5. Client.abort . . . . .	449
A.4.6. Client.store_internal . . . . .	450
A.4.7. Client.put . . . . .	450
A.4.8. Client.commit . . . . .	451
A.4.9. Client.fence . . . . .	451
A.4.10. Client.get . . . . .	452
A.4.11. Client.publish . . . . .	452
A.4.12. Client.lookup . . . . .	453
A.4.13. Client.unpublish . . . . .	453
A.4.14. Client.spawn . . . . .	454
A.4.15. Client.connect . . . . .	454
A.4.16. Client.disconnect . . . . .	455
A.4.17. Client.resolve_peers . . . . .	455
A.4.18. Client.resolve_nodes . . . . .	456
A.4.19. Client.query . . . . .	456
A.4.20. Client.log . . . . .	457
A.4.21. Client.allocate . . . . .	457
A.4.22. Client.job_ctrl . . . . .	458
A.4.23. Client.monitor . . . . .	458
A.4.24. Client.get_credential . . . . .	459
A.4.25. Client.validate_credential . . . . .	459
A.4.26. Client.group_construct . . . . .	460
A.4.27. Client.group_invite . . . . .	460
A.4.28. Client.group_join . . . . .	461
A.4.29. Client.group_leave . . . . .	462

A.4.30.	Client.group_destruct . . . . .	462
A.4.31.	Client.register_event_handler . . . . .	462
A.4.32.	Client.deregister_event_handler . . . . .	463
A.4.33.	Client.notify_event . . . . .	463
A.4.34.	Client.fabric_register . . . . .	464
A.4.35.	Client.fabric_update . . . . .	464
A.4.36.	Client.fabric_deregister . . . . .	465
A.4.37.	Client.fabric_get_vertex_info . . . . .	465
A.4.38.	Client.fabric_get_device_index . . . . .	466
A.4.39.	Client.error_string . . . . .	466
A.4.40.	Client.proc_state_string . . . . .	466
A.4.41.	Client.scope_string . . . . .	467
A.4.42.	Client.persistence_string . . . . .	467
A.4.43.	Client.data_range_string . . . . .	468
A.4.44.	Client.info_directives_string . . . . .	468
A.4.45.	Client.data_type_string . . . . .	468
A.4.46.	Client.alloc_directive_string . . . . .	469
A.4.47.	Client.iof_channel_string . . . . .	469
A.4.48.	Client.job_state_string . . . . .	470
A.4.49.	Client.get_attribute_string . . . . .	470
A.4.50.	Client.get_attribute_name . . . . .	470
A.4.51.	Client.link_state_string . . . . .	471
A.5.	PMIxServer . . . . .	471
A.5.1.	Server.init . . . . .	471
A.5.2.	Server.finalize . . . . .	472
A.5.3.	Server.generate_regex . . . . .	472
A.5.4.	Server.generate_ppn . . . . .	473
A.5.5.	Server.register_nspace . . . . .	473
A.5.6.	Server.deregister_nspace . . . . .	474
A.5.7.	Server.register_client . . . . .	474
A.5.8.	Server.deregister_client . . . . .	475
A.5.9.	Server.setup_fork . . . . .	475
A.5.10.	Server.dmodex_request . . . . .	476

A.5.11.	Server.setup_application . . . . .	476
A.5.12.	Server.register_attributes . . . . .	477
A.5.13.	Server.setup_local_support . . . . .	477
A.5.14.	Server.iof_deliver . . . . .	478
A.5.15.	Server.collect_inventory . . . . .	478
A.5.16.	Server.deliver_inventory . . . . .	479
A.6.	PMIxTool . . . . .	479
A.6.1.	Tool.init . . . . .	479
A.6.2.	Tool.finalize . . . . .	480
A.6.3.	Tool.connect_to_server . . . . .	480
A.6.4.	Tool.iof_pull . . . . .	481
A.6.5.	Tool.iof_deregister . . . . .	481
A.6.6.	Tool.iof_push . . . . .	482
A.7.	Example Usage . . . . .	482
A.7.1.	Python Client . . . . .	483
A.7.2.	Python Server . . . . .	485
<b>B. Acknowledgements</b>		<b>489</b>
B.1.	Version 3.0 . . . . .	489
B.2.	Version 2.0 . . . . .	490
B.3.	Version 1.0 . . . . .	491
<b>Bibliography</b>		<b>492</b>
<b>Index</b>		<b>493</b>
<b>Index of APIs</b>		<b>494</b>
<b>Index of Support Macros</b>		<b>500</b>
<b>Index of Data Structures</b>		<b>503</b>
<b>Index of Constants</b>		<b>505</b>
<b>Index of Environmental Variables</b>		<b>512</b>



## CHAPTER 1

# Introduction

---

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

## 1.1 Background

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

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

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

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

## 1.2 PMIx Architecture Overview

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

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

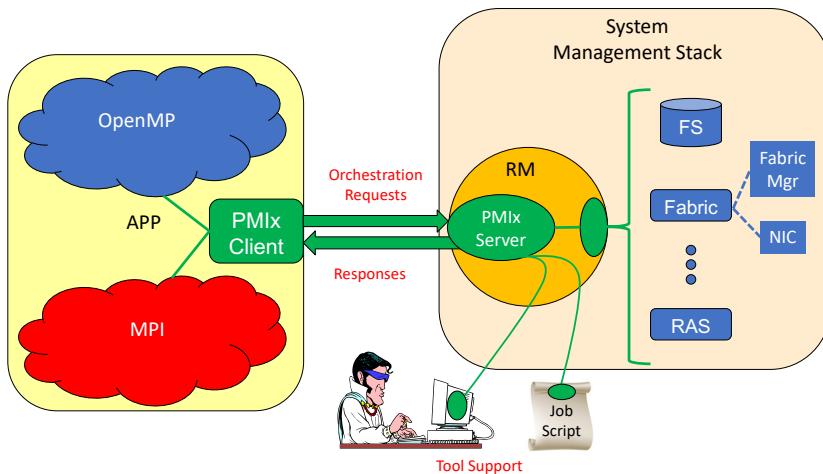


Figure 1.1.: PMIx-SMS Interactions

Fig. 1.1 shows a typical PMIx implementation in which the application is built against a PMIx client library that contains the client-side APIs, attribute definitions, and communication support for interacting with the local PMIx server. PMIx clients are processes which are started through the PMIx infrastructure, either by the PMIx implementation directly or through 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 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    1.3.1    Optional Nature of Attributes

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

8    An application can adapt to the attribute support on a particular system in one of two ways. PMIx  
9    provides an API to enable an application to query the attributes supported by a particular API (See  
10   See 7.1.4.2). Through this API, the PMIx implementation can provide detailed information about the  
11   attributes supported on a system for each API call queried. Alternatively, the application can mark  
12   attributes as required using a flag within the [pmix\\_info\\_t](#) (See 3.2.18). If the required attribute  
13   is not available on the system or the desired value for the attribute is not available, the call will  
14   return the error code for *not supported*.

15   For example, the [PMIX\\_TIMEOUT](#) attribute can be used to specify the time (in seconds) before the  
16   requested operation should time out. The intent of this attribute is to allow the client to avoid  
17   “hanging” in a request that takes longer than the client wishes to wait, or may never return (e.g., a  
18   [PMIx\\_Fence](#) that a blocked participant never enters).

19   The application can query the attribute support for [PMIx\\_Fence](#) and search whether  
20   [PMIX\\_TIMEOUT](#) is listed as a supported attribute. The application can also set the required flag in  
21   the [pmix\\_info\\_t](#) for that attribute when making the [PMIx\\_Fence](#) call. This will return an  
22   error if this attribute is not supported. If the required flag is not set, the library and SMS host are  
23   allowed to treat the attribute as optional, ignoring it if support is not available.

24   It is therefore critical that users and application implementers:

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

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

36   A PMIx implementation may be able to support only a subset of the PMIx API and attributes on a  
37   particular system due to either its own limitations or limitations of the SMS with which it  
38   interfaces. A PMIx implementation may also provide additional attributes beyond those defined  
39   herein in order to allow applications to access the full features of the underlying SMS. PMIx

1 implementations are strongly advised to document the PMIx interfaces and associated attributes  
2 they support, with any annotations about behavior limitations. The PMIx standard cannot define  
3 this support for implementations, but such documentation is valuable to PMIx clients desiring to  
4 support a broad range of systems.

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

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

## 14 **1.4 Organization of this document**

15 The remainder of this document is structured as follows:

- 16     • Introduction and Overview in Chapter 1 on page 1
- 17     • Terms and Conventions in Chapter 2 on page 13
- 18     • Data Structures and Types in Chapter 3 on page 19
- 19     • PMIx Initialization and Finalization in Chapter 4 on page 120
- 20     • Key/Value Management in Chapter 5 on page 125
- 21     • Process Management in Chapter 6 on page 154
- 22     • Job Management in Chapter 7 on page 179
- 23     • Event Notification in Chapter 8 on page 216
- 24     • Data Packing and Unpacking in Chapter 9 on page 226
- 25     • Security in Chapter 10 on page 236
- 26     • PMIx Server Specific Interfaces in Chapter 11 on page 245
- 27     • Scheduler-Specific Interface in Chapter ?? on page ??
- 28     • Process Sets and Groups in Chapter 13 on page 361
- 29     • Network Coordinates in Chapter ?? on page ??
- 30     • Python Bindings in Appendix A on page 427

## 1 1.5 Version 1.0: June 12, 2015

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

5 • Client APIs

- 6 – `PMIx_Init`, `PMIx_Initialized`, `PMIx_Abort`, `PMIx_Finalize`
- 7 – `PMIx_Put`, `PMIx_Commit`,
- 8 – `PMIx_Fence`, `PMIx_Fence_nb`
- 9 – `PMIx_Get`, `PMIx_Get_nb`
- 10 – `PMIx_Publish`, `PMIx_Publish_nb`
- 11 – `PMIx_Lookup`, `PMIx_Lookup_nb`
- 12 – `PMIx_Unpublish`, `PMIx_Unpublish_nb`
- 13 – `PMIx_Spawn`, `PMIx_Spawn_nb`
- 14 – `PMIx_Connect`, `PMIx_Connect_nb`
- 15 – `PMIx_Disconnect`, `PMIx_Disconnect_nb`
- 16 – `PMIx_Resolve_nodes`, `PMIx_Resolve_peers`

17 • Server APIs

- 18 – `PMIx_server_init`, `PMIx_server_finalize`
- 19 – `PMIx_generate_regex`, `PMIx_generate_ppn`
- 20 – `PMIx_server_register_nspace`, `PMIx_server_deregister_nspace`
- 21 – `PMIx_server_register_client`, `PMIx_server_deregister_client`
- 22 – `PMIx_server_setup_fork`, `PMIx_server_dmodex_request`

23 • Common APIs

- 24 – `PMIx_Get_version`, `PMIx_Store_internal`, `PMIx_Error_string`
- 25 – `PMIx_Register_errhandler`, `PMIx_Deregister_errhandler`, `PMIx_Notify_error`

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

## 27 1.6 Version 2.0: Sept. 2018

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

- 1     • Client APIs
  - 2       – `PMIx_Query_info_nb`, `PMIx_Log_nb`
  - 3       – `PMIx_Allocation_request_nb`, `PMIx_Job_control_nb`,  
4        `PMIx_Process_monitor_nb`, `PMIx_Heartbeat`
- 5     • Server APIs
  - 6       – `PMIx_server_setup_application`, `PMIx_server_setup_local_support`
- 7     • Tool APIs
  - 8       – `PMIx_tool_init`, `PMIx_tool_finalize`
- 9     • Common APIs
  - 10      – `PMIx_Register_event_handler`, `PMIx_Deregister_event_handler`
  - 11      – `PMIx_Notify_event`
  - 12      – `PMIx_Proc_state_string`, `PMIx_Scope_string`
  - 13      – `PMIx_Persistence_string`, `PMIx_Data_range_string`
  - 14      – `PMIx_Info_directives_string`, `PMIx_Data_type_string`
  - 15      – `PMIx_Alloc_directive_string`
  - 16      – `PMIx_Data_pack`, `PMIx_Data_unpack`, `PMIx_Data_copy`
  - 17      – `PMIx_Data_print`, `PMIx_Data_copy_payload`

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

## 22    1.7 Version 2.1: Dec. 2018

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

- 25     • Clarify description of `PMIx_Connect` and `PMIx_Disconnect` APIs.
- 26     • Explain that values for the `PMIX_COLLECTIVE_ALGO` are environment-dependent
- 27     • Identify the namespace/rank values required for retrieving attribute-associated information using  
28       the `PMIx_Get` API
- 29     • Provide definitions for `session`, `job`, `application`, and other terms used throughout the  
30       document

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

## 1.8 Version 2.2: Jan 2019

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

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

## 1.9 Version 3.0: Dec. 2018

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

- Client APIs
  - `PMIx_Log`, `PMIx_Job_control`

- `PMIx_Allocation_request`, `PMIx_Process_monitor`
- `PMIx_Get_credential`, `PMIx_Validate_credential`
- Server APIs
  - `PMIx_server_IOF_deliver`
  - `PMIx_server_collect_inventory`, `PMIx_server_deliver_inventory`
- Tool APIs
  - `PMIx_IOF_pull`, `PMIx_IOF_push`, `PMIx_IOF_deregister`
  - `PMIx_tool_connect_to_server`
- Common APIs
  - `PMIx_IOF_channel_string`

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

## 1.10 Version 3.1: Jan. 2019

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

- Direct modex upcall function (`pmix_server_dmodex_req_fn_t`) cannot complete atomically as the API cannot return the requested information except via the provided callback function
- Fix typo in name of `PMIX_FWD_STDDIAG` attribute
- Correctly identify the information retrieval and storage attributes as “new” to v3 of the standard
  - Add missing `pmix_data_array_t` definition and support macros
  - Add a rule divider between implementer and host environment required attributes for clarity
  - Add `PMIX_QUERY_QUALIFIERS_CREATE` macro to simplify creation of `pmix_query_t` qualifiers
- Add `PMIX_APP_INFO_CREATE` macro to simplify creation of `pmix_app_t` directives
- Add new attributes to specify the level of information being requested where ambiguity may exist (see 3.4.10)
- Add new attributes to assemble information by its level for storage where ambiguity may exist (see 3.4.11)

- Add flag and `PMIX_INFO_IS_END` macro for marking and detecting the end of a `pmix_info_t` array
- Clarify that `PMIX_NUM_SLOTS` is duplicative of (a) `PMIX_UNIV_SIZE` when used at the `session` level and (b) `PMIX_MAX_PROCS` when used at the `job` and `application` levels, but leave it in for backward compatibility.
- Clarify difference between `PMIX_JOB_SIZE` and `PMIX_MAX_PROCS`
- Clarify that `PMIx_server_setup_application` must be called per-`job` instead of per-`application` as the name implies. Unfortunately, this is a historical artifact. Note that both `PMIX_NODE_MAP` and `PMIX_PROC_MAP` must be included as input in the `info` array provided to that function. Further descriptive explanation of the “instant on” procedure will be provided in the next version of the PMIx Standard.
- Clarify how the PMIx server expects data passed to the host by `pmix_server_fencenb_fn_t` should be aggregated across nodes, and provide a code snippet example

## 1.11 Version 3.2: Oct. 2019

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

- Correct an error in the `PMIx_Allocation_request` function signature, and clarify the allocation ID attributes
- Rename the `PMIX_ALLOC_ID` attribute to `PMIX_ALLOC_REQ_ID` to clarify that this is a string the user provides as a means to identify their request to query status
- Add a new `PMIX_ALLOC_ID` attribute that contains the identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to `PMIx_Spawn`

## 1.12 Version 4.0: Sept 2020

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 datatype to support generation and parsing of PMIx locality 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.

The above changes included introduction of the following APIs:

- 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`
- Server APIs
  - `PMIx_Fabric_register`, `PMIx_Fabric_register_nb`
  - `PMIx_Fabric_update`, `PMIx_Fabric_update_nb`

- 1     – `PMIx_Fabric_deregister`, `PMIx_Fabric_deregister_nb`
- 2     – `PMIx_Fabric_get_vertex_info`, `PMIx_Fabric_get_vertex_info_nb`
- 3     – `PMIx_Fabric_get_device_index`, `PMIx_Fabric_get_device_index_nb`
- 4     – `PMIx_generate_locality_string`
- 5     – `PMIx_Register_attributes`
- 6     – `pmix_server_grp_fn_t`, `pmix_server_fabric_fn_t`
- 7     • Tool APIs
  - 8         – `PMIx_tool_disconnect`
  - 9         – `PMIx_tool_set_server`

## CHAPTER 2

# PMIx Terms and Conventions

---

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

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

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

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

### Advice to users

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

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

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

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

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

15    This document borrows freely from other standards (most notably from the Message Passing  
16    Interface (MPI) and OpenMP standards) in its use of notation and conventions in an attempt to  
17    reduce confusion. The following sections provide an overview of the conventions used throughout  
18    the PMIx Standard document.

## 19    **2.1 Notational Conventions**

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

22    C specific text...

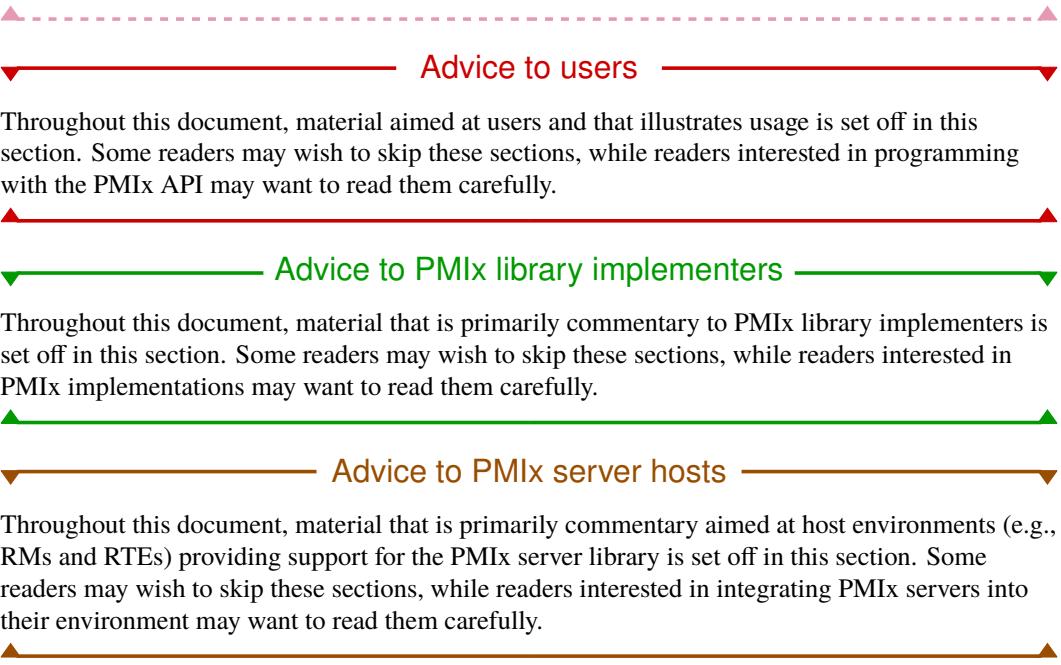
23    int foo = 42;

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

26    Note: General text...

### Rationale

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



## Advice to users

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

## Advice to PMIx library implementers

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

## Advice to PMIx server hosts

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

## 2.2 Semantics

The following terms will be taken to mean:

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

## 1    2.3 Naming Conventions

2    The PMIx standard has adopted the following conventions:

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

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

## 15   2.4 Procedure Conventions

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

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

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

31   A similar model is used for the server module functions (see [11.4.1](#)). In this case, the PMIx library  
32   is making an upcall into its host via the PMIx server module function and passing a specific cbfunc  
33   and cbdata. The PMIx library expects the host to call the cbfunc with the necessary arguments and

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

## 5 **2.5 Standard vs Reference Implementation**

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

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

## CHAPTER 3

# Data Structures and Types

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

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

### Advice to PMIx library implementers

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

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

### Advice to users

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

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

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

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

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

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

## 5 3.1 Constants

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

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

### Advice to PMIx library implementers

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

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

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

### Advice to PMIx library implementers

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

### 3.1.1 PMIx Error Constants

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

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

#### Advice to PMIx library implementers

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

#### Advice to users

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

#### 3.1.1.1 General Error Constants

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

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

```

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

```

```

1   PMIX_JCTRL_PREEMPT_ALERT    Job control: Monitored by PMIx client to detect an RM
2       intending to preempt the job.
3   PMIX_MONITOR_HEARTBEAT_ALERT  Job monitoring: Heartbeat alert
4   PMIX_MONITOR_FILE_ALERT      Job monitoring: File alert
5   PMIX_PROC_TERMINATED        Process terminated - can be either normal or abnormal
6       termination
7   PMIX_ERR_INVALID_TERMINATION Process terminated without calling
8       PMIx_Finalize, or was a member of an assemblage formed via PMIx_Connect and
9       terminated or called PMIx_Finalize without first calling PMIx_Disconnect (or its
10      non-blocking form) from that assemblage.

```

### 3.1.1.2 Job-Related Error Constants

```

12  PMIX_ERR_JOB_FAILED_TO_START  At least one process in the job failed to start
13  PMIX_ERR_JOB_APP_NOT_EXECUTABLE A specified application executable is does not
14      have execution permissions
15  PMIX_ERR_JOB_NO_EXE_SPECIFIED  No executable was specified in a spawn request
16  PMIX_ERR_JOB_FAILED_TO_MAP     The RM was unable to map (i.e., assign locations) the
17      processes in the job
18  PMIX_ERR_JOB_CANCELLED        The submitted job was canceled prior to launch
19  PMIX_ERR_JOB_FAILED_TO_LAUNCH The job failed to launch
20  PMIX_ERR_JOB_ABORTED         The job was aborted due to at least one process calling
21      PMIx_Abort
22  PMIX_ERR_JOB_KILLED_BY_CMD   The job was killed in response to a user command
23  PMIX_ERR_JOB_ABORTED_BY_SIG  The job was killed in response to a signal (e.g.,
24      SIGTERM or SIGKILL)
25  PMIX_ERR_JOB_TERM_WO_SYNC   The job was killed due to at least one process terminating
26      without first calling PMIx_Finalize
27  PMIX_ERR_JOB_SENSOR_BOUND_EXCEEDED The job was killed due to at least one
28      process exceeding a monitor threshold
29  PMIX_ERR_JOB_NEVER_LAUNCHED  The job was never launched
30  PMIX_ERR_JOB_NON_ZERO_TERM   The job was killed due to at least one process exiting
31      with a non-zero status
32  PMIX_ERR_JOB_ALLOC_FAILED   The job was unable to obtain an allocation
33  PMIX_ERR_JOB_CANNOT_LAUNCH The resources required by the job within the given
34      allocation are busy, thus preventing the job from being launched

```

### 3.1.1.3 Operational Error Constants

```

36  PMIX_ERR_EVENT_REGISTRATION  Error in event registration
37  PMIX_ERR_JOB_TERMINATED     Changed to PMIX_EVENT_JOB_END
38  PMIX_ERR_UPDATE_ENDPOINTS  Error updating endpoints
39  PMIX_MODEL_DECLARED        Model declared
40  PMIX_GDS_ACTION_COMPLETE   The global data storage (GDS) action has completed

```

1           **PMIX\_ERR\_INVALID\_OPERATION**    The requested operation is supported by the  
2           implementation and host environment, but fails to meet a requirement (e.g., requesting to  
3           *disconnect* from processes without first *connecting* to them).  
4           **PMIX\_PROC\_HAS\_CONNECTED**    A tool or client has connected to the PMIx server  
5           **PMIX\_CONNECT\_REQUESTED**    Connection has been requested by a PMIx-based tool  
6           **PMIX\_MODEL\_RESOURCES**    Resource usage by a programming model has changed  
7           **PMIX\_OPENMP\_PARALLEL\_ENTERED**    An OpenMP parallel code region has been entered  
8           **PMIX\_OPENMP\_PARALLEL\_EXITED**    An OpenMP parallel code region has completed  
9           **PMIX\_OPERATION\_IN\_PROGRESS**    A requested operation is already in progress  
10          **PMIX\_OPERATION\_SUCCEEDED**    The requested operation was performed atomically - no  
11          callback function will be executed  
12          **PMIX\_ERR\_PARTIAL\_SUCCESS**    The operation is considered successful but not all elements  
13          of the operation were concluded (e.g., some members of a group construct operation chose  
14          not to participate)  
15          **PMIX\_ERR\_DUPLICATE\_KEY**    The provided key has already been published on a different  
16          data range  
17          **PMIX\_ERR\_INVALID\_OPERATION**    The requested operation is not valid - this can possibly  
18          indicate the inclusion of conflicting directives or a request to perform an operation that  
19          conflicts with an ongoing one.  
20          **PMIX\_GROUP\_INVITED**    The process has been invited to join a PMIx Group - the identifier of  
21          the group and the ID's of other invited (or already joined) members will be included in the  
22          notification  
23          **PMIX\_GROUP\_LEFT**    A process has asynchronously left a PMIx Group - the process identifier  
24          of the departing process will in included in the notification  
25          **PMIX\_GROUP\_MEMBER\_FAILED**    A member of a PMIx Group has abnormally terminated  
26          (i.e., without formally leaving the group prior to termination) - the process identifier of the  
27          failed process will in included in the notification  
28          **PMIX\_GROUP\_INVITE\_ACCEPTED**    A process has accepted an invitation to join a PMIx  
29          Group - the identifier of the group being joined will be included in the notification  
30          **PMIX\_GROUP\_INVITE\_DECLINED**    A process has declined an invitation to join a PMIx  
31          Group - the identifier of the declined group will be included in the notification  
32          **PMIX\_GROUP\_INVITE\_FAILED**    An invited process failed or terminated prior to responding  
33          to the invitation - the identifier of the failed process will be included in the notification.  
34          **PMIX\_GROUP\_MEMBERSHIP\_UPDATE**    The membership of a PMIx group has changed - the  
35          identifiers of the revised membership will be included in the notification.  
36          **PMIX\_GROUP\_CONSTRUCT\_ABORT**    Any participant in a PMIx group construct operation  
37          that returns **PMIX\_GROUP\_CONSTRUCT\_ABORT** from the *leader\_failed* event handler will  
38          cause all participants to receive an event notifying them of that status. Similarly, the leader  
39          may elect to abort the procedure by either returning this error code from the handler assigned  
40          to the **PMIX\_GROUP\_INVITE\_ACCEPTED** or **PMIX\_GROUP\_INVITE\_DECLINED**  
41          codes, or by generating an event for the abort code. Abort events will be sent to all invited or  
42          existing members of the group.

1           **PMIX\_GROUP\_CONSTRUCT\_COMPLETE**    The group construct operation has completed - the  
2           final membership will be included in the notification.  
3           **PMIX\_GROUP\_LEADER\_FAILED**    The current *leader* of a group including this process has  
4           abnormally terminated - the group identifier will be included in the notification.  
5           **PMIX\_GROUP\_LEADER\_SELECTED**    A new *leader* of a group including this process has been  
6           selected - the identifier of the new leader will be included in the notification  
7           **PMIX\_GROUP\_CONTEXT\_ID\_ASSIGNED**    A new Process Group Context  
8           IDentifier (PGCID) has been assigned by the host environment to a group that includes this  
9           process - the group identifier will be included in the notification.  
10          **PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION**    The attributes for an identical function have  
11          already been registered at the specified level (host, server, or client)  
12          **PMIX\_ERR\_IOF\_FAILURE**    An IO forwarding operation failed - the affected channel will be  
13          included in the notification  
14          **PMIX\_ERR\_IOF\_COMPLETE**    IO forwarding of the standard input for this process has  
15          completed - i.e., the stdin file descriptor has closed  
16          **PMIX\_ERR\_GET\_MALLOC\_REQD**    The data returned by **PMIx\_Get** contains values that  
17          required dynamic memory allocations (i.e., "malloc"), despite a request for static pointers to  
18          the values in the key-value store. User is responsible for releasing the memory when done  
19          with the information.

### 20        3.1.1.4 Job-related constants

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

22          **PMIX\_EVENT\_JOB\_START**    The first process in the job has been spawned - includes  
23           **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_JOBID** and/or **PMIX\_NSPACE** of the  
24          job.  
25          **PMIX\_LAUNCH\_COMPLETE**    All processes in the job have been spawned - includes  
26           **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_JOBID** and/or **PMIX\_NSPACE** of the  
27          job.  
28          **PMIX\_EVENT\_JOB\_END**    All processes in the job have terminated - includes  
29           **PMIX\_EVENT\_TIMESTAMP** when the last process terminated as well as the **PMIX\_JOBID**  
30          and/or **PMIX\_NSPACE** of the job.  
31          **PMIX\_EVENT\_SESSION\_START**    The allocation has been instantiated and is ready for use -  
32          includes **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_SESSION\_ID** of the  
33          allocation. This event is issued after any system-controlled prologue has completed, but  
34          before any user-specified actions are taken.  
35          **PMIX\_EVENT\_SESSION\_END**    The allocation has terminated - includes  
36           **PMIX\_EVENT\_TIMESTAMP** as well as the **PMIX\_SESSION\_ID** of the allocation. This  
37          event is issued after any user-specified actions have completed, but before any  
38          system-controlled epilogue is performed.

39          The next set of constants are used to indicate that a job has abnormally terminated and to convey  
40          some information as to the cause:

1           **PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE**     The specified application executable either  
2                could not be found, or lacks execution privileges.  
3           **PMIX\_ERR\_JOB\_NO\_EXE\_SPECIFIED**     The job request did not specify an executable.  
4           **PMIX\_ERR\_JOB\_FAILED\_TO\_MAP**     The launcher was unable to map the processes for the  
5                specified job request.  
6           **PMIX\_ERR\_JOB\_CANCELED**     The job was canceled by the host environment  
7           **PMIX\_ERR\_JOB\_FAILED\_TO\_LAUNCH**     One or more processes in the job request failed to  
8                launch  
9           **PMIX\_ERR\_JOB\_ABORTED**     One or more processes in the job called abort, causing the job to  
10               be terminated  
11           **PMIX\_ERR\_JOB\_KILLED\_BY\_CMD**     The job was killed by user command  
12           **PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG**     The job was aborted due to receipt of an error signal  
13               (e.g., SIGKILL)  
14           **PMIX\_ERR\_JOB\_TERM\_WO\_SYNC**     The job was terminated due to one or more processes  
15               exiting without first calling **PMIx\_Finalize**  
16           **PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED**     The job was terminated due to one or more  
17               processes exceeding a specified sensor limit  
18           **PMIX\_ERR\_JOB\_NON\_ZERO\_TERM**     The job was terminated due to one or more processes  
19               exiting with a non-zero status  
20           **PMIX\_ERR\_JOB\_ALLOC\_FAILED**     The job request could not be executed due to failure to  
21               obtain the specified allocation  
22           **PMIX\_ERR\_JOB\_ABORTED\_BY\_SYS\_EVENT**     The job was aborted due to receipt of a  
23               system event

### 24     **3.1.1.5 System error constants**

25           **PMIX\_ERR\_SYS\_BASE**     Mark the beginning of a dedicated range of constants for system event  
26               reporting.  
27           **PMIX\_ERR\_NODE\_DOWN**     A node has gone down - the identifier of the affected node will be  
28               included in the notification  
29           **PMIX\_ERR\_NODE\_OFFLINE**     A node has been marked as *offline* - the identifier of the affected  
30               node will be included in the notification  
31           **PMIX\_ERR\_SYS\_OTHER**     Mark the end of a dedicated range of constants for system event  
32               reporting.

### 33     **3.1.1.6 Event handler error constants**

34           **PMIX\_EVENT\_NO\_ACTION\_TAKEN**     Event handler: No action taken  
35           **PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN**     Event handler: Partial action taken  
36           **PMIX\_EVENT\_ACTION\_DEFERRED**     Event handler: Action deferred  
37           **PMIX\_EVENT\_ACTION\_COMPLETE**     Event handler: Action complete

### 3.1.1.7 User-Defined Error Constants

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

**PMIX\_EXTERNAL\_ERR\_BASE** A starting point for user-level defined error constants.

Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions should always be based on the **PMIX\_EXTERNAL\_ERR\_BASE** constant and not a specific value as the value of the constant may change.

## 3.1.2 Macros for use with PMIx constants

### 3.1.2.1 Detect system event constant

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

PMIx v2.2

C

**PMIX\_SYSTEM\_EVENT (a)**

C

IN a

Error constant to be checked ( **pmix\_status\_t** )

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

## 3.2 Data Types

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

### 3.2.1 Key Structure

The **pmix\_key\_t** structure is a statically defined character array of length **PMIX\_MAX\_KEYLEN** +1, thus supporting keys of maximum length **PMIX\_MAX\_KEYLEN** while preserving space for a mandatory **NULL** terminator.

PMIx v2.0

C

**typedef char pmix\_key\_t [PMIX\_MAX\_KEYLEN+1];**

C

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

## Advice to users

1 References to keys in PMIx v1 were defined simply as an array of characters of size  
2 **PMIX\_MAX\_KEYLEN+1**. The **pmix\_key\_t** type definition was introduced in version 2 of the  
3 standard. The two definitions are code-compatible and thus do not represent a break in backward  
4 compatibility.

5 Passing a **pmix\_key\_t** value to the standard *sizeof* utility can result in compiler warnings of  
6 incorrect returned value. Users are advised to avoid using *sizeof(pmix\_key\_t)* and instead rely on  
7 the **PMIX\_MAX\_KEYLEN** constant.

### 3.2.1.1 Check key macro

Compare the key in a **pmix\_info\_t** to a given value

PMIx v3.0

C

**PMIX\_CHECK\_KEY(a, b)**

C

IN a

Pointer to the structure whose key is to be checked (pointer to **pmix\_info\_t**)

IN b

String value to be compared against (**char\***)

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

### 3.2.1.2 Load key macro

Load a key into a **pmix\_info\_t**

PMIx v4.0

C

**PMIX\_LOAD\_KEY(a, b)**

C

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.

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 Check namespace macro

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

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

### 1 3.2.2.2 Load namespace macro

2 Load a namespace into a `pmix_nspace_t`

PMIx v4.0

C

3 `PMIX_LOAD_NSPACE (a, b)`

C

4 IN **a**

5 Pointer to the target structure (pointer to `pmix_nspace_t`)

6 IN **b**

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

9 No return value.

### 10 3.2.3 Rank Structure

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

PMIx v1.0

C

12 `typedef uint32_t pmix_rank_t;`

C

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

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

19 **PMIX\_RANK\_WILDCARD** A value to indicate that the user wants the data for the given key  
20 from every rank that posted that key.

21 **PMIX\_RANK\_LOCAL\_NODE** Special rank value used to define groups of ranks. This constant  
22 defines the group of all ranks on a local node.

23 **PMIX\_RANK\_LOCAL\_PEERS** Special rank value used to define groups of rankss. This  
24 constant defines the group of all ranks on a local node within the same namespace as the  
25 current process.

26 **PMIX\_RANK\_INVALID** An invalid rank value.

27 **PMIX\_RANK\_VALID** Define an upper boundary for valid rank values.

## 1 3.2.4 Process Structure

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

PMIx v1.0

```
4     typedef struct pmix_proc {  
5         pmix_nspace_t nspace;  
6         pmix_rank_t rank;  
7     } pmix_proc_t;
```

C

C

## 8 3.2.5 Process structure support macros

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

### 10 3.2.5.1 Initialize the proc structure

11 Initialize the `pmix_proc_t` fields

PMIx v1.0

```
12     PMIX_PROC_CONSTRUCT(m)
```

C

C

13 IN m

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

### 15 3.2.5.2 Destruct the proc structure

16 Destruct the `pmix_proc_t` fields

```
17     PMIX_PROC_DESTRUCT(m)
```

C

C

18 IN m

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

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

### 1 3.2.5.3 Create a proc array

2 Allocate and initialize an array of `pmix_proc_t` structures

3 PMIx v1.0

C

3 `PMIX_PROC_CREATE (m, n)`

C

4 **INOUT** `m`

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

6 **IN** `n`

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

### 8 3.2.5.4 Free a proc structure

9 Release a `pmix_proc_t` structure

10 PMIx v4.0

C

10 `PMIX_PROC_RELEASE (m)`

C

11 **IN** `m`

12 Pointer to a `pmix_proc_t` structure (handle)

### 13 3.2.5.5 Free a proc array

14 Release an array of `pmix_proc_t` structures

15 PMIx v1.0

C

15 `PMIX_PROC_FREE (m, n)`

C

16 **IN** `m`

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

18 **IN** `n`

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

### 20 3.2.5.6 Load a proc structure

21 Load values into a `pmix_proc_t`

22 PMIx v2.0

C

22 `PMIX_PROC_LOAD (m, n, r)`

C

23 **IN** `m`

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

25 **IN** `n`

26 Namespace to be loaded (`pmix_nspace_t`)

27 **IN** `r`

28 Rank to be assigned (`pmix_rank_t`)

29 No return value. Deprecated in favor of `PMIX_LOAD_PROCID`

### 3.2.5.7 Compare identifiers

Compare two `pmix_proc_t` identifiers



IN `a`

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

IN `b`

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

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

- the ranks are the same value

- one of the ranks is `PMIX_RANK_WILDCARD`

### 3.2.5.8 Load a procID structure

Load values into a `pmix_proc_t`



IN `m`

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

IN `n`

Namespace to be loaded (`pmix_nspace_t`)

IN `r`

Rank to be assigned (`pmix_rank_t`)

### 3.2.5.9 Construct a multi-cluster namespace

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



IN `m`

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

IN `n`

Cluster identifier (`char*`)

IN `n`

Namespace to be loaded (`pmix_nspace_t`)

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

### 3.2.5.10 Parse a multi-cluster namespace

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

PMIx v4.0

C

PMIX\_MULTICLUSTER\_NSPACE\_PARSE(m, n, r)

C

IN m

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

IN n

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

IN n

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

## 3.2.6 Process State Structure

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

### Advice to users

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

PMIX\_PROC\_STATE\_UNDEF Undefined process state  
PMIX\_PROC\_STATE\_PREPPED Process is ready to be launched  
PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY Process launch is underway  
PMIX\_PROC\_STATE\_RESTART Process is ready for restart  
PMIX\_PROC\_STATE\_TERMINATE Process is marked for termination  
PMIX\_PROC\_STATE\_RUNNING Process has been locally `fork`'ed by the RM  
PMIX\_PROC\_STATE\_CONNECTED Process has connected to PMIx server  
PMIX\_PROC\_STATE\_UNTERMINATED Define a “boundary” between the terminated states  
and `PMIX_PROC_STATE_CONNECTED` so users can easily and quickly determine if a  
process is still running or not. Any value less than this constant means that the process has not  
terminated.  
PMIX\_PROC\_STATE\_TERMINATED Process has terminated and is no longer running  
PMIX\_PROC\_STATE\_ERROR Define a boundary so users can easily and quickly determine if  
a process abnormally terminated. Any value above this constant means that the process has  
terminated abnormally.  
PMIX\_PROC\_STATE\_KILLED\_BY\_CMD Process was killed by a command

```

1 PMIX_PROC_STATE_ABORTED      Process was aborted by a call to PMIx_Abort
2 PMIX_PROC_STATE_FAILED_TO_START  Process failed to start
3 PMIX_PROC_STATE_ABORTED_BY_SIG   Process aborted by a signal
4 PMIX_PROC_STATE_TERM_WO_SYNC    Process exited without calling PMIx_Finalize
5 PMIX_PROC_STATE_COMM FAILED    Process communication has failed
6 PMIX_PROC_STATE_SENSOR_BOUND_EXCEEDED  Process exceeded a specified sensor
7 limit
8 PMIX_PROC_STATE_CALLED_ABORT   Process called PMIx_Abort
9 PMIX_PROC_STATE_HEARTBEAT FAILED  Process failed to send heartbeat within
10 specified time limit
11 PMIX_PROC_STATE_MIGRATING     Process failed and is waiting for resources before
12 restarting
13 PMIX_PROC_STATE_CANNOT_RESTART  Process failed and cannot be restarted
14 PMIX_PROC_STATE_TERM_NON_ZERO   Process exited with a non-zero status
15 PMIX_PROC_STATE_FAILED_TO_LAUNCH  Unable to launch process

```

### 3.2.7 Job State Structure

*PMIx v4.0* The **pmix\_job\_state\_t** structure is a **uint8\_t** type for job state values. The following constants can be used to set a variable of the type **pmix\_job\_state\_t**. All values were originally defined in version 4 of the standard unless otherwise marked.

#### Advice to users

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

```

23 PMIX_JOB_STATE_UNDEF      Undefined job state
24 PMIX_JOB_STATE_PREPARED    Job is ready to be launched
25 PMIX_JOB_STATE_LAUNCH_UNDERWAY  Job launch procedure is in progress
26 PMIX_JOB_STATE_RUNNING     Job is running
27 PMIX_JOB_STATE_SUSPENDED    Job has been suspended
28 PMIX_JOB_STATE_CONNECTED    All processes in the job have connected to their PMIx
29 server
30 PMIX_JOB_STATE_TERMINATED   The job has terminated and is no longer running -
31 typically will be accompanied by the job exit status in response to a query
32 PMIX_JOB_STATE_TERMINATED_WITH_ERROR  The job has terminated and is no longer
33 running - typically will be accompanied by the job-related error code in response to a query

```

## 3.2.8 Process Information Structure

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

PMIx v2.0

```
4     typedef struct pmix_proc_info {
5         /** Process structure */
6         pmix_proc_t proc;
7         /** Hostname where process resides */
8         char *hostname;
9         /** Name of the executable */
10        char *executable_name;
11        /** Process ID on the host */
12        pid_t pid;
13        /** Exit code of the process. Default: 0 */
14        int exit_code;
15        /** Current state of the process */
16        pmix_proc_state_t state;
17    } pmix_proc_info_t;
```

C

C

## 3.2.9 Process Information Structure support macros

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

### 3.2.9.1 Initialize the process information structure

Initialize the `pmix_proc_info_t` fields

PMIx v2.0

```
22    PMIX_PROC_INFO_CONSTRUCT(m)
```

C

C

IN m

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

### 3.2.9.2 Destruct the process information structure

Destruct the `pmix_proc_info_t` fields

PMIx v2.0

```
27    PMIX_PROC_INFO_DESTRUCT(m)
```

C

C

IN m

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

### 3.2.9.3 Create a process information array

Allocate and initialize a `pmix_proc_info_t` array

PMIx v2.0

C

`PMIX_PROC_INFO_CREATE (m, n)`

C

INOUT `m`

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

IN `n`

Number of structures to be allocated (`size_t`)

### 3.2.9.4 Free a process information structure

Release a `pmix_proc_info_t` structure

PMIx v2.0

C

`PMIX_PROC_INFO_RELEASE (m)`

C

IN `m`

Pointer to a `pmix_proc_info_t` structure (handle)

### 3.2.9.5 Free a process information array

Release an array of `pmix_proc_info_t` structures

PMIx v2.0

C

`PMIX_PROC_INFO_FREE (m, n)`

C

IN `m`

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

IN `n`

Number of structures in the array (`size_t`)

## 1 3.2.10 Scope of Put Data

2 *PMIx v1.0* The `pmix_scope_t` structure is a `uint8_t` type that defines the scope for data passed to  
3 `PMIx_Put`. The following constants can be used to set a variable of the type `pmix_scope_t`.  
4 All definitions were introduced in version 1 of the standard unless otherwise marked.  
  
5 Specific implementations may support different scope values, but all implementations must support  
6 at least `PMIX_GLOBAL`. If a scope value is not supported, then the `PMIx_Put` call must return  
7 `PMIX_ERR_NOT_SUPPORTED`.  
  
8 **PMIX\_SCOPE\_UNDEF** Undefined scope  
9 **PMIX\_LOCAL** The data is intended only for other application processes on the same node.  
10 Data marked in this way will not be included in data packages sent to remote requestors —  
11 i.e., it is only available to processes on the local node.  
12 **PMIX\_REMOTE** The data is intended solely for applications processes on remote nodes. Data  
13 marked in this way will not be shared with other processes on the same node — i.e., it is only  
14 available to processes on remote nodes.  
15 **PMIX\_GLOBAL** The data is to be shared with all other requesting processes, regardless of  
16 location.  
17 *PMIx v2.0* **PMIX\_INTERNAL** The data is intended solely for this process and is not shared with other  
18 processes.

## 19 3.2.11 Job State Structure

20 *PMIx v4.0* The `pmix_job_state_t` structure is a `uint8_t` type for job state values. The following  
21 constants can be used to set a variable of the type `pmix_job_state_t`. All values were  
22 originally defined in version 4 of the standard unless otherwise marked.

### Advice to users

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

26 **PMIX\_JOB\_STATE\_UNDEF** Undefined job state  
27 **PMIX\_JOB\_STATE\_PREPPED** Job is ready to be launched  
28 **PMIX\_JOB\_STATE\_LAUNCH\_UNDERWAY** Job launch is underway  
29 **PMIX\_JOB\_STATE\_RUNNING** All processes in the job have been spawned  
30 **PMIX\_JOB\_STATE\_SUSPENDED** All processes in the job have been suspended  
31 **PMIX\_JOB\_STATE\_CONNECTED** All processes in the job have connected to their PMIx  
32 server  
33 **PMIX\_JOB\_STATE\_UNTERMINATED** Define a “boundary” between the terminated states  
34 and `PMIX_JOB_STATE_TERMINATED` so users can easily and quickly determine if a job  
35 is still running or not. Any value less than this constant means that the job has not terminated.

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

### 7 3.2.12 Range of Published Data

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

```
13 PMIX_RANGE_UNDEF Undefined range  
14 PMIX_RANGE_RM Data is intended for the host resource manager.  
15 PMIX_RANGE_LOCAL Data is only available to processes on the local node.  
16 PMIX_RANGE_NAMESPACE Data is only available to processes in the same namespace.  
17 PMIX_RANGE_SESSION Data is only available to all processes in the session.  
18 PMIX_RANGE_GLOBAL Data is available to all processes.  
19 PMIX_RANGE_CUSTOM Range is specified in the pmix_info_t associated with this call.  
20 PMIX_RANGE_PROC_LOCAL Data is only available to this process.  
21 PMIX_RANGE_INVALID Invalid value
```

#### Advice to users

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

### 24 3.2.13 Data Persistence Structure

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

```
29 PMIX_PERSIST_INDEF Retain data until specifically deleted.  
30 PMIX_PERSIST_FIRST_READ Retain data until the first access, then the data is deleted.  
31 PMIX_PERSIST_PROC Retain data until the publishing process terminates.  
32 PMIX_PERSIST_APP Retain data until the application terminates.  
33 PMIX_PERSIST_SESSION Retain data until the session/allocation terminates.  
34 PMIX_PERSIST_INVALID Invalid value
```

## 3.2.14 Data Array Structure

PMIx v2.0

C

```
2     typedef struct pmix_data_array
3         pmix_data_type_t type;
4         size_t size;
5         void *array;
6     pmix_data_array_t;
```

C

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

## 3.2.15 Data array structure support macros

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

### 3.2.15.1 Initialize the data array structure

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

PMIx v2.2

C

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

C

IN m

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

IN n

Number of elements in the array (`size_t`)

IN t

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

### 3.2.15.2 Destruct the data array structure

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

PMIx v2.2

C

```
22     PMIX_DATA_ARRAY_DESTRUCT(m)
```

C

IN m

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

### 3.2.15.3 Create and initialize a data array object

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

PMIx v2.2

C

4 `PMIX_DATA_ARRAY_CREATE(m, n, t)`

C

5 **INOUT** `m`

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

7 **IN** `n`

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

9 **IN** `t`

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

### 3.2.15.4 Free a data array object

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

PMIx v2.2

C

13 `PMIX_DATA_ARRAY_FREE(m)`

C

14 **IN** `m`

15 Pointer to the `pmix_data_array_t` structure (handle)

## 3.2.16 Value Structure

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

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

PMIx v1.0

```

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

```

### 3.2.17 Value structure support macros

The following macros are provided to support the [pmix\\_value\\_t](#) structure.

### 3.2.17.1 Initialize the value structure

Initialize the `pmix_value_t` fields

*PMIx v1.0*

C

`PMIX_VALUE_CONSTRUCT(m)`

C

IN m

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

### 3.2.17.2 Destruct the value structure

Destruct the `pmix_value_t` fields

*PMIx v1.0*

C

`PMIX_VALUE_DESTRUCT(m)`

C

IN m

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

### 3.2.17.3 Create a value array

Allocate and initialize an array of `pmix_value_t` structures

*PMIx v1.0*

C

`PMIX_VALUE_CREATE(m, n)`

C

**INOUT** m

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

IN n

Number of structures to be allocated (`size_t`)

### 3.2.17.4 Free a value structure

Release a `pmix_value_t` structure

*PMIx v4.0*

C

`PMIX_VALUE_RELEASE(m)`

C

IN m

Pointer to a `pmix_value_t` structure (handle)

### 1 3.2.17.5 Free a value array

2 Release an array of `pmix_value_t` structures

PMIx v1.0

C

3 `PMIX_VALUE_FREE (m, n)`

C

4 IN `m`

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

6 IN `n`

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

### 8 3.2.17.6 Load a value structure

#### 9 Summary

10 Load data into a `pmix_value_t` structure.

PMIx v2.0

C

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

C

12 IN `v`

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

14 IN `d`

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

16 IN `t`

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

#### 18 Description

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

#### Advice to users

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

### 3.2.17.7 Unload a value structure

#### Summary

Unload data from a `pmix_value_t` structure.

PMIx v2.2

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

C

C

#### OUT r

Status code indicating result of the operation `pmix_status_t`

#### IN v

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

#### INOUT d

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

#### INOUT t

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

#### Description

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

#### Advice to users

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

### 3.2.17.8 Transfer data between value structures

#### Summary

Transfer the data value between two `pmix_value_t` structures.

PMIx v2.0

`PMIX_VALUE_XFER(r, d, s);`

C

C

#### OUT r

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

#### IN d

Pointer to the `pmix_value_t` destination (handle)

#### IN s

Pointer to the `pmix_value_t` source (handle)

1      **Description**

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

4      **Advice to users**

5      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.

6      **3.2.17.9 Retrieve a numerical value from a value struct**

7      Retrieve a numerical value from a `pmix_value_t` structure

PMIx v3.0

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

C

9      **OUT s**

10     Status code for the request (`pmix_status_t`)

11     **IN m**

12     Pointer to the `pmix_value_t` structure (handle)

13     **OUT n**

14     Variable to be set to the value (match expected type)

15     **IN t**

16     Type of number expected in m (`pmix_data_type_t`)

17     Sets the provided variable equal to the numerical value contained in the given `pmix_value_t` ,  
18     returning success if the data type of the value matches the expected type and

19     `PMIX_ERR_BAD_PARAM` if it doesn't

20      **3.2.18 Info Structure**

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

PMIx v1.0

23      `typedef struct pmix_info_t {`  
24        `pmix_key_t key;`  
25        `pmix_info_directives_t flags; // version 2.0`  
26        `pmix_value_t value;`  
27      `} pmix_info_t;`

C

## 3.2.19 Info structure support macros

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

### 3.2.19.1 Initialize the info structure

Initialize the `pmix_info_t` fields

PMIx v1.0

C

`PMIX_INFO_CONSTRUCT(m)`

C

IN m

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

### 3.2.19.2 Destruct the info structure

Destruct the `pmix_info_t` fields

PMIx v1.0

C

`PMIX_INFO_DESTRUCT(m)`

C

IN m

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

### 3.2.19.3 Create an info array

Allocate and initialize an array of info structures

PMIx v1.0

C

`PMIX_INFO_CREATE(m, n)`

C

INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

### 3.2.19.4 Free an info array

Release an array of `pmix_info_t` structures

PMIx v1.0

C

`PMIX_INFO_FREE(m, n)`

C

IN m

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

IN n

Number of structures in the array (`size_t`)

### 3.2.19.5 Load key and value data into a info struct

PMIx v1.0

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

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

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

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

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

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

#### Advice to users

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

### 3.2.19.6 Copy data between info structures

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

PMIx v2.0

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

21 IN d  
22 Pointer to the destination `pmix_info_t` (pointer to `pmix_info_t`)  
23 IN s  
24 Pointer to the source `pmix_info_t` (pointer to `pmix_info_t`)

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

#### Advice to users

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

### 3.2.19.7 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)`

C

IN m

Pointer to a `pmix_info_t` structure (handle)

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

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

### 3.2.20 Info Type Directives

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

#### Advice to users

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

#### Advice to PMIx library implementers

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

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

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

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

7 **PMIX\_INFO\_DIR\_RESERVED** A bit-mask identifying the bits reserved for internal use by  
8 implementers - these currently are set as 0xffff0000.

### Advice to PMIx server hosts

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

## 3.2.21 Info Directive support macros

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

### 3.2.21.1 Mark an info structure as required

#### Summary

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

PMIx v2.0

C

17 `PMIX_INFO_REQUIRED(info);`

C

18 IN `info`

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

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

### 3.2.21.2 Mark an info structure as optional

#### Summary

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

PMIx v2.0

C

24 `PMIX_INFO_OPTIONAL(info);`

C

25 IN `info`

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

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

### 3.2.21.3 Test an info structure for *required* directive

#### Summary

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

PMIx v2.0

```
PMIX_INFO_IS_REQUIRED(info);
```

C

C

IN `info`

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

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

### 3.2.21.4 Test an info structure for *optional* directive

#### Summary

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

PMIx v2.0

```
PMIX_INFO_IS_OPTIONAL(info);
```

C

C

IN `info`

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

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

### 3.2.21.5 Test an info structure for *end of array* directive

#### Summary

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

PMIx v2.2

```
PMIX_INFO_IS_END(info);
```

C

C

IN `info`

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

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

## 1 3.2.22 Job Allocation Directives

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

6     **PMIX\_ALLOC\_NEW** A new allocation is being requested. The resulting allocation will be  
7         disjoint (i.e., not connected in a job sense) from the requesting allocation.  
8     **PMIX\_ALLOC\_EXTEND** Extend the existing allocation, either in time or as additional  
9         resources.  
10    **PMIX\_ALLOC\_RELEASE** Release part of the existing allocation. Attributes in the  
11         accompanying `pmix_info_t` array may be used to specify permanent release of the  
12         identified resources, or “lending” of those resources for some period of time.  
13    **PMIX\_ALLOC\_REAQUIRE** Reacquire resources that were previously “lent” back to the  
14         scheduler.  
15    **PMIX\_ALLOC\_EXTERNAL** A value boundary above which implementers are free to define  
16         their own directive values.

## 17 3.2.23 IO Forwarding Channels

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

21     **PMIX\_FWD\_NO\_CHANNELS** Forward no channels  
22     **PMIX\_FWD\_STDIN\_CHANNEL** Forward stdin  
23     **PMIX\_FWD\_STDOUT\_CHANNEL** Forward stdout  
24     **PMIX\_FWD\_STDERR\_CHANNEL** Forward stderr  
25     **PMIX\_FWD\_STDDIAG\_CHANNEL** Forward stddiag, if available  
26     **PMIX\_FWD\_ALL\_CHANNELS** Forward all available channels

## 27 3.2.24 Environmental Variable Structure

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

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

C

## 3.2.25 Environmental variable support macros

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

### 3.2.25.1 Initialize the envar structure

Initialize the `pmix_envar_t` fields

C

C

**IN** **m**

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

### 3.2.25.2 Destruct the envar structure

Clear the `pmix_envar_t` fields

C

C

**IN** **m**

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

### 3.2.25.3 Create an envar array

Allocate and initialize an array of `pmix_envar_t` structures

C

C

**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`)

#### 1 3.2.25.4 Free an envar array

2 Release an array of `pmix_envar_t` structures

PMIx v3.0

C

3 `PMIX_ENVAR_FREE(m, n)`

C

4 IN `m`  
5 Pointer to the array of `pmix_envar_t` structures (handle)  
6 IN `n`  
7 Number of structures in the array (`size_t`)

#### 8 3.2.25.5 Load an envar structure

9 Load values into a `pmix_envar_t`

PMIx v2.0

C

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

C

11 IN `m`  
12 Pointer to the structure to be loaded (pointer to `pmix_envar_t`)  
13 IN `e`  
14 Environmental variable name (`char*`)  
15 IN `v`  
16 Value of variable (`char*`)  
17 IN `v`  
18 Separator character (`char`)

#### 19 3.2.26 Lookup Returned Data Structure

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

PMIx v1.0

C

21 `typedef struct pmix_pdata {`  
22     `pmix_proc_t proc;`  
23     `pmix_key_t key;`  
24     `pmix_value_t value;`  
25 } `pmix_pdata_t;`

C

#### 26 3.2.27 Lookup data structure support macros

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

### 3.2.27.1 Initialize the pdata structure

Initialize the `pmix_pdata_t` fields

*PMIx v1.0*

C

`PMIX_PDATA_CONSTRUCT(m)`

C

IN m

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

### 3.2.27.2 Destruct the pdata structure

Destruct the `pmix_pdata_t` fields

*PMIx v1.0*

C

`PMIX_PDATA_DESTRUCT(m)`

C

IN m

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

### 3.2.27.3 Create a pdata array

Allocate and initialize an array of `pmix_pdata_t` structures

*PMIx v1.0*

C

`PMIX_PDATA_CREATE(m, n)`

C

INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

### 3.2.27.4 Free a pdata structure

Release a `pmix_pdata_t` structure

*PMIx v4.0*

C

`PMIX_PDATA_RELEASE(m)`

C

IN m

Pointer to a `pmix_pdata_t` structure (handle)

### 3.2.27.5 Free a pdata array

Release an array of `pmix_pdata_t` structures

PMIx v1.0

C

`PMIX_PDATA_FREE (m, n)`

C

IN m

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

IN n

Number of structures in the array (`size_t`)

### 3.2.27.6 Load a lookup data structure

#### Summary

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

PMIx v1.0

C

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

C

IN m

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

IN p

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

IN k

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

IN d

Pointer to the data value to be loaded (handle)

IN t

Type of the provided data value (`pmix_data_type_t`)

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

#### Advice to users

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

### 3.2.27.7 Transfer a lookup data structure

#### Summary

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

PMIx v2.0

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

C

C

5 IN d

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

7 IN s

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

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

#### Advice to users

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

### 3.2.28 Application Structure

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

PMIx v1.0

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

C

C

## 3.2.29 App structure support macros

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

### 3.2.29.1 Initialize the app structure

Initialize the `pmix_app_t` fields

*PMIx v1.0*

C

`PMIX_APP_CONSTRUCT(m)`

C

IN `m`

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

### 3.2.29.2 Destruct the app structure

Destruct the `pmix_app_t` fields

*PMIx v1.0*

C

`PMIX_APP_DESTRUCT(m)`

C

IN `m`

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

### 3.2.29.3 Create an app array

Allocate and initialize an array of `pmix_app_t` structures

*PMIx v1.0*

C

`PMIX_APP_CREATE(m, n)`

C

INOUT `m`

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

IN `n`

Number of structures to be allocated (`size_t`)

### 3.2.29.4 Free an app structure

Release a `pmix_app_t` structure

*PMIx v4.0*

C

`PMIX_APP_RELEASE(m)`

C

IN `m`

Pointer to a `pmix_app_t` structure (handle)

### 3.2.29.5 Free an app array

Release an array of `pmix_app_t` structures

*PMIx v1.0*

C

`PMIX_APP_FREE (m, n)`

C

IN m

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

IN n

Number of structures in the array (`size_t`)

### 3.2.29.6 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)`

C

IN m

Pointer to the `pmix_app_t` structure (handle)

IN n

Number of directives to be allocated (`size_t`)

## 3.2.30 Query Structure

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

*PMIx v2.0*

C

```
19     typedef struct pmix_query {
20         char **keys;
21         pmix_info_t *qualifiers;
22         size_t nqual;
23     } pmix_query_t;
```

C

## 3.2.31 Query structure support macros

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

### 3.2.31.1 Initialize the query structure

2 Initialize the `pmix_query_t` fields

PMIx v2.0

C

3 `PMIX_QUERY_CONSTRUCT(m)`

C

4 **IN** `m`

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

### 3.2.31.2 Destruct the query structure

7 Destruct the `pmix_query_t` fields

PMIx v2.0

C

8 `PMIX_QUERY_DESTRUCT(m)`

C

9 **IN** `m`

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

### 3.2.31.3 Create a query array

12 Allocate and initialize an array of `pmix_query_t` structures

PMIx v2.0

C

13 `PMIX_QUERY_CREATE(m, n)`

C

**INOUT** `m`

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

16 **IN** `n`

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

### 3.2.31.4 Free a query structure

19 Release a `pmix_query_t` structure

PMIx v4.0

C

20 `PMIX_QUERY_RELEASE(m)`

C

**IN** `m`

21 Pointer to a `pmix_query_t` structure (handle)

### 3.2.31.5 Free a query array

Release an array of `pmix_query_t` structures

PMIx v2.0

C

3 `PMIX_QUERY_FREE (m, n)`

C

4 IN `m`

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

6 IN `n`

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

### 3.2.31.6 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.

PMIx v2.2

C

11 `PMIX_QUERY_QUALIFIERS_CREATE (m, n)`

C

12 IN `m`

13 Pointer to the `pmix_query_t` structure (handle)

14 IN `n`

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

## 3.2.32 Attribute registration structure

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

PMIx v4.0

C

```
18 typedef struct pmix_regattr {
19     char *name;
20     pmix_key_t *string;
21     pmix_data_type_t type;
22     pmix_info_t *info;
23     size_t ninfo;
24     char **description;
25 } pmix_regattr_t;
```

1 Note that in this structure:

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

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

#### Advice to PMIx library implementers

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

#### Advice to PMIx server hosts

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

### 24 3.2.33 Attribute registration structure support macros

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

### 3.2.33.1 Initialize the regattr structure

Initialize the `pmix_regattr_t` fields

PMIx v4.0

C

`PMIX_REGATTR_CONSTRUCT(m)`

C

IN `m`

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

### 3.2.33.2 Destruct the regattr structure

Destruct the `pmix_regattr_t` fields, releasing all strings.

PMIx v4.0

C

`PMIX_REGATTR_DESTRUCT(m)`

C

IN `m`

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

### 3.2.33.3 Create a regattr array

Allocate and initialize an array of `pmix_regattr_t` structures

PMIx v4.0

C

`PMIX_REGATTR_CREATE(m, n)`

C

INPUT `m`

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

IN `n`

Number of structures to be allocated (`size_t`)

### 3.2.33.4 Free a regattr array

Release an array of `pmix_regattr_t` structures

PMIx v4.0

C

`PMIX_REGATTR_FREE(m, n)`

C

INPUT `m`

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

IN `n`

Number of structures in the array (`size_t`)

### 3.2.33.5 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.

PMIx v4.0

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

C

C

IN `a`  
Pointer to the structure to be loaded (pointer to `pmix_proc_t`)  
IN `n`  
String name of the attribute (string)  
IN `k`  
Key value to be loaded (`pmix_key_t`)  
IN `t`  
Type of data associated with the provided key (`pmix_data_type_t`)  
IN `ni`  
Number of `pmix_info_t` elements to be allocated in `info` (`size_t`)  
IN `v`  
One-line description to be loaded (more can be added separately) (string)

### 3.2.33.6 Transfer a regattr to another regattr

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

PMIx v4.0

`PMIX_REGATTR_XFER(m, n)`

C

C

INOUT `m`  
Pointer to the destination `pmix_regattr_t` structure (handle)  
IN `m`  
Pointer to the source `pmix_regattr_t` structure (handle)

## 3.2.34 PMIx Group Directives

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

`PMIX_GROUP_DECLINE` Decline the invitation

`PMIX_GROUP_ACCEPT` Accept the invitation.

### 3.2.35 Byte Object Type

The `pmix_byte_object_t` structure describes a raw byte sequence.

PMIx v1.0

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

C

C

### 3.2.36 Byte object support macros

The following macros support the `pmix_byte_object_t` structure.

#### 3.2.36.1 Initialize the byte object structure

Initialize the `pmix_byte_object_t` fields

PMIx v2.0

```
PMIX_BYTE_OBJECT_CONSTRUCT(m)
```

C

C

IN m

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

#### 3.2.36.2 Destruct the byte object structure

Clear the `pmix_byte_object_t` fields

PMIx v2.0

```
PMIX_BYTE_OBJECT_DESTRUCT(m)
```

C

C

IN m

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

#### 3.2.36.3 Create a byte object structure

Allocate and intitialize an array of `pmix_byte_object_t` structures

PMIx v2.0

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

C

C

INOUT m

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

IN n

Number of structures to be allocated (`size_t`)

#### 1 3.2.36.4 Free a byte object array

2 Release an array of `pmix_byte_object_t` structures

PMIx v2.0

C

3 `PMIX_BYTE_OBJECT_FREE(m, n)`

C

4 IN `m`

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

6 IN `n`

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

#### 8 3.2.36.5 Load a byte object structure

9 Load values into a `pmix_byte_object_t`

PMIx v2.0

C

10 `PMIX_BYTE_OBJECT_LOAD(b, d, s)`

C

11 IN `b`

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

13 IN `d`

14 Pointer to the data to be loaded (`char*`)

15 IN `s`

16 Number of bytes in the data array (`size_t`)

#### 17 3.2.37 Data Array Structure

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

PMIx v2.0

C

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

C

#### 24 3.2.38 Data array support macros

25 The following macros support the `pmix_data_array_t` structure.

### 3.2.38.1 Initialize a data array structure

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

*PMIx v2.2*

C

`PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)`

C

IN m

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

IN n

Number of elements in the array (`size_t`)

IN t

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

### 3.2.38.2 Destruct a data array structure

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

*PMIx v2.2*

C

`PMIX_DATA_ARRAY_DESTRUCT(m)`

C

IN m

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

### 3.2.38.3 Create a data array structure

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

*PMIx v2.2*

C

`PMIX_DATA_ARRAY_CREATE(m, n, t)`

C

INOUT m

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

IN n

Number of elements in the array (`size_t`)

IN t

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

#### 1 3.2.38.4 Free a data array structure

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

PMIx v2.2

C

3 `PMIX_DATA_ARRAY_FREE (m)`

C

4 **IN m**

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

#### 6 3.2.39 Argument Array Macros

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

##### 9 3.2.39.1 Argument array extension

###### 10 Summary

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

C

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

C

13 **OUT r**

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

15 **INOUT a**

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

17 **IN b**

18 Argument to append to the list (string)

###### 19 Description

20 This function helps the caller build the **argv** portion of `pmix_app_t` structure, arrays of keys  
21 for querying, or other places where argv-style string arrays are required in the way that the **PRI!**  
22 expects it to be constructed.

###### Advice to users

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

## 3.2.39.2 Argument array prepend

### Summary

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

PMIX\_ARGV\_PREPEND(r, a, b);

#### OUT r

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

#### INOUT a

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

#### IN b

Argument to append to the list (string)

### Description

This function helps the caller build the **argv** portion of [pmix\\_app\\_t](#) structure, arrays of keys for querying, or other places where argv-style string arrays are required in the way that the **PRI!** expects it to be constructed.

### Advice to users

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

## 3.2.39.3 Argument array extension - unique

### Summary

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

PMIX\_ARGV\_APPEND\_UNIQUE(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)

1           **Description**

2     This function helps the caller build the **argv** portion of **pmix\_app\_t** structure, arrays of keys  
3     for querying, or other places where argv-style string arrays are required in the way that the **PRI!**  
4     expects it to be constructed.

5           **Advice to users**

6     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.

7     **3.2.39.4 Argument array release**

8           **Summary**

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

10    **PMIX\_ARGV\_FREE(a);**

11    **IN a**

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

13           **Description**

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

15     **3.2.39.5 Argument array split**

16           **Summary**

17     Split a string into a NULL-terminated argv array.

18    **PMIX\_ARGV\_SPLIT(a, b, c);**

19    **OUT a**

20       Resulting argv-style array (**char\*\***)

21    **IN b**

22       String to be split (**char\***)

23    **IN c**

24       Delimiter character (**char**)

1    **Description**

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

4    **Advice to users**

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

7    **3.2.39.6 Argument array join**

8    **Summary**

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

10    **C** `PMIX_ARGV_JOIN(a, b, c);`

11    **OUT a**  
12       Resulting string (`char*`)  
13    **IN b**  
14       Argv-style array to be joined (`char**`)  
15    **IN c**  
16       Delimiter character (`char`)

17    **Description**

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

19    **3.2.39.7 Argument array count**

20    **Summary**

21    Return the length of a NULL-terminated argv array.

22    **C** `PMIX_ARGV_COUNT(r, a);`

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

27    **Description**

28    Count the number of elements in an argv array

### 1 3.2.39.8 Argument array copy

#### 2 Summary

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

4 `PMIX_ARGV_COPY(a, b);`

5 **OUT a**

6 New argv-style array (`char**`)

7 **IN b**

8 Argv-style array (`char**`)

#### 9 Description

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

## 11 3.2.40 Set Environment Variable

#### 12 Summary

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

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

15 **OUT r**

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

17 **IN name**

18 Argument name (string)

19 **IN value**

20 Argument value (string)

21 **INOUT env**

22 Environment array to update (pointer to array of strings)

#### 23 Description

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

#### Advice to users

27 The provided name and value are copied into the destination environment array - thus, the source  
28 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.

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

```

1   PMIX_PERSIST    Persistance ( pmix\_persistence\_t )
2   PMIX_POINTER    Pointer to an object (void\*)
3   PMIX_SCOPE      Scope ( pmix\_scope\_t )
4   PMIX_DATA_RANGE Range for data ( pmix\_data\_range\_t )
5   PMIX_COMMAND    PMIx command code (used internally)
6   PMIX_INFO_DIRECTIVES Directives flag for pmix\_info\_t (
7     pmix\_info\_directives\_t )
8   PMIX_DATA_TYPE   Data type code ( pmix\_data\_type\_t )
9   PMIX_PROC_STATE  Process state ( pmix\_proc\_state\_t )
10  PMIX_PROC_INFO   Process information ( pmix\_proc\_info\_t )
11  PMIX_DATA_ARRAY  Data array ( pmix\_data\_array\_t )
12  PMIX_PROC_RANK   Process rank ( pmix\_rank\_t )
13  PMIX_QUERY       Query structure ( pmix\_query\_t )
14  PMIX_COMPRESSED_STRING String compressed with zlib (char\*)
15  PMIX_ALLOC_DIRECTIVE Allocation directive ( pmix\_alloc\_directive\_t )
16  PMIX_IOF_CHANNEL Input/output forwarding channel ( pmix\_iof\_channel\_t )
17  PMIX_ENVAR        Environmental variable structure ( pmix\_envar\_t )
18  PMIX_COORD        Structure containing fabric coordinates ( pmix\_coord\_t )
19  PMIX_REGATTR      Structure supporting attribute registrations ( pmix\_regattr\_t )
20  PMIX_REGEX        Regular expressions - can be a valid NULL-terminated string or an arbitrary
21    array of bytes
22  PMIX_JOB_STATE   Job state ( pmix\_job\_state\_t )
23  PMIX_LINK_STATE  Link state ( pmix\_link\_state\_t )
24  PMIX_DATA_TYPE_MAX A starting point for implementer-specific data types. Values above
25    this are guaranteed not to conflict with PMIx values. Definitions should always be based on
26    the PMIX\_DATA\_TYPE\_MAX constant and not a specific value as the value of the constant
27    may change.

```

## 3.4 Reserved attributes

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

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

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

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

4 **PMIX\_ATTR\_UNDEF NULL (NULL)**

5 Constant representing an undefined attribute.

### 6 3.4.1 Initialization attributes

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

9 **PMIX\_EVENT\_BASE "pmix.evbase" (struct event\_base \*)**

10 Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.

11 **PMIX\_SERVER\_TOOL\_SUPPORT "pmix.srvr.tool" (bool)**

12 The host RM wants to declare itself as willing to accept tool connection requests.

13 **PMIX\_SERVER\_REMOTE\_CONNECTIONS "pmix.srvr.remote" (bool)**

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

16 **PMIX\_SERVER\_SYSTEM\_SUPPORT "pmix.srvr.sys" (bool)**

17 The host RM wants to declare itself as being the local system server for PMIx connection  
18 requests.

19 **PMIX\_SERVER\_SESSION\_SUPPORT "pmix.srvr.sess" (bool)**

20 The host RM wants to declare itself as being the local session server for PMIx connection  
21 requests.

22 **PMIX\_SERVER\_START\_TIME "pmix.srvr.strtime" (char\*)**

23 Time when the server started - i.e., when the server created it's rendezvous file (given in  
24 ctime string format)

25 **PMIX\_SERVER\_TMPDIR "pmix.srvr.tmpdir" (char\*)**

26 Top-level temporary directory for all client processes connected to this server, and where the  
27 PMIx server will place its tool rendezvous point and contact information.

28 **PMIX\_SYSTEM\_TMPDIR "pmix.sys.tmpdir" (char\*)**

29 Temporary directory for this system, and where a PMIx server that declares itself to be a  
30 system-level server will place a tool rendezvous point and contact information.

31 **PMIX\_SERVER\_ENABLE\_MONITORING "pmix.srv.monitor" (bool)**

32 Enable PMIx internal monitoring by the PMIx server.

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

34 Name of the namespace to use for this PMIx server.

35 **PMIX\_SERVER\_RANK "pmix.srv.rank" (pmix\_rank\_t)**

36 Rank of this PMIx server

37 **PMIX\_SERVER\_GATEWAY "pmix.srv.gway" (bool)**

38 Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes  
39 (e.g., logging to email)

---

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

## 1 3.4.2 Identification attributes

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

```
4 PMIX_USERID "pmix.euid" (uint32_t)
5   Effective user id.
6 PMIX_GRP_ID "pmix.egid" (uint32_t)
7   Effective group id.
8 PMIX_DSTPATH "pmix.dstpath" (char*)
9   Path to shared memory data storage (dstore) files.
10 PMIX_VERSION_INFO "pmix.version" (char*)
11   PMIx version of contractor.
12 PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)
13   The requesting process is a PMIx tool.
14 PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)
15   The requesting process is a PMIx client.
16 PMIX_PSET_NAME "pmix.pset.nm" (char*)
17   User-assigned name for the process set containing the given process.
18 PMIX_REINCARNATION "pmix.reinc" (uint32_t)
19   Number of times this process has been re-instantiated - i.e, a value of zero indicates that the
20   process has never failed and been restarted.
```

## 21 3.4.3 Programming model attributes

22 These attributes are associated with programming models.

```
23 PMIX_PROGRAMMING_MODEL "pmix.pgm.model" (char*)
24   Programming model being initialized (e.g., "MPI" or "OpenMP")
25 PMIX_MODEL_LIBRARY_NAME "pmix.mdl.name" (char*)
26   Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
27 PMIX_MODEL_LIBRARY_VERSION "pmix.mld.vrs" (char*)
28   Programming model version string (e.g., "2.1.1")
29 PMIX_THREADING_MODEL "pmix.threads" (char*)
30   Threading model used (e.g., "pthreads")
31 PMIX_MODEL_NUM_THREADS "pmix.mdl.nthrds" (uint64_t)
32   Number of active threads being used by the model
33 PMIX_MODEL_NUM_CPUS "pmix.mdl.ncpu" (uint64_t)
34   Number of cpus being used by the model
35 PMIX_MODEL_CPU_TYPE "pmix.mdl.cputype" (char*)
36   Granularity - "hwthread", "core", etc.
37 PMIX_MODEL_PHASE_NAME "pmix.mdl.phase" (char*)
38   User-assigned name for a phase in the application execution (e.g., "cfd reduction")
39 PMIX_MODEL_PHASE_TYPE "pmix.mdl.ptype" (char*)
40   Type of phase being executed (e.g., "matrix multiply")
```

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

#### 4 3.4.4 UNIX socket rendezvous socket attributes

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

```
8 PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)
9     Disable legacy UNIX socket (usock) support
10 PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)
11     POSIX mode_t (9 bits valid)
12 PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)
13     Use only one rendezvous socket, letting priorities and/or environment parameters select the
14     active transport.
```

#### 15 3.4.5 TCP connection attributes

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

```
19 PMIX_TCP_REPORT_URI "pmix.tcp.repuri" (char*)
20     If provided, directs that the TCP uniform resource identifier (URI) be reported and indicates
21     the desired method of reporting: '-' for stdout, '+' for stderr, or filename.
22 PMIX_TCP_URI "pmix.tcp.uri" (char*)
23     The URI of the PMIx server to connect to, or a file name containing it in the form of
24     file:<name of file containing it>.
25 PMIX_TCP_IF_INCLUDE "pmix.tcp.ifinclude" (char*)
26     Comma-delimited list of devices and/or Classless Inter-Domain Routing (CIDR) notation to
27     include when establishing the TCP connection.
28 PMIX_TCP_IF_EXCLUDE "pmix.tcp.ifexclude" (char*)
29     Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
30     TCP connection.
31 PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)
32     The IPv4 port to be used.
33 PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)
34     The IPv6 port to be used.
35 PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)
36     Set to true to disable IPv4 family of addresses.
37 PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)
38     Set to true to disable IPv6 family of addresses.
```

## 1 3.4.6 Global Data Storage (GDS) attributes

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

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

## 7 3.4.7 General process-level attributes

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

9 `PMIX_CPUSET "pmix.cpuset" (char*)`  
10 HWLOC<sup>2</sup> bitmap applied to the process upon launch.  
11 `PMIX_CREDENTIAL "pmix.cred" (char*)`  
12 Security credential assigned to the process.  
13 `PMIX_SPAWNED "pmix.spawned" (bool)`  
14 `true` if this process resulted from a call to `PMIx_Spawn`. Lack of inclusion (i.e., a return  
15 status of `PMIX_ERR_NOT_FOUND`) corresponds to a value of `false` for this attribute.  
16 `PMIX_ARCH "pmix.arch" (uint32_t)`  
17 Architecture flag.

## 18 3.4.8 Scratch directory attributes

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

21 `PMIX_TMPDIR "pmix.tmpdir" (char*)`  
22 Full path to the top-level temporary directory assigned to the session.  
23 `PMIX_NSDIR "pmix.nsdir" (char*)`  
24 Full path to the temporary directory assigned to the namespace, under `PMIX_TMPDIR`.  
25 `PMIX_PROCDIR "pmix.pdir" (char*)`  
26 Full path to the subdirectory under `PMIX_NSDIR` assigned to the process.  
27 `PMIX_TDIR_RMCLEAN "pmix.tdir.rmclean" (bool)`  
28 Resource Manager will clean session directories

---

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

## 3.4.9 Relative Rank Descriptive Attributes

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

**PMIX\_CLUSTER\_ID** "pmix.clid" (`char*`)  
A string name for the cluster this allocation is on

**PMIX\_PROCID** "pmix.procid" (`pmix_proc_t`)  
Process identifier

**PMIX\_NSPACE** "pmix.nspace" (`char*`)  
Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system.

**PMIX\_JOBID** "pmix.jobid" (`char*`)  
Job identifier assigned by the scheduler - may be identical to the namespace, but is often a numerical value expressed as a string (e.g., "12345.3").

**PMIX\_APPNUM** "pmix.appnum" (`uint32_t`)  
Application number within the job.

**PMIX\_RANK** "pmix.rank" (`pmix_rank_t`)  
Process rank within the job.

**PMIX\_GLOBAL\_RANK** "pmix.grank" (`pmix_rank_t`)  
Process rank spanning across all jobs in this session.

**PMIX\_APP\_RANK** "pmix.apprank" (`pmix_rank_t`)  
Process rank within this application.

**PMIX\_NPROC\_OFFSET** "pmix.offset" (`pmix_rank_t`)  
Starting global rank of this job - referenced using **PMIX\_RANK\_WILDCARD**.

**PMIX\_LOCAL\_RANK** "pmix.lrank" (`uint16_t`)  
Local rank on this node within this job.

**PMIX\_NODE\_RANK** "pmix.nrank" (`uint16_t`)  
Process rank on this node spanning all jobs.

**PMIX\_PACKAGE\_RANK** "pmix.pkgrank" (`uint16_t`)  
Process rank within this job on the package where this process resides

**PMIX\_LOCALLDR** "pmix.lldr" (`pmix_rank_t`)  
Lowest rank on this node within this job - referenced using **PMIX\_RANK\_WILDCARD**.

**PMIX\_APPLDR** "pmix.aldr" (`pmix_rank_t`)  
Lowest rank in this application within this job - referenced using **PMIX\_RANK\_WILDCARD**.

**PMIX\_PROC\_PID** "pmix.ppid" (`pid_t`)  
PID of specified process.

**PMIX\_SESSION\_ID** "pmix.session.id" (`uint32_t`)  
Session identifier assigned by the scheduler - referenced using **PMIX\_RANK\_WILDCARD**.

**PMIX\_NODE\_LIST** "pmix.nlist" (`char*`)  
Comma-delimited list of nodes running processes for the specified namespace - referenced using **PMIX\_RANK\_WILDCARD**.

**PMIX\_ALLOCATED\_NODELIST** "pmix.alist" (`char*`)

```

1      Comma-delimited list or regular expression of all nodes in this allocation regardless of
2      whether or not they currently host processes - referenced using PMIX_RANK_WILDCARD .
3      PMIX_HOSTNAME "pmix.hname" (char*)
4          Name of the host (e.g., where a specified process is running, or a given device is located).
5      PMIX_HOSTNAME_ALIASES "pmix.alias" (char*)
6          Comma-delimited list of names by which this node is known.
7      PMIX_HOSTNAME_KEEP_FQDN "pmix.fqdn" (bool)
8          FQDN hostnames are being retained
9      PMIX_NODEID "pmix.nodeid" (uint32_t)
10     Node identifier expressed as the node's index (beginning at zero) in an array of nodes within
11     the active session. The value must be unique and directly correlate to the PMIX_HOSTNAME
12     of the node - i.e., users can interchangeably reference the same location using either the
13     PMIX_HOSTNAME or corresponding PMIX_NODEID .
14      PMIX_LOCAL_PEERS "pmix.lpeers" (char*)
15          Comma-delimited list of ranks on this node within the specified namespace - referenced
16          using PMIX_RANK_WILDCARD .
17      PMIX_LOCAL_PROCS "pmix.lprocs" (pmix_proc_t array)
18          Array of pmix_proc_t of all processes on the specified node - referenced using
19          PMIX_RANK_WILDCARD .
20      PMIX_LOCAL_CPUSETS "pmix.lcpus" (char*)
21          Colon-delimited cpusets of local peers within the specified namespace - referenced using
22          PMIX_RANK_WILDCARD .
23      PMIX_PROC_URI "pmix.puri" (char*)
24          URI containing contact information for a given process.
25      PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)
26          Process identifier of the parent process of the calling process.
27      PMIX_EXIT_CODE "pmix.exit.code" (int)
28          Exit code returned when process terminated

```

### 3.4.10 Information retrieval attributes

The following attributes are used to specify the level of information (e.g., **session**, **job**, or **application**) being requested where ambiguity may exist - see [5.3.5](#) for examples of their use.

```

32      PMIX_SESSION_INFO "pmix.ssn.info" (bool)
33          Return information about the specified session. If information about a session other than the
34          one containing the requesting process is desired, then the attribute array must contain a
35          PMIX_SESSION_ID attribute identifying the desired target.
36      PMIX_JOB_INFO "pmix.job.info" (bool)
37          Return information about the specified job or namespace. If information about a job or
38          namespace other than the one containing the requesting process is desired, then the attribute
39          array must contain a PMIX_JOBID or PMIX_NSPACE attribute identifying the desired
40          target. Similarly, if information is requested about a job or namespace in a session other than
41          the one containing the requesting process, then an attribute identifying the target session
42          must be provided.

```

```

1   PMIX_APP_INFO "pmix.app.info" (bool)
2       Return information about the specified application. If information about an application other
3       than the one containing the requesting process is desired, then the attribute array must
4       contain a PMIX_APPNUM attribute identifying the desired target. Similarly, if information
5       is requested about an application in a job or session other than the one containing the
6       requesting process, then attributes identifying the target job and/or session must be provided.
7   PMIX_NODE_INFO "pmix.node.info" (bool)
8       Return information about the specified node. If information about a node other than the one
9       containing the requesting process is desired, then the attribute array must contain either the
10      PMIX_NODEID or PMIX_HOSTNAME attribute identifying the desired target.

```

### 3.4.11 Information storage attributes

The following attributes are used to assemble information by its level (e.g., **session**, **job**, or **application**) for storage where ambiguity may exist - see 11.3.3.1 for examples of their use.

```

14  PMIX_SESSION_INFO_ARRAY "pmix.ssn.arr" (pmix_data_array_t)
15      Provide an array of pmix_info_t containing session-level information. The
16      PMIX_SESSION_ID attribute is required to be included in the array.
17  PMIX_JOB_INFO_ARRAY "pmix.job.arr" (pmix_data_array_t)
18      Provide an array of pmix_info_t containing job-level information. The
19      PMIX_SESSION_ID attribute of the session containing the job is required to be
20      included in the array whenever the PMIx server library may host multiple sessions (e.g.,
21      when executing with a host RM daemon). As information is registered one job (aka
22      namespace) at a time via the PMIx_server_register_nspace API, there is no
23      requirement that the array contain either the PMIX_NSPACE or PMIX_JOBID attributes
24      when used in that context (though either or both of them may be included). At least one of
25      the job identifiers must be provided in all other contexts where the job being referenced is
26      ambiguous.
27  PMIX_APP_INFO_ARRAY "pmix.app.arr" (pmix_data_array_t)
28      Provide an array of pmix_info_t containing app-level information. The PMIX_NSPACE
29      or PMIX_JOBID attributes of the job containing the application, plus its PMIX_APPNUM
30      attribute, must to be included in the array when the array is not included as part of a call to
31      PMIx_server_register_nspace - i.e., when the job containing the application is
32      ambiguous. The job identification is otherwise optional.
33  PMIX_PROC_INFO_ARRAY "pmix.pdata" (pmix_data_array_t)
34      Provide an array of pmix_info_t containing process-level information. The
35      PMIX_RANK and PMIX_NSPACE attributes, or the PMIX_PROCID attribute, are
36      required to be included in the array when the array is not included as part of a call to
37      PMIx_server_register_nspace - i.e., when the job containing the process is
38      ambiguous. All three may be included if desired. When the array is included in some
39      broader structure that identifies the job, then only the PMIX_RANK or the PMIX_PROCID
40      attribute must be included (the others are optional).
41  PMIX_NODE_INFO_ARRAY "pmix.node.arr" (pmix_data_array_t)

```

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

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

5           Array of data on a given server, starting with its `PMIX_NSPACE`

6       Note that these assemblages can be used hierarchically:

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

### Advice to PMIx library implementers

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

## 3.4.12 Size information attributes

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

21       `PMIX_UNIV_SIZE "pmix.univ.size" (uint32_t)`

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

26       `PMIX_JOB_SIZE "pmix.job.size" (uint32_t)`

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

32       `PMIX_JOB_NUM_APPS "pmix.job.napps" (uint32_t)`

33           Number of applications in this job.

34       `PMIX_APP_SIZE "pmix.app.size" (uint32_t)`

35           Number of processes in this application.

36       `PMIX_LOCAL_SIZE "pmix.local.size" (uint32_t)`

```
1     Number of processes in this job or application on this node.  
2 PMIX_NODE_SIZE "pmix.node.size" (uint32_t)  
3     Number of processes across all jobs on this node.  
4 PMIX_MAX_PROCS "pmix.max.size" (uint32_t)  
5     Maximum number of processes that can be executed in this context (session, namespace,  
6     application, or node). Typically, this is a constraint imposed by a scheduler or by user  
7     settings in a hostfile or other resource description.  
8 PMIX_NUM_SLOTS "pmix.num.slots" (uint32_t)  
9     Number of slots allocated in this context (session, namespace, application, or node). Note  
10    that this attribute is the equivalent to PMIX_MAX_PROCS used in the corresponding  
11    context - it is included in the Standard for historical reasons.  
12 PMIX_NUM_NODES "pmix.num.nodes" (uint32_t)  
13    Number of nodes in this session, or that are currently executing processes from the  
14    associated namespace or application.
```

### 15 **3.4.13 Memory information attributes**

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

```
18 PMIX_AVAIL_PHYS_MEMORY "pmix.pmem" (uint64_t)  
19     Total available physical memory on this node.  
20 PMIX_DAEMON_MEMORY "pmix.dmn.mem" (float)  
21     Megabytes of memory currently used by the RM daemon.  
22 PMIX_CLIENT_AVG_MEMORY "pmix.cl.mem.avg" (float)  
23     Average Megabytes of memory used by client processes.
```

### 24 **3.4.14 Topology information attributes**

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

---

#### Advice to users

---

```
27 The following attributes are being removed from the PMIx Standard as they are internal to a given  
28 PMIx implementation. Users are referred to the PMIX_Load_topology API for obtaining the  
29 local topology description.
```

```

1  PMIX_LOCAL_TOPO "pmix.ltopo" (char*)
2      eXtensible Markup Language (XML) representation of local node topology.
3  PMIX_TOPOLOGY_XML "pmix.topo.xml" (char*)
4      XML-based description of topology
5  PMIX_TOPOLOGY_FILE "pmix.topo.file" (char*)
6      Full path to file containing XML topology description
7  PMIX_TOPOLOGY_SIGNATURE "pmix.toposig" (char*)
8      Topology signature string.
9  PMIX_HWLOC_SHMEM_ADDR "pmix.hwlocaddr" (size_t)
10     Address of the HWLOC shared memory segment.
11  PMIX_HWLOC_SHMEM_SIZE "pmix.hwlocsize" (size_t)
12     Size of the HWLOC shared memory segment.
13  PMIX_HWLOC_SHMEM_FILE "pmix.hwlocfile" (char*)
14     Path to the HWLOC shared memory file.
15  PMIX_HWLOC_XML_V1 "pmix.hwlocxml1" (char*)
16     XML representation of local topology using HWLOC's v1.x format.
17  PMIX_HWLOC_XML_V2 "pmix.hwlocxml2" (char*)
18     XML representation of local topology using HWLOC's v2.x format.
19  PMIX_HWLOC_SHARE_TOPO "pmix.hwlocsh" (bool)
20     Share the HWLOC topology via shared memory
21  PMIX_HWLOC_HOLE_KIND "pmix.hwlocholek" (char*)
22     Kind of VM "hole" HWLOC should use for shared memory

```

### 3.4.15 Request-related attributes

These attributes are used to influence the behavior of various PMIx operations - they do not represent values accessed using the [PMIx\\_Get](#) API.

```

26  PMIX_COLLECT_DATA "pmix.collect" (bool)
27      Collect data and return it at the end of the operation.
28  PMIX_TIMEOUT "pmix.timeout" (int)
29      Time in seconds before the specified operation should time out (0 indicating infinite) in
30      error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
31      the target process from ever exposing its data.
32  PMIX_IMMEDIATE "pmix.immediate" (bool)
33      Specified operation should immediately return an error from the PMIx server if the requested
34      data cannot be found - do not request it from the host RM.
35  PMIX_WAIT "pmix.wait" (int)
36      Caller requests that the PMIx server wait until at least the specified number of values are
37      found (0 indicates all and is the default).
38  PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

```

1 Comma-delimited list of algorithms to use for the collective operation. PMIx does not  
 2 impose any requirements on a host environment's collective algorithms. Thus, the  
 3 acceptable values for this attribute will be environment-dependent - users are encouraged to  
 4 check their host environment for supported values.  
 5 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)  
 6 If **true**, indicates that the requested choice of algorithm is mandatory.  
 7 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)  
 8 Value for calls to publish/lookup/unpublish or for monitoring event notifications.  
 9 **PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)  
 10 Value for calls to **PMIx\_Publish**.  
 11 **PMIX\_DATA\_SCOPE** "pmix.scope" (pmix\_scope\_t)  
 12 Scope of the data to be found in a **PMIx\_Get** call.  
 13 **PMIX\_OPTIONAL** "pmix.optional" (bool)  
 14 Look only in the client's local data store for the requested value - do not request data from  
 15 the PMIx server if not found.  
 16 **PMIX\_GET\_STATIC\_VALUES** "pmix.get.static" (bool)  
 17 Request that any pointers in the returned value point directly to values in the key-value store  
 18 **PMIX\_EMBED\_BARRIER** "pmix.embed.barrier" (bool)  
 19 Execute a blocking fence operation before executing the specified operation. For example,  
 20 **PMIx\_Finalize** does not include an internal barrier operation by default. This attribute  
 21 would direct **PMIx\_Finalize** to execute a barrier as part of the finalize operation.  
 22 **PMIX\_JOB\_TERM\_STATUS** "pmix.job.term.status" (pmix\_status\_t)  
 23 Status returned by job upon its termination. The status will be communicated as part of a  
 24 PMIx event payload provided by the host environment upon termination of a job. Note that  
 25 generation of the **PMIX\_EVENT\_JOB\_END** event is optional and host environments may  
 26 choose to provide it only upon request.  
 27 **PMIX\_PROC\_STATE\_STATUS** "pmix.proc.state" (pmix\_proc\_state\_t)  
 28 State of the specified process as of the last report - may not be the actual current state based  
 29 on update rate.  
 30 **PMIX\_PROC\_TERM\_STATUS** "pmix.proc.term.status" (pmix\_status\_t)  
 31 Status returned by a process upon its termination. The status will be communicated as part  
 32 of a PMIx event payload provided by the host environment upon termination of a process.  
 33 Note that generation of the **PMIX\_PROC\_TERMINATED** event is optional and host  
 34 environments may choose to provide it only upon request.  
 35 **PMIX\_NOTIFY\_LAUNCH** "pmix.note.lnch" (bool)  
 36 Notify the requester upon launch of the child job and return its namespace in the event  
 37 **PMIX\_GET\_REFRESH\_CACHE** "pmix.get.refresh" (bool)  
 38 When retrieving data for a remote process, refresh the existing local data cache for the  
 39 process in case new values have been put and committed by it since the last refresh

## 1 3.4.16 Server-to-PMIx library attributes

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

5 **PMIX\_REGISTER\_NODATA** "pmix.reg.nodata" (bool)  
6 Registration is for this namespace only, do not copy job data - this attribute is not accessed  
7 using the **PMIx\_Get**  
8 **PMIX\_PROC\_DATA** "pmix\_pdata" (pmix\_data\_array\_t)  
9 Deprecated in favor of the **PMIX\_PROC\_INFO\_ARRAY** attribute  
10 **PMIX\_NODE\_MAP** "pmix.nmap" (char\*)  
11 Regular expression of nodes - see [11.3.3.1](#) for an explanation of its generation.  
12 **PMIX\_PROC\_MAP** "pmix.pmap" (char\*)  
13 Regular expression describing processes on each node - see [11.3.3.1](#) for an explanation of its  
14 generation.  
15 **PMIX\_ANL\_MAP** "pmix.anlmap" (char\*)  
16 Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.  
17 **PMIX\_APP\_MAP\_TYPE** "pmix.apmap.type" (char\*)  
18 Type of mapping used to layout the application (e.g., **cyclic**).  
19 **PMIX\_APP\_MAP\_REGEX** "pmix.apmap.regex" (char\*)  
20 Regular expression describing the result of the process mapping.  
21 **PMIX\_REQUIRED\_KEY** "pmix.req.key" (char\*)  
22 Key the user needs prior to responding from a dmmodex request

## 23 3.4.17 Server-to-Client attributes

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

26 **PMIX\_PROC\_BLOB** "pmix.pblob" (pmix\_byte\_object\_t)  
27 Packed blob of process data.  
28 **PMIX\_MAP\_BLOB** "pmix.mblob" (pmix\_byte\_object\_t)  
29 Packed blob of process location.

## 30 3.4.18 Event handler registration and notification attributes

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

33 **PMIX\_EVENT\_HDLR\_NAME** "pmix.evname" (char\*)  
34 String name identifying this handler.  
35 **PMIX\_EVENT\_HDLR\_FIRST** "pmix.evfirrst" (bool)  
36 Invoke this event handler before any other handlers.  
37 **PMIX\_EVENT\_HDLR\_LAST** "pmix.evlast" (bool)  
38 Invoke this event handler after all other handlers have been called.

```

1   PMIX_EVENT_HDLR_FIRST_IN_CATEGORY "pmix.evfirscat" (bool)
2       Invoke this event handler before any other handlers in this category.
3   PMIX_EVENT_HDLR_LAST_IN_CATEGORY "pmix.evlastcat" (bool)
4       Invoke this event handler after all other handlers in this category have been called.
5   PMIX_EVENT_HDLR_BEFORE "pmix.evbefore" (char*)
6       Put this event handler immediately before the one specified in the (char*) value.
7   PMIX_EVENT_HDLR_AFTER "pmix.evafter" (char*)
8       Put this event handler immediately after the one specified in the (char*) value.
9   PMIX_EVENT_HDLR_PREPEND "pmix.evprepend" (bool)
10      Prepend this handler to the precedence list within its category.
11   PMIX_EVENT_HDLR_APPEND "pmix.evappend" (bool)
12      Append this handler to the precedence list within its category.
13   PMIX_EVENT_CUSTOM_RANGE "pmix.evrangle" (pmix_data_array_t*)
14      Array of pmix\_proc\_t defining range of event notification.
15   PMIX_EVENT_AFFECTED_PROC "pmix.evproc" (pmix_proc_t)
16      The single process that was affected.
17   PMIX_EVENT_AFFECTED_PROCS "pmix.evaaffected" (pmix_data_array_t*)
18      Array of pmix\_proc\_t defining affected processes.
19   PMIX_EVENT_NON_DEFAULT "pmix.evnonddef" (bool)
20      Event is not to be delivered to default event handlers.
21   PMIX_EVENT_RETURN_OBJECT "pmix.evobject" (void *)
22      Object to be returned whenever the registered callback function cbfunc is invoked. The
23      object will only be returned to the process that registered it.
24   PMIX_EVENT_DO_NOT_CACHE "pmix.evnocache" (bool)
25      Instruct the PMIx server not to cache the event.
26   PMIX_EVENT_SILENT_TERMINATION "pmix.evsilentterm" (bool)
27      Do not generate an event when this job normally terminates.
28   PMIX_EVENT_PROXY "pmix.evproxy" (pmix_proc_t*)
29      PMIx server that sourced the event
30   PMIX_EVENT_TEXT_MESSAGE "pmix.evttext" (char*)
31      Text message suitable for output by recipient - e.g., describing the cause of the event
32   PMIX_EVENT_TIMESTAMP "pmix.evtstamp" (time_t)
33      System time when the associated event occurred.

```

### 34 3.4.19 Fault tolerance attributes

Attributes to support fault tolerance behaviors - they are values passed to the event notification API and therefore are not accessed using the [PMIx\\_Get](#) API.

```

37   PMIX_EVENT_TERMINATE_SESSION "pmix.evterm.sess" (bool)
38      The RM intends to terminate this session.
39   PMIX_EVENT_TERMINATE_JOB "pmix.evterm.job" (bool)
40      The RM intends to terminate this job.
41   PMIX_EVENT_TERMINATE_NODE "pmix.evterm.node" (bool)

```

```

1      The RM intends to terminate all processes on this node.
2      PMIX_EVENT_TERMINATE_PROC "pmix.evtterm.proc" (bool)
3          The RM intends to terminate just this process.
4      PMIX_EVENT_ACTION_TIMEOUT "pmix.evttimeout" (int)
5          The time in seconds before the RM will execute error response.
6      PMIX_EVENT_NO_TERMINATION "pmix.evnoterm" (bool)
7          Indicates that the handler has satisfactorily handled the event and believes termination of the
8          application is not required.
9      PMIX_EVENT_WANT_TERMINATION "pmix.evtterm" (bool)
10         Indicates that the handler has determined that the application should be terminated

```

### 11 3.4.20 Spawn attributes

```

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

```

```

19 PMIX_PERSONALITY "pmix.pers" (char*)
20     Name of personality to use.
21 PMIX_HOST "pmix.host" (char*)
22     Comma-delimited list of hosts to use for spawned processes.
23 PMIX_HOSTFILE "pmix.hostfile" (char*)
24     Hostfile to use for spawned processes.
25 PMIX_ADD_HOST "pmix.addhost" (char*)
26     Comma-delimited list of hosts to add to the allocation.
27 PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)
28     Hostfile listing hosts to add to existing allocation.
29 PMIX_PREFIX "pmix.prefix" (char*)
30     Prefix to use for starting spawned processes.
31 PMIX_WDIR "pmix.wdir" (char*)
32     Working directory for spawned processes.
33 PMIX_MAPPER "pmix.mapper" (char*)
34     Mapping mechanism to use for placing spawned processes - when accessed using
35     PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping
36     mechanism used for the provided namespace.
37 PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
38     Display process mapping upon spawn.
39 PMIX_PPR "pmix.ppr" (char*)
40     Number of processes to spawn on each identified resource.
41 PMIX_MAPBY "pmix.mapby" (char*)

```

```

1 Process mapping policy - when accessed using PMIx_Get , use the
2 PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
3 provided namespace
4 PMIX_RANKBY "pmix.rankby" (char*)
5 Process ranking policy - when accessed using PMIx_Get , use the
6 PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the
7 provided namespace
8 PMIX_BINDTO "pmix.bindto" (char*)
9 Process binding policy - when accessed using PMIx_Get , use the
10 PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the
11 provided namespace
12 PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
13 Preload binaries onto nodes.
14 PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
15 Comma-delimited list of files to pre-position on nodes.
16 PMIX_NON_PMI "pmix.nonpmi" (bool)
17 Spawning processes will not call PMIx_Init .
18 PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
19 Spawning process rank that is to receive any forwarded stdin.
20 PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
21 Set the application's current working directory to the session working directory assigned by
22 the RM - when accessed using PMIx_Get , use the PMIX_RANK_WILDCARD value for
23 the rank to discover the session working directory assigned to the provided namespace
24 PMIX_TAG_OUTPUT "pmix.tagout" (bool)
25 Tag application output with the identity of the source process.
26 PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
27 Timestamp output from applications.
28 PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
29 Merge stdout and stderr streams from application processes.
30 PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
31 Direct application output (both stdout and stderr) into files of form "<filename>.rank"
32 PMIX_OUTPUT_TO_DIRECTORY "pmix.outdir" (char*)
33 Direct application output into files of form "<directory>/<jobid>/rank.<rank>/stdout[err]"
34 PMIX_INDEX_ARGV "pmix.indxargv" (bool)
35 Mark the argv with the rank of the process.
36 PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
37 Number of cpus to assign to each rank - when accessed using PMIx_Get , use the
38 PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the
39 provided namespace
40 PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
41 Do not place processes on the head node.
42 PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
43 Do not oversubscribe the cpus.

```

```

1  PMIX_REPORT_BINDINGS "pmix.repbinding" (bool)
2      Report bindings of the individual processes.
3  PMIX_CPU_LIST "pmix.cpulist" (char*)
4      List of cpus to use for this job - when accessed using PMIx_Get , use the
5          PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided
6          namespace
7  PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
8      Application supports recoverable operations.
9  PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
10     Application is continuous, all failed processes should be immediately restarted.
11  PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
12      Maximum number of times to restart a job - when accessed using PMIx_Get , use the
13          PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
14          namespace
15  PMIX_SPAWN_TOOL "pmix.spwn.tool" (bool)
16      Indicate that the job being spawned is a tool
17  PMIX_CMD_LINE "pmix.cmd.line" (char*)
18      Command line used to execute the specified process (e.g., "mpirun -n 2 --map-by foo
19          ./myapp")
20  PMIX_FORK_EXEC_AGENT "pmix.fe.agnt" (char*)
21      Command line of fork/exec agent to be used for starting local processes
22  PMIX_TIMEOUT_STACKTRACES "pmix.tim.stack" (bool)
23      Include process stacktraces in timeout report from a job
24  PMIX_TIMEOUT_REPORT_STATE "pmix.tim.state" (bool)
25      Report process states in timeout report from a job
26  PMIX_APP_ARGV "pmix.app.argv" (char*)
27      Consolidated argv passed to the spawn command for the given process (e.g., "./myapp arg1
28          arg2 arg3")
29  PMIX_NOTIFY_JOB_EVENTS "pmix.note.jev" (bool)
30      Requests that the PMIx server provide the requester with the PMIX_EVENT_JOB_START ,
31          PMIX_LAUNCH_COMPLETE , and PMIX_EVENT_JOB_END events. Each event is to
32          include at least the namespace of the corresponding job and a PMIX_EVENT_TIMESTAMP
33          indicating the time the event occurred. Note that these events can be captured and processed
34          by registering a default event handler instead of individual handlers and then processing the
35          events based on the returned status code. Another common method is to register one event
36          handler for all job-related events, with a separate handler for non-job events - see
37          PMIx_Register_event_handler for details.
38  PMIX_NOTIFY_COMPLETION "pmix.notecomp" (bool)
39      Requests that the PMIx generate the PMIX_EVENT_JOB_END event for normal or
40          abnormal termination of the spawned job. The event shall include the returned status code
41          for the corresponding job and a PMIX_EVENT_TIMESTAMP indicating the time the
42          termination occurred.
43  PMIX_LOG_JOB_EVENTS "pmix.log.jev" (bool)

```

1 Requests that the PMIx server log the **PMIX\_EVENT\_JOB\_START**,  
2 **PMIX\_LAUNCH\_COMPLETE**, and **PMIX\_EVENT\_JOB\_END** events using **PMIx\_Log**  
3 instead of generating PMIx events, subject to the logging attributes of Section 3.4.22

#### 4 3.4.21 Query attributes

5 Attributes used to describe **PMIX\_Query\_info\_nb** behavior - these are values passed to the  
6 **PMIx\_Query\_info\_nb** API and therefore are not passed to the **PMIx\_Get** API.

7 **PMIX\_QUERY\_SUPPORTED\_KEYS** "pmixqry.keys" (char\*)  
8 Returns comma-delimited list of keys supported by the query function. NO QUALIFIERS  
9 **PMIX\_QUERY\_SUPPORTED\_QUALIFIERS** "pmixqry.quals" (char\*)  
10 Return comma-delimited list of qualifiers supported by a query on the provided key, instead  
11 of actually performing the query on the key. NO QUALIFIERS  
12 **PMIX\_QUERY\_REFRESH\_CACHE** "pmixqry.rfsh" (bool)  
13 Retrieve updated information from server. NO QUALIFIERS  
14 **PMIX\_QUERY\_NAMESPACES** "pmixqry.ns" (char\*)  
15 Request a comma-delimited list of active namespaces. NO QUALIFIERS  
16 **PMIX\_QUERY\_NAMESPACE\_INFO** "pmixqry.nsinfo" (pmix\_data\_array\_t\*)  
17 Return an array of active namespace information - each element will itself contain an array  
18 including the namespace plus the command line of the application executing within it.  
19 SUPPORTED QUALIFIERS: **PMIX\_NSPACE** of specific namespace whose info is being  
20 requested  
21 **PMIX\_QUERY\_JOB\_STATUS** "pmixqry.jst" (pmix\_status\_t)  
22 Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX\_NSPACE**  
23 indicating the namespace whose status is being queried  
24 **PMIX\_QUERY\_QUEUE\_LIST** "pmixqry.qlst" (char\*)  
25 Request a comma-delimited list of scheduler queues. NO QUALIFIERS  
26 **PMIX\_QUERY\_QUEUE\_STATUS** "pmixqry.qst" (char\*)  
27 Returns status of a specified scheduler queue, expressed as a string. SUPPORTED  
28 QUALIFIERS: **PMIX\_ALLOC\_QUEUE** naming specific queue whose status is being  
29 requested  
30 **PMIX\_QUERY\_PROC\_TABLE** "pmixqry.ptable" (char\*)  
31 Returns a (pmix\_data\_array\_t) array of **pmix\_proc\_info\_t**, one entry for each  
32 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
33 **PMIX\_NSPACE** indicating the namespace whose process table is being queried  
34 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmixqry.lptable" (char\*)  
35 Returns a (pmix\_data\_array\_t) array of **pmix\_proc\_info\_t**, one entry for each  
36 process in the specified namespace executing on the same node as the requester, ordered by  
37 process job rank. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the namespace  
38 whose process table is being queried  
39 **PMIX\_QUERY\_AUTHORIZATIONS** "pmixqry.auths" (bool)  
40 Return operations the PMIx tool is authorized to perform. NO QUALIFIERS  
41 **PMIX\_QUERY\_SPAWN\_SUPPORT** "pmixqry.spawn" (bool)

```

1      Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS
2      PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
3      Return a comma-delimited list of supported debug attributes. NO QUALIFIERS
4      PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
5          Return information on memory usage for the processes indicated in the qualifiers.
6          SUPPORTED QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
7          specific process(es) whose memory usage is being requested
8      PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
9          Constrain the query to local information only. NO QUALIFIERS
10     PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
11     Report only average values for sampled information. NO QUALIFIERS
12     PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
13     Report minimum and maximum values. NO QUALIFIERS
14     PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
15     String identifier of the allocation whose status is being requested. NO QUALIFIERS
16     PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
17     Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
18     SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
19     requested (defaults to allocation containing the caller)
20     PMIX_QUERY_ATTRIBUTE_SUPPORT "pmix.qry.attrs" (bool)
21     Query list of supported attributes for specified APIs. SUPPORTED QUALIFIERS:
22         PMIX_CLIENT_FUNCTIONS , PMIX_SERVER_FUNCTIONS ,
23         PMIX_TOOL_FUNCTIONS , and/or PMIX_HOST_FUNCTIONS
24     PMIX_QUERY_NUM_PSETS "pmix.qry.psetnum" (size_t)
25     Return the number of psets defined in the specified range (defaults to session).
26     PMIX_QUERY_PSET_NAMES "pmix.qry.psets" (char*)
27     Return a comma-delimited list of the names of the psets defined in the specified range
28     (defaults to session).
29     PMIX_QUERY_AVAIL_SERVERS "pmix.qry.asrvrs" (pmix_data_array_t*)
30     Return an array of pmix_info_t , each element itself containing an array of
31         pmix_info_t of available data for servers on this node to which the caller might be able
32         to connect. Each array must include at least one of the rendezvous-required pieces of
33         information.

```

### 34    3.4.22 Log attributes

35    Attributes used to describe **PMIx\_Log** behavior - these are values passed to the **PMIx\_Log** API  
 36    and therefore are not accessed using the **PMIx\_Get** API.

```

37     PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
38         ID of source of the log request
39     PMIX_LOG_STDERR "pmix.log.stderr" (char*)
40         Log string to stderr.
41     PMIX_LOG_STDOUT "pmix.log.stdout" (char*)

```

```

1      Log string to stdout.
2  PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
3      Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
4      otherwise to local syslog
5  PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)
6      Log data to local syslog. Defaults to ERROR priority.
7  PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
8      Forward data to system “gateway” and log msg to that syslog Defaults to ERROR priority.
9  PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)
10     Syslog priority level
11  PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
12      Timestamp for log report
13  PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstamp" (bool)
14      Generate timestamp for log
15  PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
16      Label the output stream with the channel name (e.g., “stdout”)
17  PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
18      Print timestamp in output string
19  PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
20      Print the output stream in XML format
21  PMIX_LOG_ONCE "pmix.log.once" (bool)
22      Only log this once with whichever channel can first support it, taking the channels in priority
23      order
24  PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
25      Message blob to be sent somewhere.
26  PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
27      Log via email based on pmix_info_t containing directives.
28  PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
29      Comma-delimited list of email addresses that are to receive the message.
30  PMIX_LOG_EMAIL_SENDER_ADDR "pmix.log.emfaddr" (char*)
31      Return email address of sender
32  PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
33      Subject line for email.
34  PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
35      Message to be included in email.
36  PMIX_LOG_EMAIL_SERVER "pmix.log.esrvr" (char*)
37      Hostname (or IP address) of estmp server
38  PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (int32_t)
39      Port the email server is listening to
40  PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
41      Store the log data in a global data store (e.g., database)
42  PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
43      Log the provided information to the host environment’s job record

```

## 1 3.4.23 Resource manager attributes

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

```
6 PMIX_RM_NAME "pmix.rm.name" (char*)
7     String name of the RM.
8 PMIX_RM_VERSION "pmix.rm.version" (char*)
9     RM version string.
```

## 10 3.4.24 Environment variable attributes

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

```
13 PMIX_SET_ENVAR "pmix.environ.set" (pmix_envar_t*)
14     Set the envar to the given value, overwriting any pre-existing one
15 PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
16     Unset the environment variable specified in the string.
17 PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
18     Add the environment variable, but do not overwrite any pre-existing one
19 PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
20     Prepend the given value to the specified environmental value using the given separator
21     character, creating the variable if it doesn't already exist
22 PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
23     Append the given value to the specified environmental value using the given separator
24     character, creating the variable if it doesn't already exist
25 PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
26     Ensure the given value appears first in the specified envar using the separator character,
27     creating the envar if it doesn't already exist
```

## 28 3.4.25 Job Allocation attributes

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

```
32 PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
33     User-provided string identifier for this allocation request which can later be used to query
34     status of the request.
35 PMIX_ALLOC_ID "pmix.alloc.id" (char*)
36     A string identifier (provided by the host environment) for the resulting allocation which can
37     later be used to reference the allocated resources in, for example, a call to PMIx_Spawn.
38 PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
```

1 Name of the WLM queue to which the allocation request is to be directed, or the queue being  
 2 referenced in a query.  
 3 **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (**uint64\_t**)  
 4 The number of nodes being requested in an allocation request  
 5 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (**char\***)  
 6 Regular expression of the specific nodes being requested in an allocation request  
 7 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (**uint64\_t**)  
 8 Number of cpus being requested in an allocation request.  
 9 **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (**char\***)  
 10 Regular expression of the number of cpus for each node being requested in an allocation  
 11 request.  
 12 **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (**char\***)  
 13 Regular expression of the specific cpus being requested in an allocation request.  
 14 **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (**float**)  
 15 Number of Megabytes[base2] of memory (per process) being requested in an allocation  
 16 request.  
 17 **PMIX\_ALLOC\_NETWORK** "pmix.alloc.net" (**array**)  
 18 Changed to **PMIX\_ALLOC\_FABRIC**  
 19 **PMIX\_ALLOC\_FABRIC** "pmix.alloc.net" (**array**)  
 20 Array of **pmix\_info\_t** describing requested fabric resources. This must include at least:  
 21 **PMIX\_ALLOC\_FABRIC\_ID**, **PMIX\_ALLOC\_FABRIC\_TYPE**, and  
 22 **PMIX\_ALLOC\_FABRIC\_ENDPTS**, plus whatever other descriptors are desired.  
 23 **PMIX\_ALLOC\_NETWORK\_ID** "pmix.alloc.netid" (**char\***)  
 24 Changed to **PMIX\_ALLOC\_FABRIC\_ID**  
 25 **PMIX\_ALLOC\_FABRIC\_ID** "pmix.alloc.netid" (**char\***)  
 26 The key to be used when accessing this requested fabric allocation. The allocation will be  
 27 returned/stored as a **pmix\_data\_array\_t** of **pmix\_info\_t** indexed by this key and  
 28 containing at least one entry with the same key and the allocated resource description. The  
 29 type of the included value depends upon the fabric support. For example, a TCP allocation  
 30 might consist of a comma-delimited string of socket ranges such as  
 31 "32000-32100,33005,38123-38146". Additional entries will consist of any provided  
 32 resource request directives, along with their assigned values. Examples include:  
 33 **PMIX\_ALLOC\_FABRIC\_TYPE** - the type of resources provided;  
 34 **PMIX\_ALLOC\_FABRIC\_PLANE** - if applicable, what plane the resources were assigned  
 35 from; **PMIX\_ALLOC\_FABRIC\_QOS** - the assigned QoS; **PMIX\_ALLOC\_BANDWIDTH** -  
 36 the allocated bandwidth; **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** - a security key for the  
 37 requested fabric allocation. NOTE: the assigned values may differ from those requested,  
 38 especially if **PMIX\_INFO\_REQD** was not set in the request.  
 39 **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (**float**)  
 40 Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
 41 request.  
 42 **PMIX\_ALLOC\_NETWORK\_QOS** "pmix.alloc.netqos" (**char\***)  
 43 Changed to **PMIX\_ALLOC\_FABRIC\_QOS**

```

1   PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
2       Fabric quality of service level for the job being requested in an allocation request.
3   PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
4       Total session time (in seconds) being requested in an allocation request.
5   PMIX_ALLOC_NETWORK_TYPE "pmix.alloc.nettype" (char*)
6       Changed to PMIX_ALLOC_FABRIC_TYPE
7   PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
8       Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
9   PMIX_ALLOC_NETWORK_PLANE "pmix.alloc.netplane" (char*)
10      Changed to PMIX_ALLOC_FABRIC_PLANE
11   PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
12       ID string for the NIC (aka plane) to be used for the requested allocation (e.g., CIDR for
13       Ethernet)
14   PMIX_ALLOC_NETWORK_ENDPTS "pmix.alloc.endpts" (size_t)
15      Changed to PMIX_ALLOC_FABRIC_ENDPTS
16   PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
17       Number of endpoints to allocate per process in the job
18   PMIX_ALLOC_NETWORK_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
19      Changed to PMIX_ALLOC_FABRIC_ENDPTS_NODE
20   PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
21      Number of endpoints to allocate per node for the job
22   PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
23      Changed to PMIX_ALLOC_FABRIC_SEC_KEY
24   PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
25       Fabric security key for the spawned job

```

## 3.4.26 Job control attributes

Attributes used to request control operations on an executing application - these are values passed to the [PMIx\\_Job\\_control\\_nb](#) API and are not accessed using the [PMIx\\_Get](#) API.

```

29   PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
30       Provide a string identifier for this request. The user can provide an identifier for the
31       requested operation, thus allowing them to later request status of the operation or to
32       terminate it. The host, therefore, shall track it with the request for future reference.
33   PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
34       Pause the specified processes.
35   PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
36       Resume ("un-pause") the specified processes.
37   PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
38       Cancel the specified request - the provided request ID must match the
39       PMIX_JOB_CTRL_ID provided to a previous call to PMIx\_Job\_control. An ID of
40       NULL implies cancel all requests from this requestor.
41   PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)

```

```

1           Forcibly terminate the specified processes and cleanup.
2 PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
3           Restart the specified processes using the given checkpoint ID.
4 PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
5           Checkpoint the specified processes and assign the given ID to it.
6 PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
7           Use event notification to trigger a process checkpoint.
8 PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
9           Use the given signal to trigger a process checkpoint.
10 PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
11           Time in seconds to wait for a checkpoint to complete.
12 PMIX_JOB_CTRL_CHECKPOINT_METHOD
13 "pmix.jctrl.ckmethod" (pmix_data_array_t)
14           Array of pmix_info_t declaring each method and value supported by this application.
15 PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
16           Send given signal to specified processes.
17 PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
18           Regular expression identifying nodes that are to be provisioned.
19 PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimg" (char*)
20           Name of the image that is to be provisioned.
21 PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
22           Indicate that the job can be pre-empted.
23 PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
24           Politely terminate the specified processes.
25 PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)
26           Comma-delimited list of files to be removed upon process termination
27 PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)
28           Comma-delimited list of directories to be removed upon process termination
29 PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)
30           Recursively cleanup all subdirectories under the specified one(s)
31 PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)
32           Only remove empty subdirectories
33 PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)
34           Comma-delimited list of filenames that are not to be removed
35 PMIX_CLEANUP_LEAVE_TOPDIR "pmix.clnup.lvtop" (bool)
36           When recursively cleaning subdirectories, do not remove the top-level directory (the one
37           given in the cleanup request)

```

## 1 3.4.27 Monitoring attributes

2 Attributes used to control monitoring of an executing application- these are values passed to the  
3 [PMIx\\_Process\\_monitor\\_nb](#) API and are not accessed using the [PMIx\\_Get](#) API.

4 **PMIX\_MONITOR\_ID** "pmix.monitor.id" (**char\***)  
5     Provide a string identifier for this request.  
6 **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (**char\***)  
7     Identifier to be canceled (**NULL** means cancel all monitoring for this process).  
8 **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (**bool**)  
9     The application desires to control the response to a monitoring event.  
10 **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (**void**)  
11     Register to have the PMIx server monitor the requestor for heartbeats.  
12 **PMIX\_SEND\_HEARTBEAT** "pmix.monitor.beat" (**void**)  
13     Send heartbeat to local PMIx server.  
14 **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (**uint32\_t**)  
15     Time in seconds before declaring heartbeat missed.  
16 **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (**uint32\_t**)  
17     Number of heartbeats that can be missed before generating the event.  
18 **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (**char\***)  
19     Register to monitor file for signs of life.  
20 **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (**bool**)  
21     Monitor size of given file is growing to determine if the application is running.  
22 **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (**char\***)  
23     Monitor time since last access of given file to determine if the application is running.  
24 **PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (**char\***)  
25     Monitor time since last modified of given file to determine if the application is running.  
26 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (**uint32\_t**)  
27     Time in seconds between checking the file.  
28 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (**uint32\_t**)  
29     Number of file checks that can be missed before generating the event.

## 30 3.4.28 Security attributes

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

32 **PMIX\_CRED\_TYPE** "pmix.sec.ctype" (**char\***)  
33     When passed in [PMIx\\_Get\\_credential](#), a prioritized, comma-delimited list of desired  
34     credential types for use in environments where multiple authentication mechanisms may be  
35     available. When returned in a callback function, a string identifier of the credential type.  
36 **PMIX\_CRYPTO\_KEY** "pmix.sec.key" (**pmix\_byte\_object\_t**)  
37     Blob containing crypto key

## 1 3.4.29 Application setup attributes

```
2 PMIx v3.0 Attributes for controlling contents of application setup data  
3     PMIX_SETUP_APP_ENVARS "pmix.setup.env" (bool)  
4         Harvest and include relevant environmental variables  
5     PMIX_SETUP_APP_NONENVARS "" "pmix.setup.nenv" (bool)  
6         Include all relevant data other than environmental variables  
7     PMIX_SETUP_APP_ALL "pmix.setup.all" (bool)  
8         Include all relevant data
```

## 9 3.4.30 Attribute support level attributes

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

## 26 3.4.31 Descriptive attributes

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

### 1 3.4.32 Process group attributes

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

3     **PMIX\_GROUP\_ID** "pmix.grp.id" (**char\***)  
4         User-provided group identifier

5     **PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (**bool**)  
6         This process is the leader of the group

7     **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (**bool**)  
8         Participation is optional - do not return an error if any of the specified processes terminate  
9         without having joined. The default is false

10    **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (**bool**)  
11         Notify remaining members when another member terminates without first leaving the group.  
12         The default is false

13    **PMIX\_GROUP\_INVITE\_DECLINE** "pmix.grp.decline" (**bool**)  
14         Notify the inviting process that this process does not wish to participate in the proposed  
15         group. The default is true

16    **PMIX\_GROUP\_FT\_COLLECTIVE** "pmix.grp.ftcoll" (**bool**)  
17         Adjust internal tracking for terminated processes. Default is false

18    **PMIX\_GROUP\_MEMBERSHIP** "pmix.grp.mbrs" (**pmix\_data\_array\_t\***)  
19         Array of group member ID's

20    **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (**bool**)  
21         Requests that the RM assign a new context identifier to the newly created group. The  
22         identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
23         specified in the request. Thus, the value serves as a means of identifying the group within  
24         that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

25    **PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (**size\_t**)  
26         Context identifier assigned to the group by the host RM.

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

35    **PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (**pmix\_byte\_object\_t**)  
36         Data collected to be shared during group construction

### 37 3.4.33 Storage-related attributes

38 *PMIx v4.0* Attributes related to storage systems

39     **PMIX\_STORAGE\_ID** "pmix.strg.id" (**char\***)  
40         Identifier of the storage system being referenced

```

1   PMIX_STORAGE_PATH "pmix.strg.path" (char*)
2       Mount point corresponding to a specified storage ID
3   PMIX_STORAGE_TYPE "pmix.strg.type" (char*)
4       Qualifier indicating the type of storage being referenced by a query e.g., lustre, gpfs, online,
5       fabric-attached, ...)
6   PMIX_STORAGE_BW "pmix.strg.bw" (float)
7       Overall bandwidth (in Megabytes[base2]/sec) of specified storage system
8   PMIX_STORAGE_AVAIL_BW "pmix.strg.availbw" (float)
9       Overall bandwidth (in Megabytes[base2]/sec) of specified storage system that is available for
10      use by the calling program
11  PMIX_STORAGE_OBJECT_LIMIT "pmix.strg.obj" (uint64_t)
12      Overall limit on number of objects (e.g., inodes) of specified storage system
13  PMIX_STORAGE_OBJECTS_FREE "pmix.strg.objf" (uint64_t)
14      Number of free objects (e.g., inodes) on specified storage system
15  PMIX_STORAGE_OBJECTS_AVAIL "pmix.strg.obja" (uint64_t)
16      Number of objects (e.g., inodes) on specified storage system that are available for use by the
17      calling program
18  PMIX_STORAGE_CAPACITY_LIMIT "pmix.strg.cap" (uint64_t)
19      Overall capacity (in Megabytes[base2]) of specified storage system
20  PMIX_STORAGE_CAPACITY_FREE "pmix.strg.free" (uint64_t)
21      Free capacity (in Megabytes[base2]) of specified storage system
22  PMIX_STORAGE_CAPACITY_AVAIL "pmix.strg.avail" (uint64_t)
23      Capacity (in Megabytes[[base2]]) of specified storage system that is available for use by the
24      calling program
25  PMIX_QUERY_STORAGE_LIST "pmix.strg.list" (char*)
26      Return comma-delimited list of identifiers for all available storage systems

```

## 3.5 Callback Functions

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

### 3.5.1 Release Callback Function

#### Summary

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

```
1      Format  
2      PMIx v1.0  
3      typedef void (*pmix_release_cfunc_t)  
4          (void *cbdata)  
5      C  
6      INOUT cbdata  
7          Callback data passed to original API call (memory reference)  
8  
9      Description  
10     Since the data is “owned” by the host server, provide a callback function to notify the host server  
11     that we are done with the data so it can be released.
```

### 3.5.2 Modex Callback Function

```
10     Summary  
11     The pmix_modex_cfunc_t is used by the pmix_server_fencenb_fn_t and  
12     pmix_server_dmodex_req_fn_t PMIx server operations to return modex business card  
13     exchange (BCX) data.
```

```
PMIx v1.0  
14     typedef void (*pmix_modex_cfunc_t)  
15         (pmix_status_t status,  
16          const char *data, size_t ndata,  
17          void *cbdata,  
18          pmix_release_cfunc_t release_fn,  
19          void *release_cbdata)  
C  
20     IN status  
21         Status associated with the operation (handle)  
22     IN data  
23         Data to be passed (pointer)  
24     IN ndata  
25         size of the data (size_t)  
26     IN cbdata  
27         Callback data passed to original API call (memory reference)  
28     IN release_fn  
29         Callback for releasing data (function pointer)  
30     IN release_cbdata  
31         Pointer to be passed to release_fn (memory reference)
```

1    **Description**

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

5    **3.5.3 Spawn Callback Function**

6    **Summary**

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

PMIx v1.0

C

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

C

12     **IN status**

13       Status associated with the operation (handle)

14     **IN nspace**

15       Namespace string (`pmix_nspace_t`)

16     **IN cbdata**

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

18    **Description**

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

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

25    **3.5.4 Op Callback Function**

26    **Summary**

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

PMIx v1.0

C

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

C

30     **IN status**

31       Status associated with the operation (handle)

32     **IN cbdata**

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

1    **Description**

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

5    **3.5.5    Lookup Callback Function**

6       **Summary**

7    The `pmix_lookup_cfunc_t` is used by `PMIx_Lookup_nb` to return data.

PMIx v1.0

C

```
8       typedef void (*pmix_lookup_cfunc_t)
9              (pmix_status_t status,
10             pmix_pdata_t data[], size_t ndata,
11             void *cbdata);
```

C

12    **IN    *status***

13    Status associated with the operation (handle)

14    **IN    *data***

15    Array of data returned (`pmix_pdata_t`)

16    **IN    *ndata***

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

18    **IN    *cbdata***

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

20    **Description**

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

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

27    **3.5.6    Value Callback Function**

28       **Summary**

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

PMIx v1.0

C

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

```
1 IN  status  
2     Status associated with the operation (handle)  
3 IN  kv  
4     Key/value pair representing the data ( pmix\_value\_t )  
5 IN  cbdata  
6     Callback data passed to original API call (memory reference)
```

#### 7 **Description**

```
8 A callback function for calls to PMIx\_Get\_nb . The status indicates if the requested data was  
9 found or not. A pointer to the pmix\_value\_t structure containing the found data is returned.  
10 The pointer will be NULL if the requested data was not found.
```

### 11 3.5.7 Info Callback Function

#### 12 **Summary**

```
13 The pmix\_info\_cbfnc\_t is a general information callback used by various APIs.
```

PMIx v2.0

```
14 typedef void (*pmix_info_cbfnc_t)  
15     (pmix\_status\_t status,  
16      pmix\_info\_t info[], size\_t ninfo,  
17      void *cbdata,  
18      pmix\_release\_cbfnc\_t release_fn,  
19      void *release_cbdata);
```

```
20 IN  status  
21     Status associated with the operation ( pmix\_status\_t )  
22 IN  info  
23     Array of pmix\_info\_t returned by the operation (pointer)  
24 IN  ninfo  
25     Number of elements in the info array (size\_t)  
26 IN  cbdata  
27     Callback data passed to original API call (memory reference)  
28 IN  release_fn  
29     Function to be called when done with the info data (function pointer)  
30 IN  release_cbdata  
31     Callback data to be passed to release_fn (memory reference)
```

#### 32 **Description**

```
33 The status indicates if requested data was found or not. An array of pmix\_info\_t will contain  
34 the key/value pairs.
```

## 3.5.8 Event Handler Registration Callback Function

The `pmix_evhdlr_reg_cbfunc_t` callback function.

### Advice to users

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

PMIx v2.0

C

```
7     typedef void (*pmix_evhdlr_reg_cbfunc_t)
8         (pmix_status_t status,
9          size_t evhdlr_ref,
10         void *cbdata)
```

C

**IN status**

Status indicates if the request was successful or not (`pmix_status_t`)

**IN evhdlr\_ref**

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

**IN cbdata**

Callback data passed to original API call (memory reference)

### Description

Define a callback function for calls to `PMIx_Register_event_handler` Deprecated in v4.0 in favor of `pmix_hdlr_reg_cbfunc_t`.

## 3.5.9 Notification Handler Completion Callback Function

### Summary

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

PMIx v2.0

C

```
25     typedef void (*pmix_event_notification_cbfunc_fn_t)
26         (pmix_status_t status,
27          pmix_info_t *results, size_t nresults,
28          pmix_op_cbfunc_t cbfunc, void *thiscbdata,
29          void *notification_cbdata);
```

```

1   IN status
2     Status returned by the event handler's operation (pmix_status_t)
3   IN results
4     Results from this event handler's operation on the event (pmix_info_t)
5   IN nresults
6     Number of elements in the results array (size_t)
7   IN cbfunc
8     pmix_op_cbfunc_t function to be executed when PMIx completes processing the
9     callback (function reference)
10  IN thiscbdata
11    Callback data that was passed in to the handler (memory reference)
12  IN cbdata
13    Callback data to be returned when PMIx executes cbfunc (memory reference)

```

#### 14 **Description**

15 Define a callback by which an event handler can notify the PMIx library that it has completed its  
16 response to the notification. The handler is *required* to execute this callback so the library can  
17 determine if additional handlers need to be called. The handler shall return  
18 `PMIX_EVENT_ACTION_COMPLETE` if no further action is required. The return status of each  
19 event handler and any returned `pmix_info_t` structures will be added to the *results* array of  
20 `pmix_info_t` passed to any subsequent event handlers to help guide their operation.

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

### 23 **3.5.10 Notification Function**

#### 24 **Summary**

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

#### Advice to users

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

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

9 **IN evhdlr\_registration\_id**  
10 Registration number of the handler being called (**size\_t**)  
11 **IN status**  
12 Status associated with the operation (**pmix\_status\_t**)  
13 **IN source**  
14 Identifier of the process that generated the event (**pmix\_proc\_t**). If the source is the SMS,  
15 then the nspace will be empty and the rank will be PMIX\_RANK\_UNDEF  
16 **IN info**  
17 Information describing the event (**pmix\_info\_t**). This argument will be NULL if no  
18 additional information was provided by the event generator.  
19 **IN ninfo**  
20 Number of elements in the info array (**size\_t**)  
21 **IN results**  
22 Aggregated results from prior event handlers servicing this event (**pmix\_info\_t**). This  
23 argument will be **NULL** if this is the first handler servicing the event, or if no prior handlers  
24 provided results.  
25 **IN nresults**  
26 Number of elements in the results array (**size\_t**)  
27 **IN cbfunc**  
28 **pmix\_event\_notification\_cbfnc\_fn\_t** callback function to be executed upon  
29 completion of the handler's operation and prior to handler return (function reference).  
30 **IN cbdata**  
31 Callback data to be passed to cbfunc (memory reference)

## 32 **Description**

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

## Advice to users

Possible uses of the *info* array include:

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

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

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

## Advice to PMIx server hosts

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

### 3.5.11 Server Setup Application Callback Function

The `PMIx_server_setup_application` callback function.

#### Summary

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

## 24 3.5.12 Server Direct Modex Response Callback Function

25 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.

29 Format

```
PMIx v1.0 ▼ C ▼  
30     typedef void (*pmix_dmodex_response_fn_t)(pmix_status_t status,  
31                                     char *data, size_t sz,  
32                                     void *cbdata);
```

- 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.

## 14 3.5.13 PMIx Client Connection Callback Function

## Summary

Callback function for incoming connection request from a local client

## Format

PMIx v1.0

**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.

### 27 3.5.14 PMIx Tool Connection Callback Function

## Summary

Callback function for incoming tool connections.

1  
PMIx v2.0

## Format

C

C

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

## Description

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

## Advice to PMIx server hosts

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

## 16 3.5.15 Credential callback function

## Summary

Callback function to return a requested security credential

## Format

*PMIx v3.0*

C

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

## IN status

**pmix status t** value (handle)

### **IN** credential

**pmix\_bvte\_object\_t** structure containing the security credential (handle)

IN info

Array of provided by the system to pass any additional information about the credential - e.g., the identity of the issuing agent. (handle)

IN ninfo

Number of elements in *info* (**size t**)

IN cbdata

Object passed in original request (memory reference)

## Description

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

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

## Advice to users

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

### 3.5.16 Credential validation callback function

## Summary

Callback function for security credential validation

## Format

C

```
PMIx v3.0 ▼ C

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

9           Array of `pmix_info_t` provided by the system to pass any additional information about  
10          the authentication - e.g., the effective userid and group id of the certificate holder, and any  
11          related authorizations (handle)

IN ninfo

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

IN sbdata

**Object** passed in original request (memory reference)

## Description

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

## Advice to users

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

### 23 3.5.17 IOF delivery function

## Summary

Callback function for delivering forwarded IO to a process

1      **Format**

2      *PMIx v3.0*

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

C

C

6      **IN    iofhdlr**

7      Registration number of the handler being invoked (**size\_t**)

8      **IN    channel**

9      bitmask identifying the channel the data arrived on (**pmix\_iof\_channel\_t**)

10     **IN    source**

11     Pointer to a **pmix\_proc\_t** identifying the namespace/rank of the process that generated the  
12     data (**char\***)

13     **IN    payload**

14     Pointer to character array containing the data.

15     **IN    info**

16     Array of **pmix\_info\_t** provided by the source containing metadata about the payload.  
17     This could include **PMIX\_TOF\_COMPLETE** (handle)

18     **IN    ninfo**

19     Number of elements in *info* (**size\_t**)

20     **Description**

21     Define a callback function for delivering forwarded IO to a process. This function will be called  
22     whenever data becomes available, or a specified buffering size and/or time has been met.

23     **Advice to users**

24     Multiple strings may be included in a given *payload*, and the *payload* may *not* be **NULL** terminated.  
25     The user is responsible for releasing the *payload* memory. The *info* array is owned by the PMIx  
library and is not to be released or altered by the receiving party.

26     **3.5.18    IOF and Event registration function**

27     **Summary**

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

```
1      Format  
2      PMIx v3.0  
3      C  
4  
5      IN  status  
6          PMIX_SUCCESS or an appropriate error constant ( pmix_status_t )  
7      IN  refid  
8          reference identifier assigned to the handler by PMIx, used to deregister the handler (size_t)  
9      IN  cbdata  
10         object provided to the registration call (pointer)  
  
11     Description  
12     Callback function for calls to register handlers, e.g., event notification and IOF requests.
```

## 3.6 Constant String Functions

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

### Summary

```
16    String representation of a pmix_status_t .  
17  
PMIx v1.0  
18    const char*  
19    PMIx_Error_string(pmix_status_t status);  
C  
C
```

### Summary

```
20    String representation of a pmix_proc_state_t .  
21  
PMIx v2.0  
22    const char*  
23    PMIx_Proc_state_string(pmix_proc_state_t state);  
C  
C
```

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

```
1      Summary  
2      String representation of a pmix\_alloc\_directive\_t.  
3      PMIx v2.0   ┌─────────────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────────────┐  
4          const char*  
5          PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);  
6          ┌─────────────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
  
7      Summary  
8      String representation of a pmix\_iof\_channel\_t.  
9      PMIx v3.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────────────┐  
10         const char*  
11         PMIx_IOF_channel_string(pmix_iof_channel_t channel);  
12         ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
  
13     Summary  
14     String representation of a pmix\_job\_state\_t.  
15     PMIx v4.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
16         const char*  
17         PMIx_Job_state_string(pmix_job_state_t state);  
18         ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────┐  
  
19     Summary  
20     String representation of a PMIx attribute  
21     PMIx v4.0   ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────────────┐  
22         const char*  
23         PMIx_Get_attribute_string(char *attributename);  
24         ┌─────────────────────────────────────────────────┐ C ──────────────────────────────────┐
```

1           **Summary**  
2        Return the PMIx attribute name corresponding to the given attribute string

PMIx v4.0

C

3           **const char\***  
4        **PMIx\_Get\_attribute\_name**(char \*attributestring);

C

5           **Summary**  
6        String representation of a **pmix\_link\_state\_t**

PMIx v4.0

C

7           **const char\***  
8        **PMIx\_Link\_state\_string**(pmix\_link\_state\_t state);

C

## CHAPTER 4

# Initialization and Finalization

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

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

### Advice to users

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

## 4.1 Query

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

### 4.1.1 PMIx\_Initialized

#### Format

PMIx v1.0

```
int PMIx_Initialized(void)
```

C

C

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

#### Rationale

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

1           **Description**

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

4           **4.1.2 PMIx\_Get\_version**

5           **Summary**

6       Get the PMIx version information.

7           **Format**

8       *PMIx v1.0*

9       

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

10           **Description**

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

12           **4.2 Client Initialization and Finalization**

13       Initialization and finalization routines for PMIx clients.

14            **Advice to users**

15       The PMIx *ad hoc* v1.0 Standard defined the `PMIx_Init` function, but modified the function  
16       signature in the v1.2 version. The *ad hoc* v1.0 version is not included in this document to avoid  
confusion.  


17           **4.2.1 PMIx\_Init**

18           **Summary**

19       Initialize the PMIx client library

1           **Format**

2            pmix\_status\_t  
3            PMIx\_Init(pmix\_proc\_t \*proc,  
4                        pmix\_info\_t info[], size\_t ninfo)

C

C

5           **INOUT proc**

6            proc structure (handle)

7           **IN info**

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

9           **IN ninfo**

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

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

12           **Optional Attributes**

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

14           **PMIX\_USOCK\_DISABLE "pmix.usock.disable" (bool)**

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

16           **PMIX\_SOCKET\_MODE "pmix.sockmode" (uint32\_t)**

17            POSIX *mode\_t* (9 bits valid). If the library supports socket connections, this attribute may be supported for setting the socket mode.

18           **PMIX\_SINGLE\_LISTENER "pmix.sing.listnr" (bool)**

19            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.

20           **PMIX\_TCP\_REPORT\_URI "pmix.tcp.repuri" (char\*)**

21            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.

22           **PMIX\_TCP\_IF\_INCLUDE "pmix.tcp.ifinclude" (char\*)**

23            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.

24           **PMIX\_TCP\_IF\_EXCLUDE "pmix.tcp.ifexclude" (char\*)**

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

```

1   PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)
2     The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be
3     supported for specifying the port to be used.
4   PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)
5     The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be
6     supported for specifying the port to be used.
7   PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)
8     Set to true to disable IPv4 family of addresses. If the library supports IPV4 connections,
9     this attribute may be supported for disabling it.
10  PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)
11    Set to true to disable IPv6 family of addresses. If the library supports IPV6 connections,
12    this attribute may be supported for disabling it.
13  PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)
14    Pointer to libevent1 event_base to use in place of the internal progress thread.
15  PMIX_GDS_MODULE "pmix.gds.mod" (char*)
16    Comma-delimited string of desired modules. This attribute is specific to the PRI! and
17    controls only the selection of GDS module for internal use by the process. Module selection
18    for interacting with the server is performed dynamically during the connection process.

```



## 19 Description

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

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

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

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

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

---

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

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

### 3    4.2.2 **PMIx\_Finalize**

#### 4       **Summary**

5       Finalize the PMIx client library.

#### 6       **Format**

7       *PMIx v1.0*

8       **pmix\_status\_t**  
8       **PMIx\_Finalize(const pmix\_info\_t info[], size\_t ninfo)**

9       **IN info**  
10       Array of **pmix\_info\_t** structures (array of handles)  
11       **IN ninfo**  
12       Number of element in the *info* array (**size\_t**)

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

#### 14       **Optional Attributes**

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

16       **PMIX\_EMBED\_BARRIER "pmix.embed.barrier" (bool)**

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

#### 19       **Description**

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

## CHAPTER 5

# Key-Value Management

---

1 RHC: DEFINE INSTANT ON - All information required for setup and communication (including  
2 the address vector of endpoints for every process) is available to each process at start of execution

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

11 ON SYSTEMS WHERE THIS CANNOT BE ACHIEVED, THEN IT MAY BE NECESSARY  
12 FOR PROCESSES TO GLOBALLY EXCHANGE INFORMATION. THUS, SUPPORT IS  
13 PROVIDED FOR SUCH EVENTUALITIES.

## 5.1 Reserved Keys

15 RESERVED KEYS BEGIN WITH "PMIX" AND ARE PROVIDED BY THE HOST  
16 ENVIRONMENT. . RESERVED KEYS SHALL NOT BE PUSHED/CIRCULATED BY CLIENT  
17 PROCS. INFO REQUIRED TO BE PROVIDED AT TIME OF CLIENT START OF  
18 EXECUTION.

19 BRING OVER THE ARRAY DEFINITIONS FOR DATA REALMS (SESSION, JOB, APP,  
20 NODE). INCLUDE EXPLANATION OF HOW TO REFERENCE SUCH INFO - WHEN TO USE  
21 RANK\_WILDCARD, RANK\_UNDEF, APPNUM, NODEID, ETC. NOTE THAT THIS ALL  
22 ONLY APPLIES TO RESERVED KEYS

23 EXPLAIN GET BEHAVIOR FOR RESERVED KEYS \* Reserved keys are checked at the local  
24 client cache - if not found, then: \* if it is for a different nspace, the search is referred to the server \*  
25 if they asked us to refresh the cache (PMIX\_GET\_REFRESH\_CACHE) and we are not using the  
26 shmem region, then the search is referred to the server \* if neither of the above, an error is  
27 immediately returned

### 5.1.1 Required Keys

29 BRING OVER THE ATTRIBUTES FROM THE STRUCT CHAPTER AND THE  
30 REGISTER\_NSPACE SECTION - DEFINE THEM HERE, PASTE IN REGISTER\_NSPACE.

## 1 5.1.2 Optional Keys

## 2 5.2 Non-reserved Keys

3           NON-RESERVED KEYS MUST BEGIN WITH A DIFFERENT PREFIX AND ARE PROVIDED  
4           BY CLIENT PROCS. HOST ENVIRONMENTS AND PMIX IMPLEMENTATIONS MAY  
5           CHOOSE TO DEFINE THEIR OWN NON-RESERVED KEYS - IF DEFINED, THESE ARE TO  
6           BE TREATED AS RESERVED.

7           PROVIDE RULES FOR RETRIEVING NON-RESERVED KEYS - HOW THE GET BEHAVIOR  
8           DIFFERS FROM THAT FOR RESERVED KEYS • Get will look in the following places for the  
9           requested key (in order) 1. Local PMIx Client cache (PMIX\_OPTIONAL attribute used to stop  
10          search here) 2. Local PMIx Server cache (PMIX\_IMMEDIATE attribute used to stop search here)  
11          3. Target PMIx Server cache (PMIX\_TIMEOUT attributed used to limit waiting at remote server) •  
12          If the key is not at the target PMIx Server then PMIx\_Get will block until it is or it reaches the  
13          timeout

## 14 5.3 Setting and Accessing Key/Value Pairs

### 15 5.3.1 PMIx\_Put

#### 16       Summary

17       Push a key/value pair into the client's namespace.

#### 18       Format

PMIx v1.0

```
19       pmix_status_t
20       PMIx_Put(pmix_scope_t scope,
21                   const pmix_key_t key,
22                   pmix_value_t *val)
```

C

#### 23       IN scope

24           Distribution scope of the provided value (handle)

#### 25       IN key

26           key (pmix\_key\_t )

#### 27       IN value

28           Reference to a pmix\_value\_t structure (handle)

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

1            **Description**

2            Push a value into the client’s namespace. The client’s PMIx library will cache the information  
3            locally until **PMIx\_Commit** is called.

4            The provided *scope* is passed to the local PMIx server, which will distribute the data to other  
5            processes according to the provided scope. The **pmix\_scope\_t** values are defined in  
6            Section 3.2.10 on page 38. Specific implementations may support different scope values, but all  
7            implementations must support at least **PMIX\_GLOBAL**.

8            The **pmix\_value\_t** structure supports both string and binary values. PMIx implementations  
9            will support heterogeneous environments by properly converting binary values between host  
10          architectures, and will copy the provided *value* into internal memory.

11           **Advice to PMIx library implementers** 

12          The PMIx server library will properly pack/unpack data to accommodate heterogeneous  
13          environments. The host SMS is not involved in this action. The *value* argument must be copied -  
the caller is free to release it following return from the function.  


14           **Advice to users** 

15          The value is copied by the PMIx client library. Thus, the application is free to release and/or  
modify the value once the call to **PMIx\_Put** has completed.

16          Note that keys starting with a string of “**pmix**” are exclusively reserved for the PMIx standard and  
17          must not be used in calls to **PMIx\_Put**. Thus, applications should never use a defined “**PMIX\_**”  
18          attribute as the key in a call to **PMIx\_Put**.  


19          **5.3.2 PMIx\_Get**

20          **Summary**

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

1      **Format**

C

```
2      pmix_status_t  
3      PMIx_Get(const pmix_proc_t *proc, const pmix_key_t key,  
4                const pmix_info_t info[], size_t ninfo,  
5                pmix_value_t **val)
```

C

6      **IN proc**  
7      process reference (handle)  
8      **IN key**  
9      key to retrieve (`pmix_key_t`)  
10     **IN info**  
11     Array of info structures (array of handles)  
12     **IN ninfo**  
13     Number of element in the *info* array (integer)  
14     **OUT val**  
15     value (handle)

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

▼----- Required Attributes -----▼

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

18     **PMIX\_OPTIONAL "pmix.optional" (bool)**

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

21     **PMIX\_IMMEDIATE "pmix.immediate" (bool)**

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

24     **PMIX\_DATA\_SCOPE "pmix.scope" (pmix\_scope\_t)**

25        Scope of the data to be found in a `PMIx_Get` call.

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

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

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

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

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

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

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

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

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

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



## Optional Attributes

26     The following attributes are optional for host environments:

27     **PMIX\_TIMEOUT** "pmix.timeout" (int)

28     Time in seconds before the specified operation should time out (0 indicating infinite) in  
29     error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
30     the target process from ever exposing its data.



## Advice to PMIx library implementers

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

### Description

Retrieve information for the specified *key* associated with the process identified in the given **pmix\_proc\_t**, returning a pointer to the value in the given address. In general, data posted by a process via **PMIx\_Put** and data that refers directly to a process-related value must be retrieved by specifying the rank of the target process. All other information is retrievable using a rank of **PMIX\_RANK\_WILDCARD**, as illustrated in 5.3.5. See 3.4.10 for an explanation regarding use of the *level* attributes, and **PMIx\_server\_register\_nspace** for a description of the available information.

This is a blocking operation - the caller will block until either the specified data becomes available from the specified rank in the *proc* structure, the operation times out should the **PMIX\_TIMEOUT** attribute have been given, or the **PMIX\_OPTIONAL** or the **PMIX\_IMMEDIATE** conditions are met. The caller is responsible for freeing all memory associated with the returned *value* when no longer required.

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

### 5.3.3 PMIx\_Get\_nb

#### Summary

Nonblocking **PMIx\_Get** operation.

#### Format

C

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)
```

```

1   IN  proc
2     process reference (handle)
3   IN  key
4     key to retrieve (string)
5   IN  info
6     Array of info structures (array of handles)
7   IN  ninfo
8     Number of elements in the info array (integer)
9   IN  cbfunc
10    Callback function (function reference)
11   IN  cbdata
12     Data to be passed to the callback function (memory reference)

13 Returns one of the following:
14
15  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
16    will be returned in the provided cbfunc. Note that the library must not invoke the callback
17    function prior to returning from the API.
18
19  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
20    returned success - the cbfunc will not be called
21
22  • a PMIx error constant indicating either an error in the input or that the request was immediately
23    processed and failed - the cbfunc will not be called

24 If executed, the status returned in the provided callback function will be one of the following
25 constants:
26
27  • PMIX_SUCCESS The requested data has been returned
28
29  • PMIX_ERR_NOT_FOUND The requested data was not available
30
31  • a non-zero PMIx error constant indicating a reason for the request's failure
32
33  PMIX_OPTIONAL "pmix.optional" (bool)
34    Look only in the client's local data store for the requested value - do not request data from
35    the PMIx server if not found.

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

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

```

**Required Attributes**

```

1   PMIX_SESSION_INFO "pmix.ssn.info" (bool)
2     Return information about the specified session. If information about a session other than the
3     one containing the requesting process is desired, then the attribute array must contain a
4     PMIX_SESSION_ID attribute identifying the desired target.
5   PMIX_JOB_INFO "pmix.job.info" (bool)
6     Return information about the specified job or namespace. If information about a job or
7     namespace other than the one containing the requesting process is desired, then the attribute
8     array must contain a PMIX_JOBID or PMIX_NSPACE attribute identifying the desired
9     target. Similarly, if information is requested about a job or namespace in a session other than
10    the one containing the requesting process, then an attribute identifying the target session
11    must be provided.
12  PMIX_APP_INFO "pmix.app.info" (bool)
13    Return information about the specified application. If information about an application other
14    than the one containing the requesting process is desired, then the attribute array must
15    contain a PMIX_APPNUM attribute identifying the desired target. Similarly, if information is
16    requested about an application in a job or session other than the one containing the requesting
17    process, then attributes identifying the target job and/or session must be provided.
18  PMIX_NODE_INFO "pmix.node.info" (bool)
19    Return information about the specified node. If information about a node other than the one
20    containing the requesting process is desired, then the attribute array must contain either the
21    PMIX_NODEID or PMIX_HOSTNAME attribute identifying the desired target.
22  PMIX_GET_STATIC_VALUES "pmix.get.static" (bool)
23    Request that any pointers in the returned value point directly to values in the key-value store
24    and indicate that user takes responsibility for properly releasing memory on the returned
25    value (i.e., free'ing the value structure but not the pointer fields). Note that a return status of
26    PMIX_ERR_GET_MALLOC_REQD indicates that direct pointers could not be supported -
27    thus, the returned data contains allocated memory that the user must release.

```



## Optional Attributes

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

```

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

```



## Advice to PMIx library implementers

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

### Description

The callback function will be executed once the specified data becomes available from the identified process and retrieved by the local server. See **PMIx\_Get** for a full description.

## 5.3.4 PMIx\_Store\_internal

### Summary

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

### 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.

### Description

Store some data locally for retrieval by other areas of the proc. This is data that has only internal scope - it will never be “pushed” externally.

## 1 5.3.5 Accessing information: examples

2 This section provides examples illustrating methods for accessing information at various levels.  
3 The intent of the examples is not to provide comprehensive coding guidance, but rather to illustrate  
4 how `PMIx_Get` can be used to obtain information on a `session`, `job`, `application`,  
5 process, and node.

### 6 5.3.5.1 Session-level information

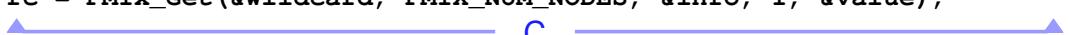
7 The `PMIx_Get` API does not include an argument for specifying the `session` associated with  
8 the information being requested. Information regarding the session containing the requestor can be  
9 obtained by the following methods:

- 10 • for session-level attributes (e.g., `PMIX_UNIV_SIZE`), specifying the requestor's namespace  
11 and a rank of `PMIX_RANK_WILDCARD`; or  
12 • for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_SESSION_INFO`  
13 attribute to indicate that the session-level information for that attribute is being requested

14 Example requests are shown below:



```
15 pmix_info_t info;
16 pmix_value_t *value;
17 pmix_status_t rc;
18 pmix_proc_t myproc, wildcard;
19
20 /* initialize the client library */
21 PMIx_Init(&myproc, NULL, 0);
22
23 /* get the #slots in our session */
24 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
25 rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);
26
27 /* get the #nodes in our session */
28 PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
29 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```



30 Information regarding a different session can be requested by either specifying the namespace and a  
31 rank of `PMIX_RANK_WILDCARD` for a process in the target session, or adding the  
32 `PMIX_SESSION_ID` attribute identifying the target session. In the latter case, the `proc` argument  
33 to `PMIx_Get` will be ignored:

```

1 pmix_info_t info[2];
2 pmix_value_t *value;
3 pmix_status_t rc;
4 pmix_proc_t myproc;
5 uint32_t sid;
6
7 /* initialize the client library */
8 PMIx_Init(&myproc, NULL, 0);
9
10 /* get the #nodes in a different session */
11 sid = 12345;
12 PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
13 PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
14 rc = PMIx_Get(&myproc, PMIX_NUM_NODES, info, 2, &value);

```

### 5.3.5.2 Job-level information

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

- for job-level attributes (e.g., `PMIX_JOB_SIZE` or `PMIX_JOB_NUM_APPS`), specifying the namespace of the job and a rank of `PMIX_RANK_WILDCARD` for the *proc* argument to `PMIx_Get`; or
- for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_JOB_INFO` attribute to indicate that the job-level information for that attribute is being requested

Example requests are shown below:

```

23 pmix_info_t info;
24 pmix_value_t *value;
25 pmix_status_t rc;
26 pmix_proc_t myproc, wildcard;
27
28 /* initialize the client library */
29 PMIx_Init(&myproc, NULL, 0);
30
31 /* get the #apps in our job */
32 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
33 rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);
34
35 /* get the #nodes in our job */
36 PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);
37 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);

```

### 5.3.5.3 Application-level information

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

- for application-level attributes (e.g., `PMIX_APP_SIZE`), specifying the namespace and rank of a process within that application;
- for application-level attributes (e.g., `PMIX_APP_SIZE`), including the `PMIX_APPNUM` attribute specifying the application whose information is being requested. In this case, the namespace field of the *proc* argument is used to reference the `job` containing the application - the `rank` field is ignored;
- or application-level attributes (e.g., `PMIX_APP_SIZE`), including the `PMIX_APPNUM` and `PMIX_NSPACE` or `PMIX_JOBID` attributes specifying the job/application whose information is being requested. In this case, the *proc* argument is ignored;
- for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_APP_INFO` attribute to indicate that the application-level information for that attribute is being requested

Example requests are shown below:

```

pmix_info_t info;
pmix_value_t *value;
pmix_status_t rc;
pmix_proc_t myproc, otherproc;
uint32_t appsize, appnum;

/* initialize the client library */
PMIx_Init(&myproc, NULL, 0);

/* get the #processes in our application */
rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
appsize = value->data.uint32;

/* get the #nodes in an application containing "otherproc".
 * Note that the rank of a process in the other application
 * must be obtained first - a simple method is shown here */

/* assume for this example that we are in the first application
 * and we want the #nodes in the second application - use the
 * rank of the first process in that application, remembering
 * that ranks start at zero */
PMIX_PROC_LOAD(&otherproc, myproc.nspace, appsize);

```

```

1 PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
2 rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);
3
4 /* alternatively, we can directly ask for the #nodes in
5 * the second application in our job, again remembering that
6 * application numbers start with zero */
7 appnum = 1;
8 PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
9 PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
10 rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
11

```

C

#### 5.3.5.4 Process-level information

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

#### 5.3.5.5 Node-level information

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

- for node-level attributes (e.g., `PMIX_NODE_SIZE`), specifying the namespace and rank of a process executing on the target node;
- for node-level attributes (e.g., `PMIX_NODE_SIZE`), including the `PMIX_NODEID` or `PMIX_HOSTNAME` attribute specifying the node whose information is being requested. In this case, the `proc` argument's values are ignored; or
- for non-specific attributes (e.g., `PMIX_MAX_PROCS`), including the `PMIX_NODE_INFO` attribute to indicate that the node-level information for that attribute is being requested

Example requests are shown below:

C

```

26 pmix_info_t info[2];
27 pmix_value_t *value;
28 pmix_status_t rc;
29 pmix_proc_t myproc, otherproc;
30 uint32_t nodeid;
31
32 /* initialize the client library */
33 PMIx_Init(&myproc, NULL, 0);
34
35 /* get the #procs on our node */
36 rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);
37

```

```
1 /* get the #slots on another node */
2 PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);
3 PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);
4 rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);
5
```

C

#### Advice to users

6 An explanation of the use of `PMIx_Get` versus `PMIx_Query_info_nb` is provided in 7.1.4.1.

## 7 5.4 Exchanging Key/Value Pairs

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

### 10 5.4.1 PMIx\_Commit

#### 11 Summary

12 Push all previously `PMIx_Put` values to the local PMIx server.

#### 13 Format

14 *PMIx v1.0*

```
15 pmix_status_t PMIx_Commit(void)
```

C

C

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

#### 16 Description

17 This is an asynchronous operation. The **PRI!** will immediately return to the caller while the data is  
18 transmitted to the local server in the background.

#### Advice to users

19 The local PMIx server will cache the information locally - i.e., the committed data will not be  
20 circulated during `PMIx_Commit`. Availability of the data upon completion of `PMIx_Commit` is  
21 therefore implementation-dependent.

## 1 5.4.2 PMIx\_Fence

### 2 Summary

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

### 5 Format

6 *PMIx v1.0*

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

C

C

#### 9 IN procs

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

#### 11 IN nprocs

12 Number of element in the *procs* array (integer)

#### 13 IN info

14 Array of info structures (array of handles)

#### 15 IN ninfo

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

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

### Required Attributes

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

19 `PMIX_COLLECT_DATA "pmix.collect" (bool)`

20 Collect data and return it at the end of the operation.

### Optional Attributes

21 The following attributes are optional for host environments:

22 `PMIX_TIMEOUT "pmix.timeout" (int)`

23 Time in seconds before the specified operation should time out (0 indicating infinite) in  
24 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
25 the target process from ever exposing its data.

26 `PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)`

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

31 `PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)`

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

### Advice to PMIx library implementers

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

### Description

8 Passing a **NULL** pointer as the *procs* parameter indicates that the fence is to span all processes in  
9 the client's namespace. Each provided **pmix\_proc\_t** struct can pass **PMIX\_RANK\_WILDCARD**  
10 to indicate that all processes in the given namespace are participating.

11 The *info* array is used to pass user requests regarding the fence operation.

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

### Advice to PMIx library implementers

13 **PMIx\_Fence** and its non-blocking form are both *collective* operations. Accordingly, the PMIx  
14 server library is required to aggregate participation by local clients, passing the request to the host  
15 environment once all local participants have executed the API.  
16

### Advice to PMIx server hosts

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

## 5.4.3 **PMIx\_Fence\_nb**

### Summary

21 Execute a nonblocking **PMIx\_Fence** across the processes identified in the specified array of  
22 processes, collecting information posted via **PMIx\_Put** as directed.  
23

1      **Format**

2      **pmix\_status\_t**  
3      **PMIx\_Fence\_nb**(const pmix\_proc\_t procs[], size\_t nprocs,  
4                    const pmix\_info\_t info[], size\_t ninfo,  
5                    pmix\_op\_cbfunc\_t cbfunc, void \*cbdata)

6      **IN procs**  
7      Array of **pmix\_proc\_t** structures (array of handles)  
8      **IN nprocs**  
9      Number of element in the *procs* array (integer)  
10     **IN info**  
11     Array of info structures (array of handles)  
12     **IN ninfo**  
13     Number of element in the *info* array (integer)  
14     **IN cbfunc**  
15     Callback function (function reference)  
16     **IN cbdata**  
17     Data to be passed to the callback function (memory reference)

18     Returns one of the following:

- **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** -----▼

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

28     **PMIX\_COLLECT\_DATA "pmix.collect" (bool)**  
29        Collect data and return it at the end of the operation.

## Optional Attributes

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

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

6 **PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (char\*)

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

11 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)

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

## Advice to PMIx library implementers

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

19 Note that PMIx libraries may choose to implement an optimization for the case where only the  
20 calling process is involved in the fence operation by immediately returning

21 **PMIX\_OPERATION\_SUCCEEDED** from the client’s call in lieu of passing the fence operation to a  
22 PMIx server. Fence operations involving more than just the calling process must be communicated  
23 to the PMIx server for proper execution of the included barrier behavior.

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

## Description

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

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

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

## 1 5.5 Publish and Lookup Data

2 The APIs defined in this section publish data from one client that can be later exchanged and looked  
3 up by another client.

### 4 Advice to PMIx library implementers

5 PMIx libraries that support any of the functions in this section are required to support *all* of them.

### 6 Advice to PMIx server hosts

5 Host environments that support any of the functions in this section are required to support *all* of  
6 them.

### 7 5.5.1 PMIx\_Publish

#### 8 Summary

9 Publish data for later access via [PMIx\\_Lookup](#).

#### 10 Format

PMIx v1.0

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

13 IN info  
14 Array of info structures (array of handles)  
15 IN ninfo  
16 Number of element in the *info* array (integer)

17 Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

#### Required Attributes

18 PMIx libraries are not required to directly support any attributes for this function. However, any  
19 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
20 *required* to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process that  
21 published the info.

## Optional Attributes

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

2   **PMIX\_TIMEOUT** "pmix.timeout" (int)

3   Time in seconds before the specified operation should time out (0 indicating infinite) in  
4   error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5   the target process from ever exposing its data.

6   **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

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

8   **PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)

9   Value for calls to **PMIx\_Publish**.

## Advice to PMIx library implementers

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

## Description

17 Publish the data in the *info* array for subsequent lookup. By default, the data will be published into  
18 the **PMIX\_RANGE\_SESSION** range and with **PMIX\_PERSIST\_APP** persistence. Changes to  
19 those values, and any additional directives, can be included in the **pmix\_info\_t** array. Attempts  
20 to access the data by processes outside of the provided data range will be rejected. The persistence  
21 parameter instructs the server as to how long the data is to be retained.

22 The blocking form will block until the server confirms that the data has been sent to the PMIx  
23 server and that it has obtained confirmation from its host SMS daemon that the data is ready to be  
24 looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be  
25 released upon return from the blocking function call.

## Advice to users

26 Publishing duplicate keys is permitted provided they are published to different ranges.

## Advice to PMIx library implementers

27 Implementations should, to the best of their ability, detect duplicate keys being posted on the same  
28 data range and protect the user from unexpected behavior by returning the  
29 **PMIX\_ERR\_DUPLICATE\_KEY** error.

## 1 5.5.2 PMIx\_Publish\_nb

### 2 Summary

3 Nonblocking [PMIx\\_Publish](#) routine.

### 4 Format

5 *PMIx v1.0*

6     pmix\_status\_t  
7     PMIx\_Publish\_nb(const pmix\_info\_t info[], size\_t ninfo,  
                      pmix\_op\_cbfunc\_t cbfunc, void \*cbdata)

C

C

#### 8 IN info

9     Array of info structures (array of handles)

#### 10 IN ninfo

11     Number of element in the *info* array (integer)

#### 12 IN cbfunc

13     Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

#### 14 IN cbdata

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

16 Returns one of the following:

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

### Required Attributes

24 PMIx libraries are not required to directly support any attributes for this function. However, any  
25 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
26 *required* to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process that  
27 published the info.

## Optional Attributes

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

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

6 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

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

8 **PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)

9 Value for calls to **PMIx\_Publish**.

## Advice to PMIx library implementers

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

### Description

16 Nonblocking **PMIx\_Publish** routine. The non-blocking form will return immediately, executing  
17 the callback when the PMIx server receives confirmation from its host SMS daemon.  
18

19 Note that the function will return an error if a **NULL** callback function is given, and that the *info*  
20 array must be maintained until the callback is provided.

### 5.5.3 PMIx\_Lookup

#### Summary

22 Lookup information published by this or another process with **PMIx\_Publish** or  
23 **PMIx\_Publish\_nb**.

1           **Format**

2        **pmix\_status\_t**  
3        **PMIx\_Lookup**(**pmix\_pdata\_t** *data*[], **size\_t** *ndata*,  
4                            **const pmix\_info\_t** *info*[], **size\_t** *ninfo*)

C

C

5        **INOUT data**

6            Array of publishable data structures (array of handles)

7        **IN ndata**

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

9        **IN info**

10           Array of info structures (array of handles)

11       **IN ninfo**

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

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

▼----- **Required Attributes** -----▼

14       PMIx libraries are not required to directly support any attributes for this function. However, any  
15       provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
16       *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process that is  
17       requesting the info.

▼----- **Optional Attributes** -----▼

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

19       **PMIX\_TIMEOUT** "pmix.timeout" (**int**)

20           Time in seconds before the specified operation should time out (0 indicating infinite) in  
21           error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
22           the target process from ever exposing its data.

23       **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)

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

25       **PMIX\_WAIT** "pmix.wait" (**int**)

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

## Advice to PMIx library implementers

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

### Description

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

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

The *data* parameter consists of an array of **pmix\_pdata\_t** struct with the keys specifying the requested information. Data will be returned for each key in the associated *value* struct. Any key that cannot be found will return with a data type of **PMIX\_UNDEF**. The function will return **PMIX\_SUCCESS** if any values can be found, so the caller must check each data element to ensure it was returned.

The *proc* field in each **pmix\_pdata\_t** struct will contain the namespace/rank of the process that published the data.

## Advice to users

Although this is a blocking function, it will not wait by default for the requested data to be published. Instead, it will block for the time required by the server to lookup its current data and return any found items. Thus, the caller is responsible for ensuring that data is published prior to executing a lookup, using **PMIX\_WAIT** to instruct the server to wait for the data to be published, or for retrying until the requested data is found.

## 1 5.5.4 PMIx\_Lookup\_nb

### 2 Summary

3 Nonblocking version of [PMIx\\_Lookup](#).

### 4 Format

5 *PMIx v1.0*

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

#### 9 IN keys

10 Array to be provided to the callback (array of strings)

#### 11 IN info

12 Array of info structures (array of handles)

#### 13 IN ninfo

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

#### 15 IN cbfunc

16 Callback function (handle)

#### 17 IN cbdata

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

19 Returns one of the following:

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

### 24 Required Attributes

25 PMIx libraries are not required to directly support any attributes for this function. However, any  
26 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
27 required to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process that is  
requesting the info.

## Optional Attributes

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

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

6 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

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

8 **PMIX\_WAIT** "pmix.wait" (int)

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

## Advice to PMIx library implementers

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

## Description

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

### 5.5.5 PMIx\_Unpublish

#### Summary

Unpublish data posted by this process using the given keys.

1           **Format**

2        *pmix\_status\_t*  
3        **PMIx\_Unpublish**(*char \*\*keys*,  
4                    *const pmix\_info\_t info[], size\_t ninfo*)

C

C

5        **IN info**

6            Array of info structures (array of handles)

7        **IN ninfo**

8            Number of element in the *info* array (integer)

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

▼----- **Required Attributes** -----▼

10      PMIx libraries are not required to directly support any attributes for this function. However, any  
11      provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
12      *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process that is  
13      requesting the operation.

▼----- **Optional Attributes** -----▼

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

15      **PMIX\_TIMEOUT "pmix.timeout" (int)**

16          Time in seconds before the specified operation should time out (0 indicating infinite) in  
17          error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
18          the target process from ever exposing its data.

19      **PMIX\_RANGE "pmix.range" (pmix\_data\_range\_t)**

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

▼----- **Advice to PMIx library implementers** -----▼

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

1    **Description**

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

5    By default, the range is assumed to be **PMIX\_RANGE\_SESSION**. Changes to the range, and any  
6    additional directives, can be provided in the *info* array.

7    **5.5.6 PMIx\_Unpublish\_nb**

8    **Summary**

9    Nonblocking version of **PMIx\_Unpublish**.

10   **Format**

PMIx v1.0

C

```
11       pmix_status_t
12      PMIx_Unpublish_nb(char **keys,
13                           const pmix_info_t info[], size_t ninfo,
14                           pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

15    **IN keys**

16       (array of strings)

17    **IN info**

18       Array of info structures (array of handles)

19    **IN ninfo**

20       Number of element in the *info* array (integer)

21    **IN cbfunc**

22       Callback function **pmix\_op\_cbfunc\_t** (function reference)

23    **IN cbdata**

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

25    Returns one of the following:

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

## Required Attributes

1 PMIx libraries are not required to directly support any attributes for this function. However, any  
2 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
3 *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the client process that is  
4 requesting the operation.

## Optional Attributes

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

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

6 Time in seconds before the specified operation should time out (0 indicating infinite) in  
7 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
8 the target process from ever exposing its data.

### **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

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

## Advice to PMIx library implementers

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

## Description

18 Non-blocking form of the **PMIx\_Unpublish** function. The callback function will be executed  
19 once the server confirms removal of the specified data. The *info* array must be maintained until the  
20 callback is provided.  
21

## CHAPTER 6

# Process Management

---

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

### 3 6.1 Abort

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

#### 6 6.1.1 PMIx\_Abort

##### 7 Summary

8 Abort the specified processes

##### 9 Format

PMIx v1.0

```
10 pmix_status_t
11 PMIx_Abort(int status, const char msg[],
12             pmix_proc_t procs[], size_t nprocs)
```

13 **IN status**  
14 Error code to return to invoking environment (integer)  
15 **IN msg**  
16 String message to be returned to user (string)  
17 **IN procs**  
18 Array of **pmix\_proc\_t** structures (array of handles)  
19 **IN nprocs**  
20 Number of elements in the *procs* array (integer)  
21 Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

1           **Description**

2         Request that the host resource manager print the provided message and abort the provided array of  
3         *procs*. A Unix or POSIX environment should handle the provided status as a return error code from  
4         the main program that launched the application. A **NULL** for the *procs* array indicates that all  
5         processes in the caller's namespace are to be aborted, including itself. Passing a **NULL** *msg*  
6         parameter is allowed.

7           **Advice to users**

8         The response to this request is somewhat dependent on the specific resource manager and its  
9         configuration (e.g., some resource managers will not abort the application if the provided status is  
10        zero unless specifically configured to do so, and some cannot abort subsets of processes in an  
11        application), and thus lies outside the control of PMIx itself. However, the PMIx client library shall  
12        inform the RM of the request that the specified *procs* be aborted, regardless of the value of the  
provided status.

13        Note that race conditions caused by multiple processes calling **PMIx\_Abort** are left to the server  
14        implementation to resolve with regard to which status is returned and what messages (if any) are  
15        printed.

16        

## 6.2 Process Creation

17        The **PMIx\_Spawn** commands spawn new processes and/or applications in the PMIx universe.  
18        This may include requests to extend the existing resource allocation or obtain a new one, depending  
19        upon provided and supported attributes.

20        

### 6.2.1 PMIx\_Spawn

21           **Summary**

22         Spawn a new job.

1           **Format**

2            pmix\_status\_t  
3        **PMIx\_Spawn**(const pmix\_info\_t job\_info[], size\_t ninfo,  
4                            const pmix\_app\_t apps[], size\_t napps,  
5                            char nspace[])

C

C

6       **IN**    job\_info  
7        Array of info structures (array of handles)  
8       **IN**    ninfo  
9        Number of elements in the *job\_info* array (integer)  
10      **IN**    apps  
11     Array of **pmix\_app\_t** structures (array of handles)  
12      **IN**    napps  
13     Number of elements in the *apps* array (integer)  
14      **OUT**   nspace  
15     Namespace of the new job (string)

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

▼----- Required Attributes -----▼

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

20     **PMIX\_SPAWNED** "pmix.spawned" (bool)  
21       **true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return  
22       status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.  
  
23     **PMIX\_PARENT\_ID** "pmix.parent" (pmix\_proc\_t)  
24       Process identifier of the parent process of the calling process.  
  
25     **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)  
26       The requesting process is a PMIx client.  
  
27     **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)  
28       The requesting process is a PMIx tool.

---

29  
30     Host environments that implement support for **PMIx\_Spawn** are required to pass the  
31     **PMIX\_SPAWNED** and **PMIX\_PARENT\_ID** attributes to all PMIx servers launching new child  
32     processes so those values can be returned to clients upon connection to the PMIx server. In  
33     addition, they are required to support the following attributes when present in either the *job\_info* or  
34     the *info* array of an element of the *apps* array:

```

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

3   PMIX_SET_SESSION_CWD "pmix.ssncwd" (bool)
4     Set the application's current working directory to the session working directory assigned by
5       the RM - when accessed using PMIx_Get, use the PMIX_RANK_WILDCARD value for
6       the rank to discover the session working directory assigned to the provided namespace

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

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

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

```

### Optional Attributes

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

```

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

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

18  PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)
19    Preload binaries onto nodes.

20  PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)
21    Comma-delimited list of files to pre-position on nodes.

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

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

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

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

32  PMIX_MAPBY "pmix.mapby" (char*)

```

```

1      Process mapping policy - when accessed using PMIx_Get , use the
2      PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
3      provided namespace
4      PMIX_RANKBY "pmix.rankby" (char*)
5          Process ranking policy - when accessed using PMIx_Get , use the
6          PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the
7          provided namespace
8      PMIX_BINDTO "pmix.bindto" (char*)
9          Process binding policy - when accessed using PMIx_Get , use the
10         PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the
11         provided namespace
12     PMIX_NON_PMI "pmix.nonpmi" (bool)
13         Spawnsed processes will not call PMIx_Init .
14     PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
15         Spawnsed process rank that is to receive any forwarded stdin.
16     PMIX_TAG_OUTPUT "pmix.tagout" (bool)
17         Tag application output with the identity of the source process.
18     PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
19         Timestamp output from applications.
20     PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
21         Merge stdout and stderr streams from application processes.
22     PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
23         Direct application output (both stdout and stderr) into files of form "<filename>.rank"
24     PMIX_INDEX_ARGV "pmix.indxargv" (bool)
25         Mark the argv with the rank of the process.
26     PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
27         Number of cpus to assign to each rank - when accessed using PMIx_Get , use the
28         PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the
29         provided namespace
30     PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
31         Do not place processes on the head node.
32     PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
33         Do not oversubscribe the cpus.
34     PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
35         Report bindings of the individual processes.
36     PMIX_CPU_LIST "pmix.cpulist" (char*)

```

```

1      List of cpus to use for this job - when accessed using PMIx_Get , use the
2      PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided
3      namespace

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

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

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

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

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

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

18     PMIX-prepend_envar "pmix.environ.prepend" (pmix_envar_t*)
19         Prepend the given value to the specified environmental value using the given separator
20         character, creating the variable if it doesn't already exist

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

24     PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
25         Ensure the given value appears first in the specified envar using the separator character,
26         creating the envar if it doesn't already exist

27     PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
28         Name of the WLM queue to which the allocation request is to be directed, or the queue being
29         referenced in a query.

30     PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
31         Total session time (in seconds) being requested in an allocation request.

32     PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
33         The number of nodes being requested in an allocation request

34     PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
35         Regular expression of the specific nodes being requested in an allocation request

36     PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)

```

```

1      Number of cpus being requested in an allocation request.
2      PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
3          Regular expression of the number of cpus for each node being requested in an allocation
4          request.
5      PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
6          Regular expression of the specific cpus being requested in an allocation request.
7      PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
8          Number of Megabytes[base2] of memory (per process) being requested in an allocation
9          request.
10     PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
11         Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
12         request.
13     PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
14         Fabric quality of service level for the job being requested in an allocation request.
15     PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
16         Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
17     PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
18         ID string for the NIC (aka plane) to be used for the requested allocation (e.g., CIDR for
19         Ethernet)
20     PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
21         Number of endpoints to allocate per process in the job
22     PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
23         Number of endpoints to allocate per node for the job

```



## 24 **Description**

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

28 By default, the spawned processes will be PMIx “connected” to the parent process upon successful  
29 launch (see **PMIx\_Connect** description for details). Note that this only means that (a) the parent  
30 process will be given a copy of the new job's information so it can query job-level info without  
31 incurring any communication penalties, (b) newly spawned child processes will receive a copy of  
32 the parent processes job-level info, and (c) both the parent process and members of the child job  
33 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.

## Advice to PMIx library implementers

Tools may utilize **PMIx\_Spawn** to start intermediate launchers as described in Section 14.2.2. 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.

### 6.2.2 PMIx\_Spawn\_nb

#### Summary

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

#### Format

PMIx v1.0

C

```
pmix_status_t  
PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,  
               const pmix_app_t apps[], size_t napps,  
               pmix_spawn_cbfunc_t cbfunc, void *cbdata)
```

C

#### IN job\_info

Array of info structures (array of handles)

#### IN ninfo

Number of elements in the *job\_info* array (integer)

#### IN apps

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

#### IN cbfunc

Callback function **pmix\_spawn\_cbfunc\_t** (function reference)

#### IN cbdata

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

Returns one of the following:

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

## Required Attributes

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

4 **PMIX\_SPAWNED** "pmix.spawned" (bool)  
5     **true** if this process resulted from a call to **PMIx\_Spawn**. Lack of inclusion (i.e., a return  
6     status of **PMIX\_ERR\_NOT\_FOUND**) corresponds to a value of **false** for this attribute.

7 **PMIX\_PARENT\_ID** "pmix.parent" (pmix\_proc\_t)  
8     Process identifier of the parent process of the calling process.

9 **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)  
10    The requesting process is a PMIx client.

11 **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)  
12    The requesting process is a PMIx tool.

---

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

19 **PMIX\_WDIR** "pmix.wdir" (char\*)  
20    Working directory for spawned processes.

21 **PMIX\_SET\_SESSION\_CWD** "pmix.ssncwd" (bool)  
22    Set the application's current working directory to the session working directory assigned by  
23    the RM - when accessed using **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for  
24    the rank to discover the session working directory assigned to the provided namespace

25 **PMIX\_PREFIX** "pmix.prefix" (char\*)  
26    Prefix to use for starting spawned processes.

27 **PMIX\_HOST** "pmix.host" (char\*)  
28    Comma-delimited list of hosts to use for spawned processes.

29 **PMIX\_HOSTFILE** "pmix.hostfile" (char\*)  
30    Hostfile to use for spawned processes.

## Optional Attributes

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

1           **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (**char\***)  
2            Hostfile listing hosts to add to existing allocation.  
3  
4           **PMIX\_ADD\_HOST** "pmix.addhost" (**char\***)  
5            Comma-delimited list of hosts to add to the allocation.  
6  
7           **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (**bool**)  
8            Preload binaries onto nodes.  
9  
10          **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (**char\***)  
11         Comma-delimited list of files to pre-position on nodes.  
12  
13          **PMIX\_PERSONALITY** "pmix.pers" (**char\***)  
14         Name of personality to use.  
15  
16          **PMIX\_MAPPER** "pmix.mapper" (**char\***)  
17         Mapping mechanism to use for placing spawned processes - when accessed using  
18         **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping  
19         mechanism used for the provided namespace.  
20  
21          **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (**bool**)  
22         Display process mapping upon spawn.  
23  
24          **PMIX\_PPR** "pmix.ppr" (**char\***)  
25         Number of processes to spawn on each identified resource.  
26  
27          **PMIX\_MAPBY** "pmix.mapby" (**char\***)  
28         Process mapping policy - when accessed using **PMIx\_Get**, use the  
29         **PMIX\_RANK\_WILDCARD** value for the rank to discover the mapping policy used for the  
30         provided namespace  
31  
32          **PMIX\_RANKBY** "pmix.rankby" (**char\***)  
33         Process ranking policy - when accessed using **PMIx\_Get**, use the  
34         **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the  
35         provided namespace  
36  
37          **PMIX\_BINDTO** "pmix.bindto" (**char\***)  
38         Process binding policy - when accessed using **PMIx\_Get**, use the  
39         **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
40         provided namespace  
41  
42          **PMIX\_NON\_PMI** "pmix.nonpmi" (**bool**)  
43         Spawned processes will not call **PMIx\_Init**.  
44  
45          **PMIX\_STDIN\_TGT** "pmix.stdin" (**uint32\_t**)  
46         Spawned process rank that is to receive any forwarded **stdin**.

```

1   PMIX_TAG_OUTPUT "pmix.tagout" (bool)
2     Tag application output with the identity of the source process.
3
4   PMIX_TIMESTAMP_OUTPUT "pmix.tsout" (bool)
5     Timestamp output from applications.
6
7   PMIX_MERGE_STDERR_STDOUT "pmix.mergeerrout" (bool)
8     Merge stdout and stderr streams from application processes.
9
10  PMIX_OUTPUT_TO_FILE "pmix.outfile" (char*)
11    Direct application output (both stdout and stderr) into files of form "<filename>.rank"
12
13  PMIX_INDEX_ARGV "pmix.idxargv" (bool)
14    Mark the argv with the rank of the process.
15
16  PMIX_CPUS_PER_PROC "pmix.cpuperproc" (uint32_t)
17    Number of cpus to assign to each rank - when accessed using PMIx_Get , use the
18    PMIX_RANK_WILDCARD value for the rank to discover the cpus/process assigned to the
19    provided namespace
20
21  PMIX_NO_PROCS_ON_HEAD "pmix.nolocal" (bool)
22    Do not place processes on the head node.
23
24  PMIX_NO_OVERSUBSCRIBE "pmix.noover" (bool)
25    Do not oversubscribe the cpus.
26
27  PMIX_REPORT_BINDINGS "pmix.repbind" (bool)
28    Report bindings of the individual processes.
29
30  PMIX_CPU_LIST "pmix.cpulist" (char*)
31    List of cpus to use for this job - when accessed using PMIx_Get , use the
32    PMIX_RANK_WILDCARD value for the rank to discover the cpu list used for the provided
33    namespace
34
35  PMIX_JOB_RECOVERABLE "pmix.recover" (bool)
36    Application supports recoverable operations.
37
38  PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
39    Application is continuous, all failed processes should be immediately restarted.
40
41  PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
42    Maximum number of times to restart a job - when accessed using PMIx_Get , use the
43    PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
44    namespace
45
46  PMIX_SET_ENVAR "pmix.environ.set" (pmix_envar_t*)
47    Set the envar to the given value, overwriting any pre-existing one
48
49  PMIX_UNSET_ENVAR "pmix.environ.unset" (char*)
50    Unset the environment variable specified in the string.

```

```

1   PMIX_ADD_ENVAR "pmix.environ.add" (pmix_envar_t*)
2     Add the environment variable, but do not overwrite any pre-existing one
3
4   PMIX_PREPEND_ENVAR "pmix.environ.prepend" (pmix_envar_t*)
5     Prepend the given value to the specified environmental value using the given separator
6     character, creating the variable if it doesn't already exist
7
8   PMIX_APPEND_ENVAR "pmix.environ.append" (pmix_envar_t*)
9     Append the given value to the specified environmental value using the given separator
10    character, creating the variable if it doesn't already exist
11
12  PMIX_FIRST_ENVAR "pmix.environ.first" (pmix_envar_t*)
13    Ensure the given value appears first in the specified envar using the separator character,
14    creating the envar if it doesn't already exist
15
16  PMIX_ALLOC_QUEUE "pmix.alloc.queue" (char*)
17    Name of the WLM queue to which the allocation request is to be directed, or the queue being
18    referenced in a query.
19
20  PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
21    Total session time (in seconds) being requested in an allocation request.
22
23  PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
24    The number of nodes being requested in an allocation request
25
26  PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)
27    Regular expression of the specific nodes being requested in an allocation request
28
29  PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
30    Number of cpus being requested in an allocation request.
31
32  PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
33    Regular expression of the number of cpus for each node being requested in an allocation
34    request.
35
36  PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
37    Regular expression of the specific cpus being requested in an allocation request.
38
39  PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
40    Number of Megabytes[base2] of memory (per process) being requested in an allocation
41    request.
42
43  PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
44    Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
45    request.
46
47  PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)
48    Fabric quality of service level for the job being requested in an allocation request.
49
50  PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)

```

1 Type of desired transport (e.g., “tcp”, “udp”) being requested in an allocation request.  
2 **PMIX\_ALLOC\_FABRIC\_PLANE** “pmix.alloc.netplane” (**char\***)  
3 ID string for the NIC (aka *plane*) to be used for the requested allocation (e.g., CIDR for  
4 Ethernet)  
5 **PMIX\_ALLOC\_FABRIC\_ENDPTS** “pmix.alloc.endpts” (**size\_t**)  
6 Number of endpoints to allocate per process in the job  
7 **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** “pmix.alloc.endpts.nd” (**size\_t**)  
8 Number of endpoints to allocate per node for the job



9 **Description**

10 Nonblocking version of the **PMIx\_Spawn** routine. The provided callback function will be  
11 executed upon successful start of *all* specified application processes.



12 **Advice to users**

13 Behavior of individual resource managers may differ, but it is expected that failure of any  
14 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.



15 

## 6.3 Connecting and Disconnecting Processes

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

## Advice to PMIx server hosts

The host environment may choose to assign a new namespace to the connected assemblage and/or assign new ranks for its members for its own internal tracking purposes. However, it is not required to communicate such assignments to the participants (e.g., in response to an appropriate call to `PMIx_Query_info_nb`). The host environment is required to generate a `PMIX_ERR_INVALID_TERMINATION` event should any process in the assemblage terminate or call `PMIx_Finalize` without first *disconnecting* from the assemblage.

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

## Advice to users

Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation. While not explicitly prohibited, users are advised that a PMIx implementation or host environment may return an error in such cases.

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

### 6.3.1 PMIx\_Connect

#### Summary

Connect namespaces.

1                   **Format**

2     *PMIx v1.0*      C

3     **pmix\_status\_t**  
4     **PMIx\_Connect**(**const pmix\_proc\_t procs[], size\_t nprocs,**  
       **const pmix\_info\_t info[], size\_t ninfo**)

5     IN **procs**  
       Array of proc structures (array of handles)  
6     IN **nprocs**  
       Number of elements in the *procs* array (integer)  
7     IN **info**  
       Array of info structures (array of handles)  
8     IN **ninfo**  
       Number of elements in the *info* array (integer)

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

10                   **Required Attributes**

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

13                   **Optional Attributes**

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

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

16     **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.

17     **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (**bool**)  
       If **true**, indicates that the requested choice of algorithm is mandatory.

## Advice to PMIx library implementers

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

## Description

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

## Advice to users

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

## Advice to PMIx library implementers

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

## Advice to PMIx server hosts

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

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

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

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

## 1 6.3.2 PMIx\_Connect\_nb

### 2 Summary

3 Nonblocking [PMIx\\_Connect\\_nb](#) routine.

### 4 Format

5 *PMIx v1.0*

6 C

```
7 pmix_status_t  
8 PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,  
9                 const pmix_info_t info[], size_t ninfo,  
10                pmix_op_cbfunc_t cbfunc, void *cbdata)
```

11 **IN procs**  
12 Array of proc structures (array of handles)  
13 **IN nprocs**  
14 Number of elements in the *procs* array (integer)  
15 **IN info**  
16 Array of info structures (array of handles)  
17 **IN ninfo**  
18 Number of element in the *info* array (integer)  
19 **IN cbfunc**  
20 Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)  
21 **IN cbdata**  
22 Data to be passed to the callback function (memory reference)

23 Returns one of the following:

- 24
- 25 • [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
26 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
27 function prior to returning from the API.
  - 28 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
29 returned *success* - the *cbfunc* will *not* be called
  - 30 • a PMIx error constant indicating either an error in the input or that the request was immediately  
31 processed and failed - the *cbfunc* will *not* be called

### 32 Required Attributes

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

## Optional Attributes

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

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

6 **PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (char\*)

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

11 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)

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

## Advice to PMIx library implementers

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

### Description

20 Nonblocking version of **PMIx\_Connect**. The callback function is called once all processes  
21 identified in *procs* have called either **PMIx\_Connect** or its non-blocking version, *and* the host  
22 environment has completed any supporting operations required to meet the terms of the PMIx  
23 definition of *connected* processes. See the advice provided in the description for **PMIx\_Connect**  
24 for more information.

### 6.3.3 PMIx\_Disconnect

#### Summary

27 Disconnect a previously connected set of processes.

1                   **Format**

2     *pmix\_status\_t*  
 3     **PMIx\_Disconnect**(const *pmix\_proc\_t* *procs*[], *size\_t* *nprocs*,  
       const *pmix\_info\_t* *info*[], *size\_t* *ninfo*);

5     **IN    *procs***  
 6     Array of proc structures (array of handles)  
 7     **IN    *nprocs***  
 8     Number of elements in the *procs* array (integer)  
 9     **IN    *info***  
 10    Array of info structures (array of handles)  
 11    **IN    *ninfo***  
 12    Number of element in the *info* array (integer)

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

▼----- **Required Attributes** -----▼

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

▲----- -----▲

▼----- **Optional Attributes** -----▼

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

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

▲----- -----▲

▼----- **Advice to PMIx library implementers** -----▼

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

1            **Description**

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

7             **Advice to users** 

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


11           **Advice to PMIx library implementers** 

12          **PMIx\_Disconnect** and its non-blocking form are both *collective* operations. Accordingly, the  
13          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.  


14           **Advice to PMIx server hosts** 

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


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

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

24          **6.3.4 PMIx\_Disconnect\_nb**

25          **Summary**

26          Nonblocking **PMIx\_Disconnect** routine.

## 1 Format

```
2 PMIx v1.0
3 pmix_status_t
4 PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,
5                      const pmix_info_t info[], size_t ninfo,
6                      pmix_op_cbfunc_t cbfunc, void *cbdata);
```

6 **IN procs**  
7 Array of proc structures (array of handles)  
8 **IN nprocs**  
9 Number of elements in the *procs* array (integer)  
10 **IN info**  
11 Array of info structures (array of handles)  
12 **IN ninfo**  
13 Number of element in the *info* array (integer)  
14 **IN cbfunc**  
15 Callback function **pmix\_op\_cbfunc\_t** (function reference)  
16 **IN cbdata**  
17 Data to be passed to the callback function (memory reference)

18 Returns one of the following:

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

## Required Attributes

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

## Optional Attributes

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

29 **PMIX\_TIMEOUT "pmix.timeout" (int)**

30 Time in seconds before the specified operation should time out (0 indicating infinite) in  
31 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
32 the target process from ever exposing its data.

## Advice to PMIx library implementers

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

### Description

Nonblocking `PMIx_Disconnect` routine. The callback function is called once all processes identified in *procs* have called either `PMIx_Disconnect_nb` or its blocking version, *and* the host environment has completed any required supporting operations. See the advice provided in the description for `PMIx_Disconnect` for more information.

## 6.4 Process Locality

The relative locality of processes is often used to optimize their interactions with the hardware and other processes. PMIx provides a means by which the host environment can communicate the locality of a given process using the `PMIx_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.

### 6.4.1 `PMIx_Load_topology`

#### Summary

Load the local hardware topology description

1           **Format**  
2        PMIx v4.0  
3        pmix\_status\_t  
4        PMIx\_Load\_topology(pmix\_topology\_t \*topo);  
5           C

6           **INOUT topo**

7           Address of a **pmix\_topology\_t** structure where the topology information is to be loaded  
8           (handle)

9           Returns **PMIX\_SUCCESS**, indicating that the *topo* was successfully loaded, or an appropriate  
10          PMIx error constant.

11          **Description**

12          Obtain a pointer to the topology description of the local node. If the *source* field of the provided  
13          **pmix\_topology\_t** is set, then the PMIx library must return a description from the specified  
14          implementation or else indicate that the implementation is not available by returning the  
15          **PMIX\_ERR\_NOT\_SUPPORTED** error constant.

16          The returned pointer may point to a shared memory region or an actual instance of the topology  
17          description. In either case, the description shall be treated as a "read-only" object - attempts to  
18          modify the object are likely to fail and return an error. The PMIx library is responsible for  
19          performing any required cleanup when the client library finalizes.

20          **Advice to users**

21          It is the responsibility of the user to ensure that the *topo* argument is properly initialized prior to  
22          calling this API, and to check the returned *source* to verify that the returned topology description is  
23          compatible with the user's code.

## 24      **6.4.2 PMIx\_Get\_relative\_locality**

25          **Summary**

26          Get the relative locality of two local processes given their locality strings.

```
1      Format  
2      PMIx v4.0  
3      pmix_status_t  
4      PMIx_Get_relative_locality(const char *locality1,  
5                                  const char *locality2,  
6                                  pmix_locality_t *locality);  
7  
8      IN  locality1  
9          String returned by the PMIx\_generate\_locality\_string API (handle)  
10     IN  locality2  
11        String returned by the PMIx\_generate\_locality\_string API (handle)  
12     INOUT locality  
13        Location where the relative locality bitmask is to be constructed (memory reference)  
14  
15 Returns PMIX\_SUCCESS, indicating that the locality was successfully loaded, or an appropriate  
16 PMIx error constant.
```

#### 14 Description

```
15 Parse the locality strings of the two processes and set the appropriate pmix\_locality\_t  
16 locality bits in the provided memory location.
```

### 17 6.4.2.1 Topology description

```
18 The pmix\_topology\_t structure contains a (case-insensitive) string identifying the source of  
19 the topology (e.g., "hwloc") and a pointer to the corresponding implementation-specific topology  
20 description.
```

```
PMIx v4.0  
21      typedef struct pmix_topology {  
22          char *source;  
23          void *topology;  
24      } pmix_topology_t;
```

### 25 6.4.2.2 Initialize the topology description structure

```
26 Initialize the pmix\_topology\_t fields to NULL
```

```
PMIx v4.0  
27      PMIX_TOPOLOGY_CONSTRUCT(m)
```

```
28      IN  m  
29          Pointer to the structure to be initialized (pointer to pmix\_topology\_t)
```

### 6.4.2.3 Relative locality of two processes

2	<i>PMIx v4.0</i>	The <code>pmix_locality_t</code> datatype is a <code>uint16_t</code> bitmask that defines the relative locality of two processes on a node. The following constants represent specific bits in the mask and can be used to test a locality value using standard bit-test methods.
5	<code>PMIX_LOCALITY_UNKNOWN</code>	All bits are set to zero, indicating that the relative locality of the two processes is unknown
7	<code>PMIX_LOCALITY_NONLOCAL</code>	The two processes do not share any common locations
8	<code>PMIX_LOCALITY_SHARE_HWTHREAD</code>	The two processes share at least one hardware thread
9	<code>PMIX_LOCALITY_SHARE_CORE</code>	The two processes share at least one core
10	<code>PMIX_LOCALITY_SHARE_L1CACHE</code>	The two processes share at least an L1 cache
11	<code>PMIX_LOCALITY_SHARE_L2CACHE</code>	The two processes share at least an L2 cache
12	<code>PMIX_LOCALITY_SHARE_L3CACHE</code>	The two processes share at least an L3 cache
13	<code>PMIX_LOCALITY_SHARE_PACKAGE</code>	The two processes share at least a package
14	<code>PMIX_LOCALITY_SHARE_NODE</code>	The two processes are executing on the same node

Implementers and vendors may choose to extend these definitions as needed to describe a particular system.

### 6.4.2.4 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.

`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 therefore been deprecated and the key will be removed in a future release.

## CHAPTER 7

# Job Management and Reporting

---

The job management APIs provide an application with the ability to orchestrate its operation in partnership with the SMS. Members of this category include the `PMIx_Allocation_request_nb`, `PMIx_Job_control_nb`, and `PMIx_Process_monitor_nb` APIs.

## 7.1 Query

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

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

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

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

### 7.1.1 `PMIx_Resolve_peers`

#### Summary

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

```
1      Format  
2      PMIx v1.0  
3      pmix_status_t  
4      PMIx_Resolve_peers(const char *nodename,  
5                          const pmix_nspace_t nspace,  
6                          pmix_proc_t **procs, size_t *nprocs)  
7  
8      IN  nodename  
9          Name of the node to query (string)  
10     IN   nspace  
11        namespace (string)  
12     OUT  procs  
13        Array of process structures (array of handles)  
14     OUT  nprocs  
15        Number of elements in the procs array (integer)  
16  
17        Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.  
18  
19  
20  
Description  
Given a nodename, return the array of processes within the specified nspace that are executing on  
that node. If the nspace is NULL, then all processes on the node will be returned. If the specified  
node does not currently host any processes, then the returned array will be NULL, and nprocs will  
be 0. The caller is responsible for releasing the procs array when done with it. The  
PMIX_PROC_FREE macro is provided for this purpose.
```

## 7.1.2 PMIx\_Resolve\_nodes

```
22     Summary  
23     Return a list of nodes hosting processes within the given namespace.  
24     Format  
25     PMIx v1.0  
26     pmix_status_t  
27     PMIx_Resolve_nodes(const char *nspace, char **nodelist)  
28  
29     IN  nspace  
30        Namespace (string)  
31     OUT  nodelist  
32        Comma-delimited list of nodenames (string)  
33  
34        Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

## Description

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

### 5 7.1.3 PMIx\_Query\_info

## Summary

Query information about the system in general.

## Format

PMIx v4.0

8

```
pmix_status_t  
PMIx_Query_info(pmix_query_t queries[], size_t nqueries,  
                 pmix_info_t *info[], size_t *ninfo)
```

## IN queries

**queries**  
Array of query structures (array of handles)

## IN array of qualities

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

INOUT info

Address where a pointer to an array of `pmix_info_t` containing the results of the query can be returned (memory reference).

#### **INOUT** pinfo

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

Returns one of the following:

- **PMIX\_SUCCESS** All data has been returned
  - **PMIX\_ERR\_NOT\_FOUND** None of the requested data was available
  - **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the data has been returned
  - **PMIX\_ERR\_NOT\_SUPPORTED** The host RM does not support this function
  - a non-zero PMIx error constant indicating a reason for the request's failure

## Required Attributes

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

**PMIX\_QUERY\_REFRESH\_CACHE** "pmix.qry.rfsh" (bool)

Retrieve updated information from server. NO QUALIFIERS

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

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

4       **PMIX\_JOB\_INFO** "pmix.job.info" (bool)  
5       Return information about the specified job or namespace. If information about a job or  
6       namespace other than the one containing the requesting process is desired, then the attribute  
7       array must contain a **PMIX\_JOBID** or **PMIX\_NSPACE** attribute identifying the desired  
8       target. Similarly, if information is requested about a job or namespace in a session other than  
9       the one containing the requesting process, then an attribute identifying the target session  
10      must be provided.

11      **PMIX\_APP\_INFO** "pmix.app.info" (bool)  
12      Return information about the specified application. If information about an application other  
13      than the one containing the requesting process is desired, then the attribute array must  
14      contain a **PMIX\_APPNUM** attribute identifying the desired target. Similarly, if information is  
15      requested about an application in a job or session other than the one containing the requesting  
16      process, then attributes identifying the target job and/or session must be provided.

17      **PMIX\_NODE\_INFO** "pmix.node.info" (bool)  
18      Return information about the specified node. If information about a node other than the one  
19      containing the requesting process is desired, then the attribute array must contain either the  
20      **PMIX\_NODEID** or **PMIX\_HOSTNAME** attribute identifying the desired target.

21      **PMIX\_PROCID** "pmix.proc\_id" (pmix\_proc\_t)  
22      Process identifier Specifies the process ID whose information is being requested - e.g., a  
23      query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Only required when the  
24      request is for information on a specific process.

25      **PMIX\_NSPACE** "pmix.nspace" (char\*)  
26      Namespace of the job - may be a numerical value expressed as a string, but is often a  
27      non-numerical string carrying information solely of use to the system. Specifies the  
28      namespace of the process whose information is being requested - e.g., a query asking for the  
29      **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_RANK**  
30      attribute. Only required when the request is for information on a specific process.

31      **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)  
32      Process rank within the job. Specifies the rank of the process whose information is being  
33      requested - e.g., a query asking for the **PMIX\_LOCAL\_RANK** of a specified process. Must  
34      be accompanied by the **PMIX\_NSPACE** attribute. Only required when the request is for  
35      information on a specific process.

36      **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** "pmix.qry.attrs" (bool)  
37      Query list of supported attributes for specified APIs. SUPPORTED QUALIFIERS:  
38      **PMIX\_CLIENT\_FUNCTIONS**, **PMIX\_SERVER\_FUNCTIONS**,  
39      **PMIX\_TOOL\_FUNCTIONS**, and/or **PMIX\_HOST\_FUNCTIONS**

```
1  PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
2      Request attributes supported by the PMIx client library
3
4  PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
5      Request attributes supported by the PMIx server library
6
7  PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
8      Request attributes supported by the host environment
9
10 PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
11     Request attributes supported by the PMIx tool library functions
12
13 Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the
14 PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a
15 process identifier must apply to all keys in that pmix_query_t. Queries for information on
16 multiple specific processes therefore requires submitting multiple pmix_query_t structures,
17 each referencing one process.
```

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

---

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

```
21  PMIX_PROCID "pmix.proc_id" (pmix_proc_t)
22      Process identifier Specifies the process ID whose information is being requested - e.g., a
23      query asking for the PMIX_LOCAL_RANK of a specified process. Only required when the
24      request is for information on a specific process.
25
26  PMIX_NSPACE "pmix.nspace" (char*)
27      Namespace of the job - may be a numerical value expressed as a string, but is often a
28      non-numerical string carrying information solely of use to the system. Specifies the
29      namespace of the process whose information is being requested - e.g., a query asking for the
30      PMIX_LOCAL_RANK of a specified process. Must be accompanied by the PMIX_RANK
31      attribute. Only required when the request is for information on a specific process.
32
33  PMIX_RANK "pmix.rank" (pmix_rank_t)
34      Process rank within the job. Specifies the rank of the process whose information is being
35      requested - e.g., a query asking for the PMIX_LOCAL_RANK of a specified process. Must
      be accompanied by the PMIX_NSPACE attribute. Only required when the request is for
      information on a specific process.
```

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

## Optional Attributes

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

7 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (**char\***)  
8 Request a comma-delimited list of active namespaces. NO QUALIFIERS

9 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (**pmix\_status\_t**)  
10 Status of a specified, currently executing job. REQUIRED QUALIFIER: **PMIX\_NSPACE**  
11 indicating the namespace whose status is being queried

12 **PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (**char\***)  
13 Request a comma-delimited list of scheduler queues. NO QUALIFIERS

14 **PMIX\_QUERY\_QUEUE\_STATUS** "pmix.qry.qst" (**char\***)  
15 Returns status of a specified scheduler queue, expressed as a string. SUPPORTED  
16 QUALIFIERS: **PMIX\_ALLOC\_QUEUE** naming specific queue whose status is being  
17 requested

18 **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.ptable" (**char\***)  
19 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
20 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:  
21 **PMIX\_NSPACE** indicating the namespace whose process table is being queried

22 **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lptable" (**char\***)  
23 Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for each  
24 process in the specified namespace executing on the same node as the requester, ordered by  
25 process job rank. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the namespace  
26 whose process table is being queried

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

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

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

35 **PMIX\_QUERY\_REPORT\_AVG** "pmix.qry.avg" (**bool**)

```
1 Report only average values for sampled information. NO QUALIFIERS
2 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
3 Report minimum and maximum values. NO QUALIFIERS
4 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
5 String identifier of the allocation whose status is being requested. NO QUALIFIERS
6 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
7 Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
8 SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
9 requested (defaults to allocation containing the caller)
10 PMIX_SERVER_URI "pmix.srvr.uri" (char*)
11 URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
12 PMIx connection. Defaults to requesting the information for the local PMIx server.
13 PMIX_PROC_URI "pmix.puri" (char*)
14 URI containing contact information for a given process. Requests the URI of the specified
15 PMIx server's out-of-band connection. Defaults to requesting the information for the local
16 PMIx server.
```

## 17 Description

18 Query information about the system in general. This can include a list of active namespaces, fabric  
19 topology, etc. Also can be used to query node-specific info such as the list of peers executing on a  
20 given node. We assume that the host RM will exercise appropriate access control on the  
21 information.

22 The returned *status* indicates if requested data was found or not. The returned array of  
23 **pmix\_info\_t** will contain each key that was provided and the corresponding value that was  
24 found. Requests for keys that are not found will return the key paired with a value of type  
25 **PMIX\_UNDEF**. The caller is responsible for releasing the returned array.

## Advice to PMIx library implementers

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

1 **7.1.4 PMIx\_Query\_info\_nb**

2      **Summary**

3      Query information about the system in general.

4      **Format**

5      PMIx v2.0

6      pmix\_status\_t  
7      PMIx\_Query\_info\_nb(pmix\_query\_t queries[], size\_t nqueries,  
8                         pmix\_info\_cfunc\_t cbfunc, void \*cbdata)

C

C

8      **IN queries**

9      Array of query structures (array of handles)

10     **IN nqueries**

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

12     **IN cbfunc**

13     Callback function **pmix\_info\_cfunc\_t** (function reference)

14     **IN cbdata**

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

16     Returns one of the following:

- 17     • **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided  
18        callback function will be executed upon completion of the operation. Note that the library must  
19        not invoke the callback function prior to returning from the API.
- 20     • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this  
21        case, the provided callback function will not be executed

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

- 24     • **PMIX\_SUCCESS** All data has been returned
- 25     • **PMIX\_ERR\_NOT\_FOUND** None of the requested data was available
- 26     • **PMIX\_ERR\_PARTIAL\_SUCCESS** Some of the data has been returned
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED** The host RM does not support this function
- 28     • a non-zero PMIx error constant indicating a reason for the request's failure

29     ----- Required Attributes -----

30     PMIx libraries that support this API are required to support the following attributes:

31     **PMIX\_QUERY\_REFRESH\_CACHE** "pmix.qry.rfsh" (bool)

32        Retrieve updated information from server. NO QUALIFIERS

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

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

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

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

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

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

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

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

21      **PMIX\_PROCID** "pmix.proc\_id" (pmix\_proc\_t)

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

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

26      Namespace of the job - may be a numerical value expressed as a string, but is often a  
27      non-numerical string carrying information solely of use to the system. Specifies the  
28      namespace of the process whose information is being requested - e.g., a query asking for the  
29      **PMIX\_LOCAL\_RANK** of a specified process. Must be accompanied by the **PMIX\_RANK**  
30      attribute. Only required when the request is for information on a specific process.

31      **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)

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

36      **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** "pmix.qry.attrs" (bool)

37      Query list of supported attributes for specified APIs. SUPPORTED QUALIFIERS:  
38      **PMIX\_CLIENT\_FUNCTIONS**, **PMIX\_SERVER\_FUNCTIONS**,  
39      **PMIX\_TOOL\_FUNCTIONS**, and/or **PMIX\_HOST\_FUNCTIONS**

```
1  PMIX_CLIENT_ATTRIBUTES "pmix.client.attrs" (bool)
2      Request attributes supported by the PMIx client library
3
4  PMIX_SERVER_ATTRIBUTES "pmix.srvr.attrs" (bool)
5      Request attributes supported by the PMIx server library
6
7  PMIX_HOST_ATTRIBUTES "pmix.host.attrs" (bool)
8      Request attributes supported by the host environment
9
10 PMIX_TOOL_ATTRIBUTES "pmix.setup.env" (bool)
11     Request attributes supported by the PMIx tool library functions
12
13 Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the
14 PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a
15 process identifier must apply to all keys in that pmix_query_t. Queries for information on
16 multiple specific processes therefore requires submitting multiple pmix_query_t structures,
17 each referencing one process.
```

```
14 PMIx libraries are not required to directly support any other attributes for this function. However,
15 any provided attributes must be passed to the host SMS daemon for processing, and the PMIx
16 library is required to add the PMIX_USERID and the PMIX_GRPID attributes of the client
17 process making the request.
```

---

```
18
19 Host environments that support this operation are required to support the following attributes as
20 qualifiers to the request:
```

```
21  PMIX_PROCID "pmix.proc_id" (pmix_proc_t)
22      Process identifier Specifies the process ID whose information is being requested - e.g., a
23      query asking for the PMIX_LOCAL_RANK of a specified process. Only required when the
24      request is for information on a specific process.
25
26  PMIX_NSPACE "pmix.nspace" (char*)
27      Namespace of the job - may be a numerical value expressed as a string, but is often a
28      non-numerical string carrying information solely of use to the system. Specifies the
29      namespace of the process whose information is being requested - e.g., a query asking for the
30      PMIX_LOCAL_RANK of a specified process. Must be accompanied by the PMIX_RANK
31      attribute. Only required when the request is for information on a specific process.
32
33  PMIX_RANK "pmix.rank" (pmix_rank_t)
34      Process rank within the job. Specifies the rank of the process whose information is being
35      requested - e.g., a query asking for the PMIX_LOCAL_RANK of a specified process. Must
36      be accompanied by the PMIX_NSPACE attribute. Only required when the request is for
37      information on a specific process.
38
39 Note that inclusion of the PMIX_PROCID directive and either the PMIX_NSPACE or the
40 PMIX_RANK attribute will return a PMIX_ERR_BAD_PARAM result, and that the inclusion of a
41 process identifier must apply to all keys in that pmix_query_t. Queries for information on
```

1 multiple specific processes therefore requires submitting multiple `pmix_query_t` structures,  
2 each referencing one process.

## Optional Attributes

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

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

```

1   PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
2     String identifier of the allocation whose status is being requested. NO QUALIFIERS
3
4   PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
5     Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
6     SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
7     requested (defaults to allocation containing the caller)
8
9   PMIX_SERVER_URI "pmix.srvr.uri" (char*)
10    URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's
11    PMIx connection. Defaults to requesting the information for the local PMIx server.
12
13   PMIX_PROC_URI "pmix.puri" (char*)
14     URI containing contact information for a given process. Requests the URI of the specified
15     PMIx server's out-of-band connection. Defaults to requesting the information for the local
16     PMIx server.

```



## 14      **Description**

15      Non-blocking form of the **PMIx\_Query\_info** API

### 16    **7.1.4.1 Using Get vs Query**

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

- 19      • information provided by the host environment at time of job start. This includes information on  
20        the number of processes in the job, their location, and possibly their communication endpoints
- 21      • information posted by processes via the **PMIx\_Put** function

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

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

- 28      • system-level information (such as the available WLM queues) that would generally not be  
29        included in job-level information provided at job start
- 30      • dynamic information such as application and queue status, and resource utilization statistics.  
31        Note that the **PMIX\_QUERY\_REFRESH\_CACHE** attribute must be provided on each query to  
32        ensure current data is returned
- 33      • information created post job start, such as process tables
- 34      • information requiring more complex search criteria than supported by the simpler **PMIx\_Get**  
35        API

- 1     • queries focused on retrieving multi-attribute blocks of data with a single request, thus bypassing  
2       the single-key limitation of the **PMIx\_Get** API

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

#### 9     **7.1.4.2 Accessing attribute support information**

10    Information as to attributes supported by either the PMIx implementation or its host environment  
11       can be obtained via the **PMIx\_Query\_info\_nb** API. The  
12       **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** attribute must be listed as the first entry in the *keys* field  
13       of the **pmix\_query\_t** structure, followed by the name of the function whose attribute support is  
14       being requested - support for multiple functions can be requested simultaneously by simply adding  
15       the function names to the array of *keys*. Function names *must* be given as user-level API names -  
16       e.g., “PMIx\_Get”, “PMIx\_server\_setup\_application”, or “PMIx\_tool\_connect\_to\_server”.

17    The desired levels (see [3.4.30](#)) of attribute support are provided as qualifiers. Multiple levels can be  
18       requested simultaneously by simply adding elements to the *qualifiers* array. Each qualifier should  
19       contain the desired level attribute with the boolean value set to indicate whether or not that level is  
20       to be included in the returned information. Failure to provide any levels is equivalent to a request  
21       for all levels.

22    Unlike other queries, queries for attribute support can result in the number of returned  
23       **pmix\_info\_t** structures being different from the number of queries. Each element in the  
24       returned array will correspond to a pair of specified attribute level and function in the query, where  
25       the *key* is the function and the *value* contains a **pmix\_data\_array\_t** of **pmix\_info\_t**.  
26       Each element of the array is marked by a *key* indicating the requested attribute *level* with a *value*  
27       composed of a **pmix\_data\_array\_t** of **pmix\_regattr\_t**, each describing a supported  
28       attribute for that function, as illustrated in Fig. 7.1 below where the requestor asked for supported  
29       attributes of **PMIx\_Get** at the *client* and *server* levels, plus attributes of  
30       **PMIx\_Allocation\_request** at all levels:

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

## 36    **7.2 Allocation Requests**

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

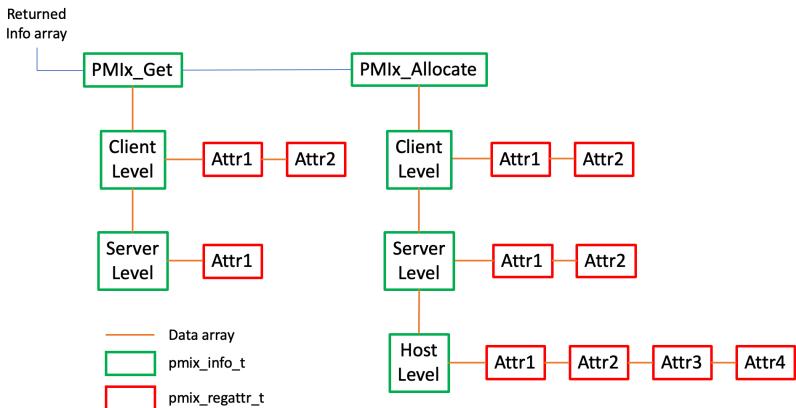


Figure 7.1.: Returned information hierarchy for attribute support request

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

## 7.2.1 PMIx\_Allocation\_request

### Summary

Request an allocation operation from the host resource manager.

### Format

PMIx v3.0

```
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 (handle)

IN **info**

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

```
1   IN  ninfo
2     Number of elements in the info array (integer)
3   INOUT results
4     Address where a pointer to an array of pmix_info_t containing the results of the request
5     can be returned (memory reference)
6   INOUT nresults
7     Address where the number of elements in results can be returned (handle)
8 Returns one of the following:
9   • PMIX_SUCCESS, indicating that the request was processed and returned success
10  • a PMIx error constant indicating either an error in the input or that the request was refused
```

### Required Attributes

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

---

15 Host environments that implement support for this operation are required to support the following  
16 attributes:

```
18   PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)
19     User-provided string identifier for this allocation request which can later be used to query
20     status of the request.

21   PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)
22     The number of nodes being requested in an allocation request

23   PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)
24     Number of cpus being requested in an allocation request.

25   PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
26     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 cpus for each node being requested in an allocation request.

**PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (**char\***)  
Regular expression of the specific cpus 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 allocation will be returned/stored as a **pmix\_data\_array\_t** of **pmix\_info\_t** indexed by this key and containing at least one entry with the same key and the allocated resource description. The type of the included value depends upon the fabric support. For example, a TCP allocation might consist of a comma-delimited string of socket ranges such as "32000-32100,33005,38123-38146". Additional entries will consist of any provided resource request directives, along with their assigned values. Examples include:  
**PMIX\_ALLOC\_FABRIC\_TYPE** - the type of resources provided;  
**PMIX\_ALLOC\_FABRIC\_PLANE** - if applicable, what plane the resources were assigned from; **PMIX\_ALLOC\_FABRIC\_QOS** - the assigned QoS; **PMIX\_ALLOC\_BANDWIDTH** - the allocated bandwidth; **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** - a security key for the requested fabric allocation. NOTE: the assigned values may differ from those requested, especially if **PMIX\_INFO\_REQD** was not set in the request.

**PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (**float**)  
Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (**char\***)  
Fabric quality of service level for the job being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (**char\***)  
Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

**PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (**char\***)

```
1 ID string for the NIC (aka plane) to be used for the requested allocation (e.g., CIDR for  
2 Ethernet)  
3 PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)  
4 Number of endpoints to allocate per process in the job  
5 PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)  
6 Number of endpoints to allocate per node for the job  
7 PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)  
8 Fabric security key for the spawned job
```



## 9 **Description**

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

- 12 Request allocation of additional resources, including memory, bandwidth, and compute. This  
13 should be accomplished in a non-blocking manner so that the application can continue to  
14 progress while waiting for resources to become available. Note that the new allocation will be  
15 disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one  
16 allocation will not impact the other.
- 17 Extend the reservation on currently allocated resources, subject to scheduling availability and  
18 priorities. This includes extending the time limit on current resources, and/or requesting  
19 additional resources be allocated to the requesting job. Any additional allocated resources will be  
20 considered as part of the current allocation, and thus will be released at the same time.
- 21 Return no-longer-required resources to the scheduler. This includes the “loan” of resources back  
22 to the scheduler with a promise to return them upon subsequent request.

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

## 26 **7.2.2 PMIx\_Allocation\_request\_nb**

### 27 **Summary**

28 Request an allocation operation from the host resource manager.

1      **Format**

2      *PMIx v2.0*

C

```
2      pmix_status_t  
3      PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,  
4                               pmix_info_t info[], size_t ninfo,  
5                               pmix_info_cfunc_t cfunc, void *cbdata);
```

C

6      **IN directive**

7      Allocation directive (handle)

8      **IN info**

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

10     **IN ninfo**

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

12     **IN cfunc**

13     Callback function **pmix\_info\_cfunc\_t** (function reference)

14     **IN cbdata**

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

16     Returns one of the following:

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

▼----- Required Attributes -----▼

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

28

---

29     Host environments that implement support for this operation are required to support the following  
30     attributes:

31     **PMIX\_ALLOC\_REQ\_ID** "pmix.alloc.reqid" (**char\***)

32       User-provided string identifier for this allocation request which can later be used to query  
33       status of the request.

34     **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (**uint64\_t**)

```
1      The number of nodes being requested in an allocation request  
2      PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)  
3          Number of cpus being requested in an allocation request.  
4      PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)  
5          Total session time (in seconds) being requested in an allocation request.
```

## Optional Attributes

```
6      The following attributes are optional for host environments that support this operation:  
7      PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)  
8          Regular expression of the specific nodes being requested in an allocation request  
9      PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)  
10         Regular expression of the number of cpus for each node being requested in an allocation  
11         request.  
12      PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)  
13         Regular expression of the specific cpus being requested in an allocation request.  
14      PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)  
15         Number of Megabytes[base2] of memory (per process) being requested in an allocation  
16         request.  
17      PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)  
18          Array of pmix_info_t describing requested fabric resources. This must include at least:  
19          PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and  
20          PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.  
21      PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)  
22          The key to be used when accessing this requested fabric allocation. The allocation will be  
23          returned/stored as a pmix_data_array_t of pmix_info_t indexed by this key and  
24          containing at least one entry with the same key and the allocated resource description. The  
25          type of the included value depends upon the fabric support. For example, a TCP allocation  
26          might consist of a comma-delimited string of socket ranges such as  
27          "32000-32100,33005,38123-38146". Additional entries will consist of any provided  
28          resource request directives, along with their assigned values. Examples include:  
29          PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;  
30          PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned  
31          from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -  
32          the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the  
33          requested fabric allocation. NOTE: the assigned values may differ from those requested,  
34          especially if PMIX_INFO_REQD was not set in the request.  
35      PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
```

1                   Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
2                   request.

3     **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (**char\***)  
4                   Fabric quality of service level for the job being requested in an allocation request.

5     **PMIX\_ALLOC\_FABRIC\_TYPE** "pmix.alloc.nettype" (**char\***)  
6                   Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.

7     **PMIX\_ALLOC\_FABRIC\_PLANE** "pmix.alloc.netplane" (**char\***)  
8                   ID string for the NIC (aka *plane*) to be used for the requested allocation (e.g., CIDR for  
9                   Ethernet)

10    **PMIX\_ALLOC\_FABRIC\_ENDPTS** "pmix.alloc.endpts" (**size\_t**)  
11                  Number of endpoints to allocate per process in the job

12    **PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE** "pmix.alloc.endpts.nd" (**size\_t**)  
13                  Number of endpoints to allocate per node for the job

14    **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** "pmix.alloc.nsec" (**pmix\_byte\_object\_t**)  
15                  Fabric security key for the spawned job

16                  

## Description

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

## 7.3 Job Control

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

29                  The job control APIs can also be used by an application to register itself as available for preemption  
30                  when operating in an environment such as a cloud or where incentives, financial or otherwise, are  
31                  provided to jobs willing to be preempted. Registration can include attributes indicating how many  
32                  resources are being offered for preemption (e.g., all or only some portion), whether the application  
33                  will require time to prepare for preemption, etc. Jobs that request a warning will receive an event  
34                  notifying them of an impending preemption (possibly including information as to the resources that  
35                  will be taken away, how much time the application will be given prior to being preempted, whether

1 the preemption will be a suspension or full termination, etc.) so they have an opportunity to save  
2 their work. Once the application is ready, it calls the provided event completion callback function to  
3 indicate that the SMS is free to suspend or terminate it, and can include directives regarding any  
4 desired restart.

### 5 7.3.1 PMIx\_Job\_control

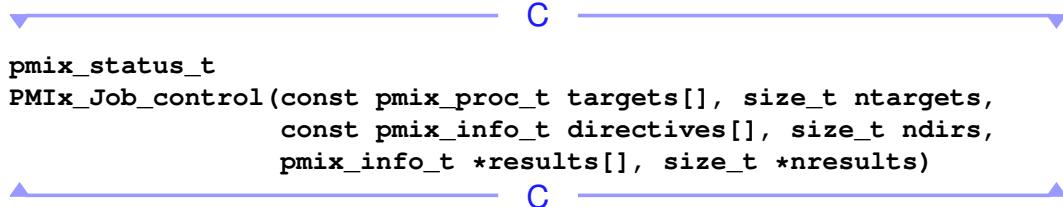
#### 6      **Summary**

7      Request a job control action.

#### 8      **Format**

9      *PMIx v3.0*

```
10     pmix_status_t
11     PMIx_Job_Control(const pmix_proc_t targets[], size_t ntargs,
12                    const pmix_info_t directives[], size_t ndirs,
13                    pmix_info_t *results[], size_t *nresults)
```



13      **IN    targets**  
14      Array of proc structures (array of handles)  
15      **IN    ntargs**  
16      Number of element in the *targets* array (integer)  
17      **IN    directives**  
18      Array of info structures (array of handles)  
19      **IN    ndirs**  
20      Number of element in the *directives* array (integer)  
21      **INOUT results**  
22      Address where a pointer to an array of **pmix\_info\_t** containing the results of the request  
23      can be returned (memory reference)  
24      **INOUT nresults**  
25      Address where the number of elements in *results* can be returned (handle)  
26      Returns one of the following:  
27      • **PMIX\_SUCCESS**, indicating that the request was processed by the host environment and  
28      returned *success*. Details of the result will be returned in the *results* array  
29      • a PMIx error constant indicating either an error in the input or that the request was refused

## Required Attributes

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

---

6 Host environments that implement support for this operation are required to support the following  
7 attributes:

8 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (**char\***)  
9     Provide a string identifier for this request. The user can provide an identifier for the  
10    requested operation, thus allowing them to later request status of the operation or to  
11    terminate it. The host, therefore, shall track it with the request for future reference.

12 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (**bool**)  
13     Pause the specified processes.

14 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (**bool**)  
15     Resume ("un-pause") the specified processes.

16 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (**bool**)  
17     Forcibly terminate the specified processes and cleanup.

18 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (**int**)  
19     Send given signal to specified processes.

20 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (**bool**)  
21     Politely terminate the specified processes.

22 **PMIX\_REGISTER\_CLEANUP** "pmix.reg.cleanup" (**char\***)  
23     Comma-delimited list of files to be removed upon process termination

24 **PMIX\_REGISTER\_CLEANUP\_DIR** "pmix.reg.cleanupdir" (**char\***)  
25     Comma-delimited list of directories to be removed upon process termination

26 **PMIX\_CLEANUP\_RECURSIVE** "pmix.clnup.recurse" (**bool**)  
27     Recursively cleanup all subdirectories under the specified one(s)

28 **PMIX\_CLEANUP\_EMPTY** "pmix.clnup.empty" (**bool**)  
29     Only remove empty subdirectories

30 **PMIX\_CLEANUP\_IGNORE** "pmix.clnup.ignore" (**char\***)  
31     Comma-delimited list of filenames that are not to be removed

32 **PMIX\_CLEANUP\_LEAVE\_TOPDIR** "pmix.clnup.lvtop" (**bool**)  
33     When recursively cleaning subdirectories, do not remove the top-level directory (the one  
34     given in the cleanup request)

## Optional Attributes

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

1           **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (**char\***)  
2           Cancel the specified request - the provided request ID must match the  
3           **PMIX\_JOB\_CTRL\_ID** provided to a previous call to **PMIx\_Job\_control**. An ID of  
4           **NULL** implies cancel all requests from this requestor.  
5

6           **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (**char\***)  
7           Restart the specified processes using the given checkpoint ID.  
8

9           **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (**char\***)  
10          Checkpoint the specified processes and assign the given ID to it.  
11

12          **PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT** "pmix.jctrl.ckptev" (**bool**)  
13          Use event notification to trigger a process checkpoint.  
14

15          **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (**int**)  
16          Use the given signal to trigger a process checkpoint.  
17

18          **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (**int**)  
19          Time in seconds to wait for a checkpoint to complete.  
20

21          **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
22          "pmix.jctrl.ckmethod" (**pmix\_data\_array\_t**)  
23          Array of **pmix\_info\_t** declaring each method and value supported by this application.  
24

25          **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (**char\***)  
26          Regular expression identifying nodes that are to be provisioned.  
27

28          **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvnimg" (**char\***)  
29          Name of the image that is to be provisioned.  
30

31          **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (**bool**)  
32          Indicate that the job can be pre-empted.

## Description

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

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

1 **7.3.2 PMIx\_Job\_control\_nb**

2      **Summary**

3      Request a job control action.

4      **Format**

PMIx v2.0

C

```
5      pmix_status_t
6      PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargs,
7                         const pmix_info_t directives[], size_t ndirs,
8                         pmix_info_cfunc_t cbfunc, void *cbdata)
```

9      **IN targets**

10     Array of proc structures (array of handles)

11      **IN ntargs**

12     Number of element in the *targets* array (integer)

13      **IN directives**

14     Array of info structures (array of handles)

15      **IN ndirs**

16     Number of element in the *directives* array (integer)

17      **IN cbfunc**

18     Callback function [pmix\\_info\\_cfunc\\_t](#) (function reference)

19      **IN cbdata**

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

21     Returns one of the following:

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

▼----- Required Attributes -----▼

29     PMIx libraries are not required to directly support any attributes for this function. However, any  
30     provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is  
31     *required* to add the [PMIX\\_USERID](#) and the [PMIX\\_GRPID](#) attributes of the client process making  
32     the request.

1 Host environments that implement support for this operation are required to support the following  
2 attributes:

3 **PMIX\_JOB\_CTRL\_ID** "pmix.jctrl.id" (char\*)  
4 Provide a string identifier for this request. The user can provide an identifier for the  
5 requested operation, thus allowing them to later request status of the operation or to  
6 terminate it. The host, therefore, shall track it with the request for future reference.

7 **PMIX\_JOB\_CTRL\_PAUSE** "pmix.jctrl.pause" (bool)  
8 Pause the specified processes.

9 **PMIX\_JOB\_CTRL\_RESUME** "pmix.jctrl.resume" (bool)  
10 Resume ("un-pause") the specified processes.

11 **PMIX\_JOB\_CTRL\_KILL** "pmix.jctrl.kill" (bool)  
12 Forcibly terminate the specified processes and cleanup.

13 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)  
14 Send given signal to specified processes.

15 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)  
16 Politely terminate the specified processes.

17 **PMIX\_REGISTER\_CLEANUP** "pmix.reg.cleanup" (char\*)  
18 Comma-delimited list of files to be removed upon process termination

19 **PMIX\_REGISTER\_CLEANUP\_DIR** "pmix.reg.cleanupdir" (char\*)  
20 Comma-delimited list of directories to be removed upon process termination

21 **PMIX\_CLEANUP\_RECURSIVE** "pmix.clnup.recurse" (bool)  
22 Recursively cleanup all subdirectories under the specified one(s)

23 **PMIX\_CLEANUP\_EMPTY** "pmix.clnup.empty" (bool)  
24 Only remove empty subdirectories

25 **PMIX\_CLEANUP\_IGNORE** "pmix.clnup.ignore" (char\*)  
26 Comma-delimited list of filenames that are not to be removed

27 **PMIX\_CLEANUP\_LEAVE\_TOPDIR** "pmix.clnup.lvtop" (bool)  
28 When recursively cleaning subdirectories, do not remove the top-level directory (the one  
29 given in the cleanup request)

## Optional Attributes

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

1           **PMIX\_JOB\_CTRL\_CANCEL** "pmix.jctrl.cancel" (**char\***)  
2           Cancel the specified request - the provided request ID must match the  
3           **PMIX\_JOB\_CTRL\_ID** provided to a previous call to **PMIx\_Job\_control**. An ID of  
4           **NULL** implies cancel all requests from this requestor.  
5

6           **PMIX\_JOB\_CTRL\_RESTART** "pmix.jctrl.restart" (**char\***)  
7           Restart the specified processes using the given checkpoint ID.  
8

9           **PMIX\_JOB\_CTRL\_CHECKPOINT** "pmix.jctrl.ckpt" (**char\***)  
10          Checkpoint the specified processes and assign the given ID to it.  
11          Use event notification to trigger a process checkpoint.  
12

13          **PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL** "pmix.jctrl.ckptsig" (**int**)  
14          Use the given signal to trigger a process checkpoint.  
15

16          **PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT** "pmix.jctrl.ckptsig" (**int**)  
17          Time in seconds to wait for a checkpoint to complete.  
18

19          **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
20          "pmix.jctrl.ckmethod" (**pmix\_data\_array\_t**)  
21          Array of **pmix\_info\_t** declaring each method and value supported by this application.  
22

23          **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (**char\***)  
24          Regular expression identifying nodes that are to be provisioned.  
25

26          **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvnimg" (**char\***)  
27          Name of the image that is to be provisioned.  
28

29          **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (**bool**)  
30          Indicate that the job can be pre-empted.

### Description

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

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

## 1 7.4 Process and Job Monitoring

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

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

### 14 7.4.1 PMIx\_Process\_monitor

#### 15 Summary

16 Request that application processes be monitored.

#### 17 Format

PMIx v3.0

C

```
18 pmix_status_t
19 PMIx_Process_monitor(const pmix_info_t *monitor, pmix_status_t error,
20                      const pmix_info_t directives[], size_t ndirs,
21                      pmix_info_t *results[], size_t *nresults)
```

22 IN monitor  
23 info (handle)

24 IN error  
25 status (integer)

26 IN directives  
27 Array of info structures (array of handles)

28 IN ndirs  
29 Number of elements in the *directives* array (integer)

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

33 INOUT nresults  
34 Address where the number of elements in *results* can be returned (handle)

35 Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request was processed and returned *success*. Details of the result will be returned in the *results* array
  - a PMIx error constant indicating either an error in the input or that the request was refused

## Optional Attributes

The following attributes may be implemented by a PMIx library or by the host environment. If supported by the PMIx server library, then the library must not pass the supported attributes to the host environment. All attributes not directly supported by the server library must be passed to the host environment if it supports this operation, and the library is *required* to add the **PMIX\_USERID** and the **PMIX\_GRPID** attributes of the requesting process:

**PMIX\_MONITOR\_ID** "pmix.monitor.id" (**char\***)  
 Provide a string identifier for this request.

**PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (**char\***)  
 Identifier to be canceled (**NULL** means cancel all monitoring for this process).

**PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (**bool**)  
 The application desires to control the response to a monitoring event.

**PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (**void**)  
 Register to have the PMIx server monitor the requestor for heartbeats.

**PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (**uint32\_t**)  
 Time in seconds before declaring heartbeat missed.

**PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (**uint32\_t**)  
 Number of heartbeats that can be missed before generating the event.

**PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (**char\***)  
 Register to monitor file for signs of life.

**PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (**bool**)  
 Monitor size of given file is growing to determine if the application is running.

**PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (**char\***)  
 Monitor time since last access of given file to determine if the application is running.

**PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (**char\***)  
 Monitor time since last modified of given file to determine if the application is running.

**PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (**uint32\_t**)  
 Time in seconds between checking the file.

**PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (**uint32\_t**)  
 Number of file checks that can be missed before generating the event.

1           **Description**

2           Request that application processes be monitored via several possible methods. For example, that  
3           the server monitor this process for periodic heartbeats as an indication that the process has not  
4           become “wedged”. When a monitor detects the specified alarm condition, it will generate an event  
5           notification using the provided error code and passing along any available relevant information. It  
6           is up to the caller to register a corresponding event handler.

7           The *monitor* argument is an attribute indicating the type of monitor being requested. For example,  
8           [PMIX\\_MONITOR\\_FILE](#) to indicate that the requestor is asking that a file be monitored.

9           The *error* argument is the status code to be used when generating an event notification alerting that  
10          the monitor has been triggered. The range of the notification defaults to  
11          [PMIX\\_RANGE\\_NAMESPACE](#). This can be changed by providing a [PMIX\\_RANGE](#) directive.

12          The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency  
13          of checking to be done

14         **7.4.2 PMIx\_Process\_monitor\_nb**

15           **Summary**

16           Request that application processes be monitored.

17           **Format**

18           *PMIx v2.0*

C

```
19           pmix_status_t
20           PMIx_Process_monitor_nb(const pmix_info_t *monitor, pmix_status_t error,
21                                    const pmix_info_t directives[], size_t ndirs,
22                                    pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

22           **IN monitor**  
23           info (handle)

24           **IN error**  
25           status (integer)

26           **IN directives**  
27           Array of info structures (array of handles)

28           **IN ndirs**  
29           Number of elements in the *directives* array (integer)

30           **IN cbfunc**  
31           Callback function [pmix\\_info\\_cbfunc\\_t](#) (function reference)

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

34           Returns one of the following:

- **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:

- 13     **PMIX\_MONITOR\_ID** "pmix.monitor.id" (**char\***)  
14         Provide a string identifier for this request.
- 15     **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (**char\***)  
16         Identifier to be canceled (**NULL** means cancel all monitoring for this process).
- 17     **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (**bool**)  
18         The application desires to control the response to a monitoring event.
- 19     **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (**void**)  
20         Register to have the PMIx server monitor the requestor for heartbeats.
- 21     **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (**uint32\_t**)  
22         Time in seconds before declaring heartbeat missed.
- 23     **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (**uint32\_t**)  
24         Number of heartbeats that can be missed before generating the event.
- 25     **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (**char\***)  
26         Register to monitor file for signs of life.
- 27     **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (**bool**)  
28         Monitor size of given file is growing to determine if the application is running.
- 29     **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (**char\***)  
30         Monitor time since last access of given file to determine if the application is running.
- 31     **PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (**char\***)  
32         Monitor time since last modified of given file to determine if the application is running.
- 33     **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitorftime" (**uint32\_t**)  
34         Time in seconds between checking the file.

1   **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)

2       Number of file checks that can be missed before generating the event.

3   **Description**

4   Non-blocking form of the [PMIx\\_Process\\_monitor](#) API. The *cbfunc* function provides a  
5   *status* to indicate whether or not the request was granted, and to provide some information as to the  
6   reason for any denial in the [pmix\\_info\\_cbfunc\\_t](#) array of [pmix\\_info\\_t](#) structures.

7   **7.4.3 PMIx\_Heartbeat**

8   **Summary**

9   Send a heartbeat to the PMIx server library

10   **Format**

PMIx v2.0

11   **PMIx\_Heartbeat (void)**

C

C

12   **Description**

13   A simplified macro wrapping [PMIx\\_Process\\_monitor\\_nb](#) that sends a heartbeat to the  
14   PMIx server library.

15   **7.5 Logging**

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

20   **7.5.1 PMIx\_Log**

21   **Summary**

22   Log data to a data service.

## Format

```
1 PMIx v3.0
2 pmix_status_t
3 PMIx_Log(const pmix_info_t data[], size_t ndata,
4           const pmix_info_t directives[], size_t ndirs)
```

```
5 IN  data
6   Array of info structures (array of handles)
7 IN  ndata
8   Number of elements in the data array (size_t)
9 IN  directives
10  Array of info structures (array of handles)
11 IN  ndirs
12  Number of elements in the directives array (size_t)
```

13 Return codes are one of the following:

```
14 PMIX_SUCCESS The logging request was successful.
15 PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry.
16 PMIX_ERR_NOT_SUPPORTED The PMIx implementation or host environment does not
17 support this function.
```

## Required Attributes

18 If the PMIx library does not itself perform this operation, then it is required to pass any attributes  
19 provided by the client to the host environment for processing. In addition, it must include the  
20 following attributes in the passed *info* array:

```
21 PMIX_USERID "pmix.euid" (uint32_t)
22   Effective user id.
23 PMIX_GRP_ID "pmix.egid" (uint32_t)
24   Effective group id.
```

---

25 Host environments or PMIx libraries that implement support for this operation are required to  
26 support the following attributes:

```
28 PMIX_LOG_STDERR "pmix.log.stderr" (char*)
29   Log string to stderr.
30 PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
31   Log string to stdout.
32 PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
33   Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,
34   otherwise to local syslog
```

```

1   PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)
2     Log data to local syslog. Defaults to ERROR priority.
3   PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)
4     Forward data to system “gateway” and log msg to that syslog. Defaults to ERROR priority.
5   PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)
6     Syslog priority level
7   PMIX_LOG_ONCE "pmix.log.once" (bool)
8     Only log this once with whichever channel can first support it, taking the channels in priority
9     order

```

**Optional Attributes**

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

```

12  PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
13    ID of source of the log request
14  PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
15    Timestamp for log report
16  PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)
17    Generate timestamp for log
18  PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
19    Label the output stream with the channel name (e.g., “stdout”)
20  PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
21    Print timestamp in output string
22  PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
23    Print the output stream in XML format
24  PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
25    Log via email based on pmix_info_t containing directives.
26  PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
27    Comma-delimited list of email addresses that are to receive the message.
28  PMIX_LOG_EMAIL SUBJECT "pmix.log.emsub" (char*)
29    Subject line for email.
30  PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
31    Message to be included in email.
32  PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
33    Log the provided information to the host environment’s job record

```

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

### 3 Description

4 Log data subject to the services offered by the host environment. The data to be logged is provided  
5 in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel.

#### Advice to users

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

## 11 7.5.2 PMIx\_Log\_nb

### 12 Summary

13 Log data to a data service.

### 14 Format

PMIx v2.0

C

```
15 pmix_status_t
16 PMIx_Log_nb(const pmix_info_t data[], size_t ndata,
17             const pmix_info_t directives[], size_t ndirs,
18             pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

#### 19 IN data

20 Array of info structures (array of handles)

#### 21 IN ndata

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

#### 23 IN directives

24 Array of info structures (array of handles)

#### 25 IN ndirs

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

#### 27 IN cbfunc

28 Callback function **pmix\_op\_cbfunc\_t** (function reference)

#### 29 IN cbdata

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

31 Return codes are one of the following:

```
1 PMIX_SUCCESS The logging request is valid and is being processed. The resulting status from  
2 the operation will be provided in the callback function. Note that the library must not invoke  
3 the callback function prior to returning from the API.  
4 PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and  
5 returned success - the cbfunc will not be called  
6 PMIX_ERR_BAD_PARAM The logging request contains at least one incorrect entry that prevents  
7 it from being processed. The callback function will not be called.  
8 PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function. The  
9 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:

```
13 PMIX_USERID "pmix.euid" (uint32_t)  
14 Effective user id.  
15 PMIX_GRP_ID "pmix.egid" (uint32_t)  
16 Effective group id.
```

---

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

```
20 PMIX_LOG_STDERR "pmix.log.stderr" (char*)  
21 Log string to stderr.  
22 PMIX_LOG_STDOUT "pmix.log.stdout" (char*)  
23 Log string to stdout.  
24 PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)  
25 Log data to syslog. Defaults to ERROR priority. Will log to global syslog if available,  
26 otherwise to local syslog  
27 PMIX_LOG_LOCAL_SYSLOG "pmix.log.lsys" (char*)  
28 Log data to local syslog. Defaults to ERROR priority.  
29 PMIX_LOG_GLOBAL_SYSLOG "pmix.log.gsys" (char*)  
30 Forward data to system “gateway” and log msg to that syslog. Defaults to ERROR priority.  
31 PMIX_LOG_SYSLOG_PRI "pmix.log.syspri" (int)  
32 Syslog priority level  
33 PMIX_LOG_ONCE "pmix.log.once" (bool)  
34 Only log this once with whichever channel can first support it, taking the channels in priority  
35 order
```

## Optional Attributes

1 The following attributes are optional for host environments or PMIx libraries that support this  
2 operation:

```
3 PMIX_LOG_SOURCE "pmix.log.source" (pmix_proc_t*)
4 ID of source of the log request
5 PMIX_LOG_TIMESTAMP "pmix.log.tstmp" (time_t)
6 Timestamp for log report
7 PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)
8 Generate timestamp for log
9 PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
10 Label the output stream with the channel name (e.g., "stdout")
11 PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
12 Print timestamp in output string
13 PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
14 Print the output stream in XML format
15 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
16 Log via email based on pmix_info_t containing directives.
17 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
18 Comma-delimited list of email addresses that are to receive the message.
19 PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)
20 Subject line for email.
21 PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)
22 Message to be included in email.
23 PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
24 Log the provided information to the host environment's job record
25 PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
26 Store the log data in a global data store (e.g., database)
```

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

---

Advice to users

---

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

---

## CHAPTER 8

# Event Notification

---

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

## 8.1 Notification and Management

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

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

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

### Advice to PMIx library implementers

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

## Advice to users

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

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

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

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

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

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

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

### Advice to users

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

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

## 8.1.1 PMIx\_Register\_event\_handler

### Summary

Register an event handler

### Format

PMIx v2.0

C

```
21 pmix_status_t
22 PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
23                                pmix_info_t info[], size_t ninfo,
24                                pmix_notification_fn_t evhdlr,
25                                pmix_evhdlr_reg_cbfunc_t cbfunc,
26                                void *cbdata);
```

C

#### IN codes

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

#### IN ncodes

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

#### IN info

Array of info structures (array of handles)

```
1   IN  ninfo
2     Number of elements in the info array (size_t)
3   IN  evhdlr
4     Event handler to be called pmix_notification_fn_t (function reference)
5   IN  cbfunc
6     Callback function pmix_evhdlr_reg_cbfunc_t (function reference)
7   IN  cbdata
8     Data to be passed to the cbfunc callback function (memory reference)
```

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:

```
14  PMIX_SUCCESS indicating that the request has been accepted for processing and the provided
15    callback function will be executed upon completion of the operation. Note that the library
16    must not invoke the callback function prior to returning from the API. The event handler
17    identifier will be returned in the callback
18  a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this
19    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:

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

26  PMIX_EVENT_HDLR_FIRST  "pmix.evfist" (bool)
27    Invoke this event handler before any other handlers.

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

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

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

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

36  PMIX_EVENT_HDLR_AFTER  "pmix.evafter" (char*)
```

1 Put this event handler immediately after the one specified in the **(char\*)** value.  
2 **PMIX\_EVENT\_HDLR\_PREPEND** "pmix.evprepend" (bool)  
3 Prepend this handler to the precedence list within its category.  
4 **PMIX\_EVENT\_HDLR\_APPEND** "pmix.evappend" (bool)  
5 Append this handler to the precedence list within its category.  
6 **PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrangle" (pmix\_data\_array\_t\*)  
7 Array of **pmix\_proc\_t** defining range of event notification.  
8 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)  
9 Value for calls to publish/lookup/unpublish or for monitoring event notifications.  
10 **PMIX\_EVENT\_RETURN\_OBJECT** "pmix.evobject" (void \*)  
11 Object to be returned whenever the registered callback function **cbfunc** is invoked. The  
12 object will only be returned to the process that registered it.

---

14 Host environments that implement support for PMIx event notification are required to support the  
15 following attributes:

16 **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (pmix\_proc\_t)  
17 The single process that was affected.  
18 **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaaffected" (pmix\_data\_array\_t\*)  
19 Array of **pmix\_proc\_t** defining affected processes.



### Optional Attributes

20 Host environments that support PMIx event notification *may* offer notifications for environmental  
21 events impacting the job and for SMS events relating to the job. The following attributes are  
22 optional for host environments that support this operation:

23 **PMIX\_EVENT\_TERMINATE\_SESSION** "pmix.evterm.sess" (bool)  
24 The RM intends to terminate this session.  
25 **PMIX\_EVENT\_TERMINATE\_JOB** "pmix.evterm.job" (bool)  
26 The RM intends to terminate this job.  
27 **PMIX\_EVENT\_TERMINATE\_NODE** "pmix.evterm.node" (bool)  
28 The RM intends to terminate all processes on this node.  
29 **PMIX\_EVENT\_TERMINATE\_PROC** "pmix.evterm.proc" (bool)  
30 The RM intends to terminate just this process.  
31 **PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evttimeout" (int)  
32 The time in seconds before the RM will execute error response.  
33 **PMIX\_EVENT\_SILENT\_TERMINATION** "pmix.evsilentterm" (bool)

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

2           **Description**

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

6            **Advice to users** 

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


10           **Advice to users** 

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


18          **8.1.2 PMIx\_Deregister\_event\_handler**

19           **Summary**

20           Deregister an event handler.

## Format

0

## IN exhd1r ref

Event handler ID returned by registration (**size** t)

## IN chfunc

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*).

### 32 8.1.3 PMIx Notify event

## Summary

Report an event for notification via any registered event handler.

1           **Format**2       *PMIx v2.0*

C

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

C

8       **IN**    **status**9       Status code of the event (**pmix\_status\_t**)10      **IN**    **source**11      Pointer to a **pmix\_proc\_t** identifying the original reporter of the event (handle)12      **IN**    **range**13      Range across which this notification shall be delivered (**pmix\_data\_range\_t**)14      **IN**    **info**15      Array of **pmix\_info\_t** structures containing any further info provided by the originator of  
16      the event (array of handles)17      **IN**    **ninfo**18      Number of elements in the *info* array (**size\_t**)19      **IN**    **cbfunc**20      Callback function to be executed upon completion of operation **pmix\_op\_cbfunc\_t**  
21      (function reference)22      **IN**    **cbdata**23      Data to be passed to the *cbfunc* callback function (memory reference)24      If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation  
25      returned in the status code.26      If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the  
27      following:

28      **PMIX\_SUCCESS** The notification request is valid and is being processed. The callback function  
29      will be called when the process-local operation is complete and will provide the resulting  
30      status of that operation. Note that this does *not* reflect the success or failure of delivering the  
31      event to any recipients. The callback function must not be executed prior to returning from the  
32      API.

33      **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
34      returned *success* - the *cbfunc* will *not* be called

35      **PMIX\_ERR\_BAD\_PARAM** The request contains at least one incorrect entry that prevents it from  
36      being processed. The callback function will *not* be called.

1           **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support event notification,  
2           or in the case of a PMIx server calling the API, the range extended beyond the local node and  
3           the host SMS environment does not support event notification. The callback function will *not*  
4           be called.

## Required Attributes

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

6           **PMIX\_EVENT\_NON\_DEFAULT** "pmix.evnonddef" (**bool**)  
7           Event is not to be delivered to default event handlers.

8           **PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrangle" (**pmix\_data\_array\_t\***)  
9           Array of **pmix\_proc\_t** defining range of event notification.

---

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

13          **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (**pmix\_proc\_t**)  
14          The single process that was affected.

15          **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaaffected" (**pmix\_data\_array\_t\***)  
16          Array of **pmix\_proc\_t** defining affected processes.

## Description

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

25          Host SMS daemons call the API to pass events down to its embedded PMIx server both for  
26          transmittal to local client processes and for the host's own internal processing where the host has  
27          registered its own event handlers. The PMIx server library is not allowed to echo any event given to  
28          it by its host via this API back to the host through the **pmix\_server\_notify\_event\_fn\_t**  
29          server module function. The host is required to deliver the event to all PMIx servers where the  
30          targeted processes either are currently running, or (if they haven't started yet) might be running at  
31          some point in the future as the events are required to be cached by the PMIx server library.

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



## Advice to users

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



## CHAPTER 9

# Data Packing and Unpacking

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

## 9.1 Data Buffer Type

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

PMIx v2.0

C

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

C

## 1 9.2 Support Macros

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

### 3 9.2.1 PMIX\_DATA\_BUFFER\_CREATE

#### 4 Summary

5 Allocate memory for a `pmix_data_buffer_t` object and initialize it

#### 6 Format

PMIx v2.0

```
7 PMIX_DATA_BUFFER_CREATE(buffer);
```



#### 8 OUT buffer

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

#### 10 Description

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

## 12 9.2.2 PMIX\_DATA\_BUFFER\_RELEASE

#### 13 Summary

14 Free a `pmix_data_buffer_t` object and the data it contains

#### 15 Format

PMIx v2.0

```
16 PMIX_DATA_BUFFER_RELEASE(buffer);
```



#### 17 IN buffer

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

#### 19 Description

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

## 21 9.2.3 PMIX\_DATA\_BUFFER\_CONSTRUCT

#### 22 Summary

23 Initialize a statically declared `pmix_data_buffer_t` object

1           **Format**  
2    PMIx v2.0      C  
3    PMIX\_DATA\_BUFFER\_CONSTRUCT(buffer);  
4            C  
5    **IN buffer**  
6        Pointer to the allocated `pmix_data_buffer_t` that is to be initialized (handle)  
**Description**  
Initialize a pre-allocated buffer object

## 7 9.2.4 PMIX\_DATA\_BUFFER\_DESTRUCT

8           **Summary**  
9       Release the data contained in a `pmix_data_buffer_t` object

10          **Format**  
11     PMIx v2.0      C  
12     PMIX\_DATA\_BUFFER\_DESTRUCT(buffer);  
13            C  
14    **IN buffer**  
15        Pointer to the `pmix_data_buffer_t` whose data is to be released (handle)  
**Description**  
Free's the data contained in a `pmix_data_buffer_t` object

## 16 9.2.5 PMIX\_DATA\_BUFFER\_LOAD

17           **Summary**  
18       Load a blob into a `pmix_data_buffer_t` object

19          **Format**  
20     PMIx v2.0      C  
21     PMIX\_DATA\_BUFFER\_LOAD(buffer, data, size);  
22            C  
23    **IN buffer**  
24        Pointer to a pre-allocated `pmix_data_buffer_t` (handle)  
25    **IN data**  
26        Pointer to a blob (`char*`)  
25    **IN size**  
26        Number of bytes in the blob `size_t`

1    **Description**

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

5    **9.2.6 PMIX\_DATA\_BUFFER\_UNLOAD**

6    **Summary**

7    Unload the data from a `pmix_data_buffer_t` object

8    **Format**

PMIx v2.0

9    `PMIX_DATA_BUFFER_UNLOAD(buffer, data, size);`

10    **IN buffer**

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

12    **OUT data**

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

14    **OUT size**

15      Variable to be assigned the number of bytes in the blob `size_t`

16    **Description**

17    Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to  
18    the provided variables, usually done to transmit the blob to a remote process for unpacking. The  
19    buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release -  
20    thus, the user is responsible for releasing the blob when done with it.

21    **9.3 General Routines**

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

23    **9.3.1 PMIx\_Data\_pack**

24    **Summary**

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

1           **Format**

2            *pmix\_status\_t*  
3            **PMIx\_Data\_pack**(const *pmix\_proc\_t* \**target*,  
4                           *pmix\_data\_buffer\_t* \**buffer*,  
5                           *void* \**src*, *int32\_t* *num\_vals*,  
6                           *pmix\_data\_type\_t* *type*);

7           **IN target**

8           Pointer to a *pmix\_proc\_t* containing the nspace/rank of the process that will be unpacking  
9           the final buffer. A NULL value may be used to indicate that the target is based on the same  
10          PMIx version as the caller. Note that only the target's nspace is relevant. (handle)

11          **IN buffer**

12          Pointer to a *pmix\_data\_buffer\_t* where the packed data is to be stored (handle)

13          **IN src**

14          Pointer to a location where the data resides. Strings are to be passed as (char \*\*)— i.e., the  
15          caller must pass the address of the pointer to the string as the (void\*). This allows the caller to  
16          pass multiple strings in a single call. (memory reference)

17          **IN num\_vals**

18          Number of elements pointed to by the *src* pointer. A string value is counted as a single value  
19          regardless of length. The values must be contiguous in memory. Arrays of pointers (e.g.,  
20          string arrays) should be contiguous, although the data pointed to need not be contiguous  
21          across array entries. (*int32\_t*)

22          **IN type**

23          The type of the data to be packed (*pmix\_data\_type\_t*)

24          Returns one of the following:

25           **PMIX\_SUCCESS** The data has been packed as requested

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

27           **PMIX\_ERR\_BAD\_PARAM** The provided buffer or src is **NULL**

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

30           **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation

31           **PMIX\_ERROR** General error

32          **Description**

33          The pack function packs one or more values of a specified type into the specified buffer. The buffer  
34          must have already been initialized via the **PMIX\_DATA\_BUFFER\_CREATE** or  
35          **PMIX\_DATA\_BUFFER\_CONSTRUCT** macros — otherwise, **PMIx\_Data\_pack** will return an  
36          error. Providing an unsupported type flag will likewise be reported as an error.

37          Note that any data to be packed that is not hard type cast (i.e., not type cast to a specific size) may  
38          lose precision when unpacked by a non-homogeneous recipient. The **PMIx\_Data\_pack** function

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

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

## 10 **9.3.2 PMIx\_Data\_unpack**

### 11 **Summary**

12 Unpack values from a [pmix\\_data\\_buffer\\_t](#)

### 13 **Format**

14 *PMIx v2.0*

15     [pmix\\_status\\_t](#)  
16     [PMIx\\_Data\\_unpack](#)([const pmix\\_proc\\_t](#) \*[source](#),  
17                         [pmix\\_data\\_buffer\\_t](#) \*[buffer](#), [void](#) \*[dest](#),  
18                         [int32\\_t](#) \*[max\\_num\\_values](#),  
19                         [pmix\\_data\\_type\\_t](#) [type](#));

#### 20 **IN source**

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

#### 24 **IN buffer**

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

#### 26 **INOUT dest**

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

#### 32 **INOUT max\_num\_values**

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

1           **IN type**  
2         The type of the data to be unpacked — must be one of the PMIx defined data types (   
3         `pmix_data_type_t` )  
4         Returns one of the following:  
5         **PMIX\_SUCCESS** The data has been unpacked as requested  
6         **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx implementation does not support this function.  
7         **PMIX\_ERR\_BAD\_PARAM** The provided buffer or dest is **NULL**  
8         **PMIX\_ERR\_UNKNOWN\_DATA\_TYPE** The specified data type is not known to this  
9             implementation  
10         **PMIX\_ERR\_OUT\_OF\_RESOURCE** Not enough memory to support the operation  
11         **PMIX\_ERROR** General error

## 12           **Description**

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

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

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

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

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

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

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

3 **9.3.3 PMIx\_Data\_copy**

4 **Summary**

5 Copy a data value from one location to another.

6 **Format**

PMIx v2.0

```
7     pmix_status_t
8     PMIx_Data_copy(void **dest, void *src,
9                      pmix_data_type_t type);
```

10 **IN dest**

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

12 **IN src**

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

14 **IN type**

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

16 **pmix\_data\_type\_t**)

17 Returns one of the following:

```
18     PMIX_SUCCESS The data has been copied as requested
19     PMIX_ERR_NOT_SUPPORTED The PMIx implementation does not support this function.
20     PMIX_ERR_BAD_PARAM The provided src or dest is NULL
21     PMIX_ERR_UNKNOWN_DATA_TYPE The specified data type is not known to this
22         implementation
23     PMIX_ERR_OUT_OF_RESOURCE Not enough memory to support the operation
24     PMIX_ERROR General error
```

25 **Description**

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

30 **9.3.4 PMIx\_Data\_print**

31 **Summary**

32 Pretty-print a data value.

1           **Format**

2        **pmix\_status\_t**  
3        **PMIx\_Data\_print**(**char \*\*output**, **char \*prefix**,  
4                    **void \*src**, **pmix\_data\_type\_t type**);

C

C

5           **IN output**

6           The address of a pointer into which the address of the resulting output is to be stored.  
7           (**char\*\***)

8           **IN prefix**

9           String to be prepended to the resulting output (**char\***)

10          **IN src**

11          A pointer to the memory location of the data value to be printed (handle)

12          **IN type**

13          The type of the data value to be printed — must be one of the PMIx defined data types. ( **pmix\_data\_type\_t** )

14  
15          Returns one of the following:

16           **PMIX\_SUCCESS** The data has been printed as requested

17           **PMIX\_ERR\_BAD\_PARAM** The provided data type is not recognized.

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

19          **Description**

20          Since registered data types can be complex structures, the system needs some way to know how to  
21          print them (i.e., convert them to a string representation). Primarily for debug purposes.

22     **9.3.5 PMIx\_Data\_copy\_payload**

23          **Summary**

24          Copy a payload from one buffer to another

25          **Format**

PMIx v2.0

```
1 pmix_status_t  
2 PMIx_Data_copy_payload(pmix_data_buffer_t *dest,  
3                         pmix_data_buffer_t *src);
```

4 **IN dest**

5 Pointer to the destination `pmix_data_buffer_t` (handle)

6 **IN src**

7 Pointer to the source `pmix_data_buffer_t` (handle)

8 Returns one of the following:

9 `PMIX_SUCCESS` The data has been copied as requested

10 `PMIX_ERR_BAD_PARAM` The src and dest `pmix_data_buffer_t` types do not match

11 `PMIX_ERR_NOT_SUPPORTED` The PMIx implementation does not support this function.

12 **Description**

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

## CHAPTER 10

# Security

---

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

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

12 Once the requesting client passes the initial test, the PMIx server can, at the choice of the  
13 implementor, perform additional security checks. This may involve a variety of methods such as  
14 exchange of a system-provided key or credential. At the conclusion of that process, the PMIx server  
15 reports the client connection request to the host via the  
16 **pmix\_server\_client\_connected\_fn\_t** interface. The host may then perform any  
17 additional checks and operations before responding with either **PMIX\_SUCCESS** to indicate that  
18 the connection is approved, or a PMIx error constant indicating that the connection request is  
19 refused. In this latter case, the PMIx server is required to drop the connection.

20 Tools started by the host environment are classed as a subgroup of client processes and follow the  
21 client process procedure. However, tools that are not started by the host environment must be  
22 handled differently as registration information is not available prior to the connection request. In  
23 these cases, the PMIx server library is required to use available standard OS methods to get the  
24 effective uid/gid and report them upwards as part of invoking the  
25 **pmix\_server\_tool\_connection\_fn\_t** interface, deferring initial security screening to  
26 the host. It is recognized that this may represent a security risk - for this reason, PMIx server  
27 libraries must not enable tool connections by default. Instead, the host has to explicitly enable them  
28 via the **PMIX\_SERVER\_TOOL\_SUPPORT** attribute, thus recognizing the associated risk. Once  
29 the host has completed its authentication procedure, it again informs the PMIx server of the result.

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

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

## 3 10.1 Obtaining Credentials

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

### 10 10.1.1 PMIx\_Get\_credential

#### 11 Summary

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

#### 13 Format

PMIx v3.0

```
14     pmix_status_t
15     PMIx_Get_credential(const pmix_info_t info[], size_t ninfo,
16                           pmix_byte_object_t *credential)
```

17 **IN info**  
18 Array of `pmix_info_t` structures (array of handles)  
19 **IN ninfo**  
20 Number of elements in the *info* array (`size_t`)  
21 **IN credential**  
22 Address of a `pmix_byte_object_t` within which to return credential (handle)

23 Returns one of the following:

- 24 • `PMIX_SUCCESS`, indicating that the credential has been returned in the provided  
25 `pmix_byte_object_t`
- 26 • a PMIx error constant indicating either an error in the input or that the request is unsupported

## Required Attributes

1 PMIx libraries that choose not to support this operation *must* return  
2 **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called.

3 There are no required attributes for this API. Note that implementations may choose to internally  
4 execute integration for some security environments (e.g., directly contacting a *munge* server).

5 Implementations that support the operation but cannot directly process the client's request must  
6 pass any attributes that are provided by the client to the host environment for processing. In  
7 addition, the following attributes are required to be included in the *info* array passed from the PMIx  
8 library to the host environment:

9     **PMIX\_USERID** "pmix.euid" (uint32\_t)  
10    Effective user id.

11    **PMIX\_GRP\_ID** "pmix.egid" (uint32\_t)  
12    Effective group id.

## Optional Attributes

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

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

## Advice to PMIx library implementers

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

### Description

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

## 10.1.2 **PMIx\_Get\_credential\_nb**

### Summary

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

1                   **Format**

2        **pmix\_status\_t**  
 3        **PMIx\_Get\_credential\_nb**(**const pmix\_info\_t info[]**, **size\_t ninfo**,  
 4                    **pmix\_credential\_cbfunc\_t cbfunc**, **void \*cbdata**)

5        **IN info**  
 6            Array of **pmix\_info\_t** structures (array of handles)  
 7        **IN ninfo**  
 8            Number of elements in the *info* array (**size\_t**)  
 9        **IN cbfunc**  
 10          Callback function to return credential (**pmix\_credential\_cbfunc\_t** function  
 11            reference)  
 12        **IN cbdata**  
 13          Data to be passed to the callback function (memory reference)

14          Returns one of the following:

- 15        • **PMIX\_SUCCESS** , indicating that the request has been communicated to the local PMIx server -  
 16            result will be returned in the provided *cbfunc*
- 17        • a PMIx error constant indicating either an error in the input or that the request is unsupported -  
 18            the *cbfunc* will *not* be called

19                   **Required Attributes**

20          PMIx libraries that choose not to support this operation *must* return  
**PMIX\_ERR\_NOT\_SUPPORTED** when the function is called.

21          There are no required attributes for this API. Note that implementations may choose to internally  
 22            execute integration for some security environments (e.g., directly contacting a *munge* server).

23          Implementations that support the operation but cannot directly process the client's request must  
 24            pass any attributes that are provided by the client to the host environment for processing. In  
 25            addition, the following attributes are required to be included in the *info* array passed from the PMIx  
 26            library to the host environment:

27        **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
 28            Effective user id.

29        **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
 30            Effective group id.

## Optional Attributes

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

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

## Advice to PMIx library implementers

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

### Description

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

## 14 10.2 Validating Credentials

15 The API for validating a credential is a non-blocking operation since the host environment may  
16 have to contact a remote credential service. Provision is made for the system to return additional  
17 information regarding possible authorization limitations beyond simple authentication.

### 18 10.2.1 **PMIx.Validate\_credential**

#### 19 Summary

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

1      **Format**

C

```
2      pmix_status_t  
3      PMIx_Validate_credential(const pmix_byte_object_t *cred,  
4                            const pmix_info_t info[], size_t ninfo,  
5                            pmix_info_t **results, size_t *nresults)
```

C

6      **IN cred**

7      Pointer to `pmix_byte_object_t` containing the credential (handle)

8      **IN info**

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

10     **IN ninfo**

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

12     **INOUT results**

13     Address where a pointer to an array of `pmix_info_t` containing the results of the request  
14     can be returned (memory reference)

15     **INOUT nresults**

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

17     Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request was processed and returned *success*. Details of the result will be returned in the *results* array
- a PMIx error constant indicating either an error in the input or that the request was refused

▼----- Required Attributes -----▼

21     PMIx libraries that choose not to support this operation *must* return

22     `PMIX_ERR_NOT_SUPPORTED` when the function is called.

23     There are no required attributes for this API. Note that implementations may choose to internally  
24     execute integration for some security environments (e.g., directly contacting a *munge* server).

25     Implementations that support the operation but cannot directly process the client's request must  
26     pass any attributes that are provided by the client to the host environment for processing. In  
27     addition, the following attributes are required to be included in the *info* array passed from the PMIx  
28     library to the host environment:

29     **PMIX\_USERID** "pmix.euid" (`uint32_t`)  
30       Effective user id.

31     **PMIX\_GRPID** "pmix.egid" (`uint32_t`)  
32       Effective group id.

## Optional Attributes

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

2 **PMIX\_TIMEOUT "pmix.timeout" (int)**

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

### Advice to PMIx library implementers

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

#### Description

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

### 14 **10.2.2 PMIx\_Validate\_credential\_nb**

#### 15 Summary

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

1           **Format**

2        *PMIx v3.0*

C

```
3        pmix_status_t
4        PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
5                                     const pmix_info_t info[], size_t ninfo,
6                                     pmix_validation_cbfunc_t cbfunc,
7                                     void *cbdata)
```

C

7        **IN****cred**

8           Pointer to **pmix\_byte\_object\_t** containing the credential (handle)

9        **IN****info**

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

11        **IN****ninfo**

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

13        **IN****cbfunc**

14          Callback function to return result (**pmix\_validation\_cbfunc\_t** function reference)

15        **IN****cbdata**

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

17          Returns one of the following:

- **PMIX\_SUCCESS**, indicating that the request has been communicated to the local PMIx server - result will be returned in the provided *cbfunc*
- a PMIx error constant indicating either an error in the input or that the request is unsupported - the *cbfunc* will *not* be called

▼----- **Required Attributes** -----▼

22        PMIx libraries that choose not to support this operation *must* return

23        **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called.

24        There are no required attributes for this API. Note that implementations may choose to internally  
25        execute integration for some security environments (e.g., directly contacting a *munge* server).

26        Implementations that support the operation but cannot directly process the client's request must  
27        pass any attributes that are provided by the client to the host environment for processing. In  
28        addition, the following attributes are required to be included in the *info* array passed from the PMIx  
29        library to the host environment:

30        **PMIX\_USERID** "pmix.euid" (**uint32\_t**)

31           Effective user id.

32        **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)

33           Effective group id.

## Optional Attributes

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

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

### Advice to PMIx library implementers

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

### Description

12 Request validation of a credential by the PMIx server library or the host environment.  
13

## CHAPTER 11

# Server-Specific Interfaces

---

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

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

## 11.1 Server-Specific Attributes

```
PMIX_TOPOLOGY "pmix.topo" (hwloc_topology_t)
    Provide a pointer to an HWLOC description of the local node topology.
PMIX_TOPOLOGY2 "pmix.topo2" (pmix_topology_t)
    Provide a pointer to an implementation-specific description of the local node topology.
```

## 11.2 Server Initialization and Finalization

Initialization and finalization routines for PMIx servers.

### 11.2.1 PMIx\_server\_init

#### Summary

Initialize the PMIx server.

## Format

C

```
INOUT module
    pmix_server_module_t structure (handle)
IN info
    Array of pmix_info_t structures (array of handles)
IN ninfo
    Number of elements in the info array (size_t)
```

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

## Required Attributes

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

**PMIX\_SERVER\_NSPACE** "pmix.srv.nspace" (char\*)  
Name of the namespace to use for this PMIX server.

**PMIX\_SERVER\_RANK** "pmix.srv.rank" (pmix\_rank\_t)  
Rank of this PMIx server

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

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

**PMIX\_SERVER\_TOOL\_SUPPORT** "pmix.srvr.tool" (bool)  
The host RM wants to declare itself as willing to accept tool connection requests.

**PMIX\_SERVER\_SYSTEM\_SUPPORT** "pmix.srvr.sys" (bool)  
The host RM wants to declare itself as being the local system server for PMIx connection requests.

## Optional Attributes

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

2   **PMIX\_USOCK\_DISABLE** "pmix.usock.disable" (bool)

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

5   **PMIX\_SOCKET\_MODE** "pmix.sockmode" (uint32\_t)

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

8   **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (char\*)

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

12   **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)

13       Comma-delimited list of devices and/or CIDR notation to include when establishing the  
14       TCP connection. If the library supports TCP socket connections, this attribute may be  
15       supported for specifying the interfaces to be used.

16   **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)

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

20   **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)

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

23   **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)

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

26   **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)

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

29   **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)

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

32   **PMIX\_SERVER\_REMOTE\_CONNECTIONS** "pmix.srvr.remote" (bool)

33       Allow connections from remote tools. Forces the PMIx server to not exclusively use  
34       loopback device. If the library supports connections from remote tools, this attribute may  
35       be supported for enabling or disabling it.

36   **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)

1           Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.

2   **PMIX\_TOPOLOGY2** "pmix.topo2" (**pmix\_topology\_t**)

3       Provide a pointer to an implementation-specific description of the local node topology. If

4       provided, the PMIx server will perform the necessary actions to scalably expose the

5       description to the local clients. This includes creating any required shared memory backing

6       stores and/ or XML representations, plus ensuring that all necessary key-value pairs for

7       clients to access the description are included in the job-level information provided to each

8       client.

## 9           **Description**

10          Initialize the PMIx server support library, and provide a pointer to a **pmix\_server\_module\_t**  
11          structure containing the caller's callback functions. The array of **pmix\_info\_t** structs is used to  
12          pass additional info that may be required by the server when initializing. For example, it may  
13          include the **PMIX\_SERVER\_TOOL\_SUPPORT** attribute, thereby indicating that the daemon is  
14          willing to accept connection requests from tools.

## ▼ Advice to PMIx server hosts ▼

15          Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module*  
16          structure. Doing so indicates that the host environment will not provide support for multi-node  
17          operations such as **PMIx\_Fence**, but does intend to support local clients access to information.

## 18   **11.2.2 PMIx\_server\_finalize**

### 19          **Summary**

20          Finalize the PMIx server library.

### 21          **Format**

22          *PMIx v1.0*

C

22          **pmix\_status\_t**  
23          **PMIx\_server\_finalize(void)**

C

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

### 25          **Description**

26          Finalize the PMIx server support library, terminating all connections to attached tools and any local  
27          clients. All memory usage is released.

---

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

## 11.3 Server Support Functions

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

### 11.3.1 PMIx\_generate\_regex

#### Summary

Generate a compressed representation of the input string.

#### Format

PMIx v1.0

C

```
pmix_status_t  
PMIx_generate_regex(const char *input, char **output)
```

C

#### IN input

String to process (string)

#### OUT output

Compressed representation of *input* (array of bytes)

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

#### Description

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

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

#### Advice to PMIx server hosts

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

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

---

## 1 11.3.2 PMIx\_generate\_ppn

### 2 Summary

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

### 4 Format

5 *PMIx v1.0*

6 `pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)`

#### 7 IN `input`

8 String to process (string)

#### 9 OUT `ppn`

10 Compressed representation of *input* (array of bytes)

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

### 12 Description

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

---

### Advice to PMIx server hosts

---

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

23 Communicating the resulting output should be done by first packing the returned expression using  
24 the `PMIx_Data_pack`, declaring the input to be of type `PMIX_REGEX`, and then obtaining the  
25 blob to be communicated using the `PMIX_DATA_BUFFER_UNLOAD` macro. The pack/unpack  
26 routines will ensure proper handling of the data based on the regex prefix.

---

## 27 11.3.3 PMIx\_server\_register\_nspace

### 28 Summary

29 Setup the data about a particular namespace.

1      **Format**

2      *pmix\_status\_t*  
3      **PMIx\_server\_register\_nspace**(const *pmix\_nspace\_t* *nspcse*,  
4                                    int *nlocalprocs*,  
5                                    *pmix\_info\_t* *info*[], size\_t *ninfo*,  
6                                    *pmix\_op\_cbfunc\_t* *cbfunc*, void \**cbdata*)

7      **IN nspace**

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

10     **IN nlocalprocs**

11     number of local processes (integer)

12     **IN info**

13     Array of info structures (array of handles)

14     **IN ninfo**

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

16     **IN cbfunc**

17     Callback function **pmix\_op\_cbfunc\_t** (function reference)

18     **IN cbdata**

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

20     Returns one of the following:

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

28     **Required Attributes**

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

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

31     Registration is for this namespace only, do not copy job data - this attribute is not accessed  
using the **PMIx\_Get**

---

1  
2 Host environments are required to provide a wide range of session-, job-, application-, node-, and  
3 process-level information, and may choose to provide a similarly wide range of optional  
4 information. The information is broadly separated into categories based on the *level* definitions  
5 explained in 3.4.11.

6 Session-level information may be passed as individual `pmix_info_t` entries, or as part of a  
7 `pmix_data_array_t` using the `PMIX_SESSION_INFO_ARRAY` attribute. Session-level  
8 information is always accessed using the namespace and the `PMIX_RANK_WILDCARD` rank,  
9 accompanied by the `PMIX_SESSION_INFO` attribute where ambiguity may exist. The list of data  
10 referenced in this way shall include:

- 11 • **PMIX\_UNIV\_SIZE** "pmix.univ.size" (`uint32_t`)  
12 Number of allocated slots in a session - each slot may or may not be occupied by an  
13 executing process. Note that this attribute is equivalent to providing the  
14 `PMIX_MAX_PROCS` entry in the `PMIX_SESSION_INFO_ARRAY` array - it is included  
15 in the Standard for historical reasons.
- 16 • **PMIX\_MAX\_PROCS** "pmix.max.size" (`uint32_t`)  
17 Maximum number of processes that can be executed in this context (session, namespace,  
18 application, or node). Typically, this is a constraint imposed by a scheduler or by user  
19 settings in a hostfile or other resource description. Must be provided if  
20 `PMIX_UNIV_SIZE` is not given. Requires use of the `PMIX_SESSION_INFO` attribute  
21 to avoid ambiguity when retrieving it.
- 22 • **PMIX\_SESSION\_ID** "pmix.session.id" (`uint32_t`)  
23 Session identifier assigned by the scheduler - referenced using `PMIX_RANK_WILDCARD`  
24 .

25 plus the following optional information:

- 26 • **PMIX\_CLUSTER\_ID** "pmix.clid" (`char*`)  
27 A string name for the cluster this allocation is on
- 28 • **PMIX\_ALLOCATED\_NODELIST** "pmix.alist" (`char*`)  
29 Comma-delimited list or regular expression of all nodes in this allocation regardless of  
30 whether or not they currently host processes - referenced using `PMIX_RANK_WILDCARD`  
31 .

32 Job-level information may be passed as individual `pmix_info_t` entries, or as part of a  
33 `pmix_data_array_t` using the `PMIX_JOB_INFO_ARRAY` attribute. The list of data  
34 referenced in this way shall include:

- 35 • **PMIX\_SERVER\_NSPACE** "pmix.srv.nspace" (`char*`)  
36 Name of the namespace to use for this PMIx server.
- 37 • **PMIX\_SERVER\_RANK** "pmix.srv.rank" (`pmix_rank_t`)  
38 Rank of this PMIx server

- **PMIX\_NSPACE** "pmix.namespace" (**char\***)  
Namespace of the job - may be a numerical value expressed as a string, but is often a non-numerical string carrying information solely of use to the system.
  - **PMIX\_JOBID** "pmix.jobid" (**char\***)  
Job identifier assigned by the scheduler - 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 this job across all contained applications. Note that this value can be different from **PMIX\_MAX\_PROCS**. For example, users may choose to subdivide an allocation (running several jobs in parallel within it), and dynamic programming models may support adding and removing processes from a running **job** on-the-fly. In the latter case, PMIx events must be used to notify processes within the job that the job size has changed.
  - **PMIX\_MAX\_PROCS** "pmix.max.size" (**uint32\_t**)  
Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. 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 - see [11.3.3.1](#) for an explanation of its generation.
  - **PMIX\_PROC\_MAP** "pmix.pmap" (**char\***)  
Regular expression describing processes on each node - see [11.3.3.1](#) for an explanation of its generation.
- plus the following optional information:
- **PMIX\_NPROC\_OFFSET** "pmix.offset" (**pmix\_rank\_t**)  
Starting global rank of this job - referenced using **PMIX\_RANK\_WILDCARD**.
  - **PMIX\_JOB\_NUM\_APPS** "pmix.job.napps" (**uint32\_t**)  
Number of applications in this 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
  - **PMIX\_RANKBY** "pmix.rankby" (**char\***)  
Process ranking policy - when accessed using **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace
  - **PMIX\_BINDTO** "pmix.bindto" (**char\***)

- 1                   Process binding policy - when accessed using **PMIx\_Get** , use the  
2                   **PMIX\_RANK\_WILDCARD** value for the rank to discover the binding policy used for the  
3                   provided namespace
- 4     • **PMIX\_HOSTNAME\_KEEP\_FQDN** "pmix.fqdn" (**bool**)  
5                   FQDN hostnames are being retained
- 6     • **PMIX\_ANL\_MAP** "pmix.anlmap" (**char\***)  
7                   Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
- 8                   Application-level information is accessed by providing both the namespace of the job (with the  
9                   rank set to **PMIX\_RANK\_WILDCARD** ). If application-level information is requested for an  
10                  application other than the one the caller belongs to, then the **PMIX\_APPNUM** attribute must be  
11                  provided. If more than one application is included in the namespace, then the host environment  
12                  is also required to supply data consisting of the following items for each application in the job,  
13                  passed as a **pmix\_data\_array\_t** using the **PMIX\_APP\_INFO\_ARRAY** attribute:
- 14     • **PMIX\_APPNUM** "pmix.appnum" (**uint32\_t**)  
15                   Application number within the job.
- 16     • **PMIX\_APP\_SIZE** "pmix.app.size" (**uint32\_t**)  
17                   Number of processes in this application.
- 18     • **PMIX\_MAX\_PROCS** "pmix.max.size" (**uint32\_t**)  
19                   Maximum number of processes that can be executed in this context (session, namespace,  
20                   application, or node). Typically, this is a constraint imposed by a scheduler or by user  
21                   settings in a hostfile or other resource description. Requires use of the  
22                   **PMIX\_APP\_INFO** attribute to avoid ambiguity when retrieving it.
- 23     • **PMIX\_APPLDR** "pmix.aldr" (**pmix\_rank\_t**)  
24                   Lowest rank in this application within this job - referenced using  
25                   **PMIX\_RANK\_WILDCARD**.
- 26     • **PMIX\_WDIR** "pmix.wdir" (**char\***)  
27                   Working directory for spawned processes. This attribute is required for all registrations,  
28                   but may be provided as an individual **pmix\_info\_t** entry if only one application is  
29                   included in the namespace.
- 30     • **PMIX\_APP\_ARGV** "pmix.app.argv" (**char\***)  
31                   Consolidated argv passed to the spawn command for the given process (e.g., "./myapp  
32                   arg1 arg2 arg3") This attribute is required for all registrations, but may be provided as an  
33                   individual **pmix\_info\_t** entry if only one application is included in the namespace.
- 34                   plus the following optional information:
- 35     • **PMIX\_PSET\_NAME** "pmix.pset.nm" (**char\***)  
36                   User-assigned name for the process set containing the given process.

1     The data may also include attributes provided by the host environment that identify the  
2     programming model (as specified by the user) being executed within the namespace. The PMIx  
3     server library may utilize this information to customize the environment to fit that model (e.g.,  
4     adding environmental variables specified by the corresponding standard for that model):

- 5     • **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (**char\***)  
6         Programming model being initialized (e.g., "MPI" or "OpenMP")
- 7     • **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (**char\***)  
8         Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
- 9     • **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (**char\***)  
10         Programming model version string (e.g., "2.1.1")

11     Node-level information is accessed by providing the namespace of the job with the rank set to  
12     **PMIX\_RANK\_WILDCARD**. If node-level information is requested for a node other than the one the  
13     caller is executing on, then the **PMIX\_NODEID** or the **PMIX\_HOSTNAME** attribute of the target  
14     node must be provided. Registration shall include the following data for each node in the job,  
15     passed as a **pmix\_data\_array\_t** using the **PMIX\_NODE\_INFO\_ARRAY** attribute:

- 16     • **PMIX\_NODEID** "pmix.nodeid" (**uint32\_t**)  
17         Node identifier expressed as the node's index (beginning at zero) in an array of nodes  
18         within the active session. The value must be unique and directly correlate to the  
19         **PMIX\_HOSTNAME** of the node - i.e., users can interchangeably reference the same  
20         location using either the **PMIX\_HOSTNAME** or corresponding **PMIX\_NODEID**.
- 21     • **PMIX\_HOSTNAME** "pmix.hname" (**char\***)  
22         Name of the host (e.g., where a specified process is running, or a given device is located).
- 23     • **PMIX\_HOSTNAME\_ALIASES** "pmix.alias" (**char\***)  
24         Comma-delimited list of names by which this node is known.
- 25     • **PMIX\_LOCAL\_SIZE** "pmix.local.size" (**uint32\_t**)  
26         Number of processes in this job or application on this node.
- 27     • **PMIX\_NODE\_SIZE** "pmix.node.size" (**uint32\_t**)  
28         Number of processes across all jobs on this node.
- 29     • **PMIX\_LOCALLDR** "pmix.lldr" (**pmix\_rank\_t**)  
30         Lowest rank on this node within this job - referenced using **PMIX\_RANK\_WILDCARD**.
- 31     • **PMIX\_LOCAL\_PEERS** "pmix.lpeers" (**char\***)  
32         Comma-delimited list of ranks on this node within the specified namespace - referenced  
33         using **PMIX\_RANK\_WILDCARD**.

34     plus the following information for the server's own node:

- 35     • **PMIX\_LOCAL\_CPUSETS** "pmix.lcpus" (**char\***)  
36         Colon-delimited cpusets of local peers within the specified namespace - referenced using  
37         **PMIX\_RANK\_WILDCARD**.

- **PMIX\_TMPDIR** "pmix.tmpdir" (**char\***)  
Full path to the top-level temporary directory assigned to the session.
- **PMIX\_NSDIR** "pmix.nsdir" (**char\***)  
Full path to the temporary directory assigned to the namespace, under **PMIX\_TMPDIR**.
- **PMIX\_LOCAL\_PROCS** "pmix.lprocs" (**pmix\_proc\_t array**)  
Array of **pmix\_proc\_t** of all processes on the specified node - referenced using **PMIX\_RANK\_WILDCARD**.

The data may also include one or more of the following methods for passing the HWLOC topology information for the server's own node:

- **PMIX\_HWLOC\_SHMEM\_FILE** "pmix.hwlocfile" (**char\***)  
Path to the HWLOC shared memory file.
- **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\_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.

plus the following optional information for the server's own node:

- **PMIX\_AVAIL\_PHYS\_MEMORY** "pmix.pmem" (**uint64\_t**)  
Total available physical memory on this node.

and the following optional information for arbitrary nodes:

- **PMIX\_MAX\_PROCS** "pmix.max.size" (**uint32\_t**)  
Maximum number of processes that can be executed in this context (session, namespace, application, or node). Typically, this is a constraint imposed by a scheduler or by user settings in a hostfile or other resource description. Requires use of the **PMIX\_NODE\_INFO** attribute to avoid ambiguity when retrieving it.

Process-level information is accessed by providing the namespace of the job and the rank of the target process. Registration 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.
- **PMIX\_APPNUM** "pmix.appnum" (**uint32\_t**)  
Application number within the job. This attribute may be omitted if only one application is present in the namespace.

- ```

1   • PMIX_APP_RANK "pmix.apprank" (pmix_rank_t)
2     Process rank within this application. This attribute may be omitted if only one
3     application is present in the namespace.
4   • PMIX_GLOBAL_RANK "pmix.grank" (pmix_rank_t)
5     Process rank spanning across all jobs in this session.
6   • PMIX_LOCAL_RANK "pmix.lrank" (uint16_t)
7     Local rank on this node within this job.
8   • PMIX_NODE_RANK "pmix.nrank" (uint16_t)
9     Process rank on this node spanning all jobs.
10  • PMIX_NODEID "pmix.nodeid" (uint32_t)
11    Node identifier expressed as the node's index (beginning at zero) in an array of nodes
12    within the active session. The value must be unique and directly correlate to the
13    PMIX_HOSTNAME of the node - i.e., users can interchangeably reference the same
14    location using either the PMIX_HOSTNAME or corresponding PMIX_NODEID.
15  • PMIX_REINCarnation "pmix.reinc" (uint32_t)
16    Number of times this process has been re-instantiated - i.e, a value of zero indicates that
17    the process has never failed and been restarted.
18  • PMIX_SPAWNED "pmix.spawned" (bool)
19    true if this process resulted from a call to PMIx_Spawn. Lack of inclusion (i.e., a
20    return status of PMIX_ERR_NOT_FOUND) corresponds to a value of false for this
21    attribute.
22 plus the following information for processes that are local to the server:
23  • PMIX_LOCALITY_STRING "pmix.locstr" (char*)
24    String describing a process's bound location - referenced using the process's rank. The
25    string is prefixed by the implementation that created it (e.g., "hwloc") followed by a colon.
26    The remainder of the string represents the corresponding locality as expressed by the
27    underlying implementation. The entire string must be passed to
28    PMIx_Get_relative_locality for processing. Note that hosts are only required
29    to provide locality strings for local client processes - thus, a call to PMIx_Get for the
30    locality string of a process that returns PMIX_ERR_NOT_FOUND indicates that the
31    process is not executing on the same node.
32  • PMIX_PROCDIR "pmix.pdir" (char*)
33    Full path to the subdirectory under PMIX_NSDIR assigned to the process.
34 and the following optional information - note that this information can be derived from information
35 already provided by other attributes, but it may be included here for ease of retrieval by users:
36  • PMIX_HOSTNAME "pmix.hname" (char*)
37    Name of the host (e.g., where a specified process is running, or a given device is located).

```

1  
2 Attributes not directly provided by the host environment may be derived by the PMIx server library  
3 from other required information and included in the data made available to the server library's  
4 clients.

5 **Description**  
6 Pass job-related information to the PMIx server library for distribution to local client processes.

---

**Advice to PMIx server hosts**

---

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

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

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

---

**Advice to users**

---

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

### 11.3.3.1 Assembling the registration information

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

#### Advice to PMIx server hosts

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

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

```
19 char **nodes = NULL;
20 char *nodelist;
21 char *regex;
22 size_t n;
23 pmix_status_t rc;
24 pmix_info_t info;
25
26 /* loop over an array of nodes, adding each
27  * name to the array */
28 for (n=0; n < num_nodes; n++)
29     /* filter the nodes to ignore those not included
30      * in the target range (session, job, etc.). In
31      * this example, all nodes are accepted */
32     PMIX_ARGV_APPEND(&nodes, node[n]->name);
33
34
35 /* join into a comma-delimited string */
36 nodelist = PMIX_ARGV_JOIN(nodes, ',');
```

```

1  /* release the array */
2  PMIX_ARGV_FREE(nodes);
3
4  /* generate regex */
5  rc = PMIx_generate_regex(nodelist, &regex);
6
7  /* release list */
8  free(nodelist);
9
10 /* pass the regex as the value to the PMIX_NODE_MAP key */
11 PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
12 /* release the regex */
13 free(regex);
14


```

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

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



```

20 char **ndppn;
21 char rank[30];
22 char **ppnarray = NULL;
23 char *ppn;
24 char *localranks;
25 char *regex;
26 size_t n, m;
27 pmix_status_t rc;
28 pmix_info_t info;
29
30 /* loop over an array of nodes */
31 for (n=0; n < num_nodes; n++)
32     /* for each node, construct an array of ranks on that node */
33     ndppn = NULL;
34     for (m=0; m < node[n]->num_procs; m++)
35         /* ignore processes that are not part of the target job */
36         if (!PMIX_CHECK_NSPACE(targetjob, node[n]->proc[m].nspace))
37             continue;
38
39         sprintf(rank, 30, "%d", node[n]->proc[m].rank);
40         PMIX_ARGV_APPEND(&ndppn, rank);

```

```

1      /* convert the array into a comma-delimited string of ranks */
2      localranks = PMIX_ARGV_JOIN(ndppn, ',');
3      /* release the local array */
4      PMIX_ARGV_FREE(ndppn);
5      /* add this node's contribution to the overall array */
6      PMIX_ARGV_APPEND(&ppnarray, localranks);
7      /* release the local list */
8      free(localranks);
9
10
11
12      /* join into a semicolon-delimited string */
13      ppn = PMIX_ARGV_JOIN(ppnarray, ';');
14
15      /* release the array */
16      PMIX_ARGV_FREE(ppnarray);
17
18      /* generate ppn regex */
19      rc = PMIx_generate_ppn(ppn, &regex);
20
21      /* release list */
22      free(ppn);
23
24      /* pass the regex as the value to the PMIX_PROC_MAP key */
25      PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
26      /* release the regex */
27      free(regex);
28

```

C

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**

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

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

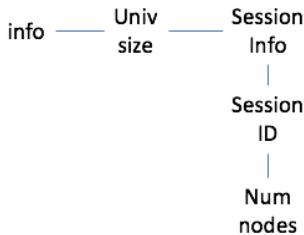


Figure 11.1.: Session-level information elements

- 5 • Job-level information includes all job-specific values such as **PMIX\_JOB\_SIZE**,  
6 **PMIX\_JOB\_NUM\_APPS**, and **PMIX\_JOBID**. Since each invocation of  
7 **PMIx\_server\_register\_nspace** describes a single **job**, job-specific values can be  
8 specified independently - i.e., in their own **pmix\_info\_t** elements of the *info* array.  
9 Alternatively, they can be provided as a **pmix\_data\_array\_t** array of **pmix\_info\_t**  
10 identified by the **PMIX\_JOB\_INFO\_ARRAY** attribute - this is required in cases where  
11 non-specific attributes (e.g., **PMIX\_NODE\_MAP**) are passed to describe aspects of the job. Note  
12 that since the invocation only involves a single namespace, there is no need to include the  
13 **PMIX\_NSPACE** attribute in the array.

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

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

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

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

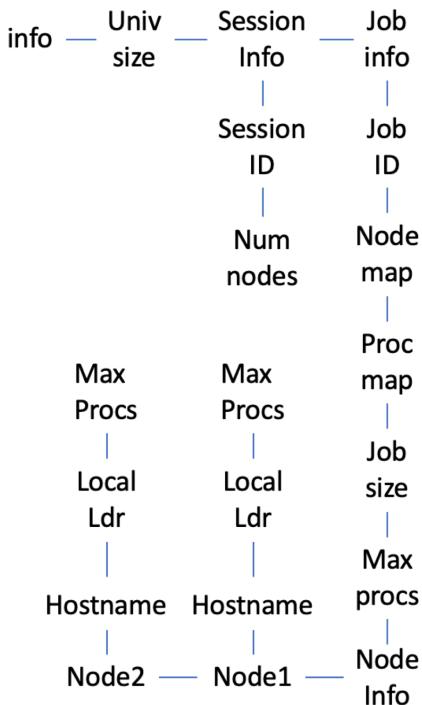


Figure 11.2.: Job-level information elements

elements of the *info* array - or as a `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_APP_INFO_ARRAY` attribute and identified by including the `PMIX_APPNUM` attribute in the array. Use of the array format is must in cases where non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the `PMIX_APP_INFO_ARRAY` format, each identified by its `PMIX_APPNUM` value.

Upon conclusion of this step, the *info* array might look like that shown in 11.3, assuming there are two applications in the job being registered:

- Process-level information includes an entry for each process in the job being registered, each entry marked with the `PMIX_PROC_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 11.4:
- For purposes of this example, node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases,

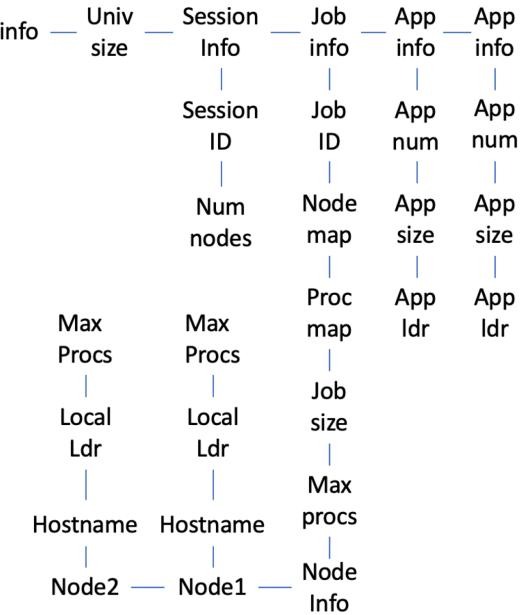


Figure 11.3.: Application-level information elements

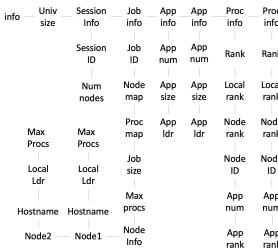


Figure 11.4.: Process-level information elements

the values included in this level are unique to it and can be specified independently - i.e., in their own `pmix_info_t` elements of the `info` array. Alternatively, they can be provided as a `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_NODE_INFO_ARRAY` attribute - this is required in cases where non-specific attributes are passed to describe aspects of the node, or where values for multiple nodes are being provided.

1 The node-level information requires two elements that must be constructed in a manner similar to  
2 that used for the node map. The **PMIX\_LOCAL\_PEERS** value is computed based on the  
3 processes on the local node, filtered to select those from the job being registered, as shown below  
4 using the tools provided by PMIx:

5     char \*\*ndppn = NULL;  
6     char rank[30];  
7     char \*localranks;  
8     size\_t m;  
9     pmix\_info\_t info;  
10  
11    for (m=0; m < mynode->num\_procs; m++)  
12      /\* ignore processes that are not part of the target job \*/  
13      if (!PMIX\_CHECK\_NSPACE(targetjob, mynode->proc[m].nspace))  
14         continue;  
15  
16      snprintf(rank, 30, "%d", mynode->proc[m].rank);  
17      PMIX\_ARGV\_APPEND(&ndppn, rank);  
18  
19      /\* convert the array into a comma-delimited string of ranks \*/  
20     localranks = PMIX\_ARGV\_JOIN(ndppn, ',');  
21      /\* release the local array \*/  
22     PMIX\_ARGV\_FREE(ndppn);  
23  
24      /\* pass the string as the value to the PMIX\_LOCAL\_PEERS key \*/  
25     PMIX\_INFO\_LOAD(&info, PMIX\_LOCAL\_PEERS, localranks, PMIX\_STRING);  
26      /\* release the list \*/  
27     free(localranks);  
28

29 The **PMIX\_LOCAL\_CPUSETS** value is constructed in a similar manner. In the provided  
30 example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a  
31 comma-delimited string of processor IDs) of the processors assigned to each process has  
32 previously been generated and stored on the process description. Thus, the value can be  
33 constructed as shown below:

```

1  char **ndcpus = NULL;
2  char *localcpus;
3  size_t m;
4  pmix_info_t info;
5
6  for (m=0; m < mynode->num_procs; m++)
7      /* ignore processes that are not part of the target job */
8      if (!PMIX_CHECK_NSPACE(targetjob,mynode->proc[m].nspace))
9          continue;
10
11     PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
12
13     /* convert the array into a colon-delimited string */
14     localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
15     /* release the local array */
16     PMIX_ARGV_FREE(ndcpus);
17
18     /* pass the string as the value to the PMIX_LOCAL_CPUETS key */
19     PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUETS, localcpus, PMIX_STRING);
20     /* release the list */
21     free(localcpus);
22

```

Note that for efficiency, these two values can be computed at the same time.

The final *info* array might therefore look like the diagram in [11.5](#):

### **11.3.4 PMIx\_server\_deregister\_nspace**

#### **Summary**

Deregister a namespace.

#### **Format**

*PMIx v1.0*

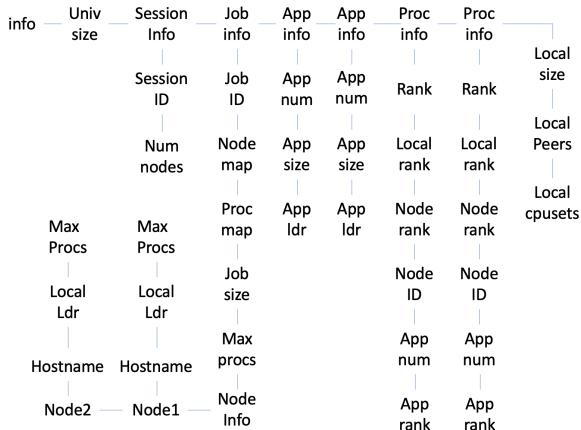


Figure 11.5.: Final information array

```

1 void PMIx_server_deregister_nspace(const pmix_nspace_t nspace,
2                                     pmix_op_cbfunc_t cbfunc, void *cbdata)

```

```

3 IN  nspace
4     Namespace (string)
5 IN  cbfunc
6     Callback function pmix_op_cbfunc_t (function reference)
7 IN  cbdata
8     Data to be passed to the callback function (memory reference)

```

### 9 Description

10 Deregister the specified *nspace* and purge all objects relating to it, including any client information  
11 from that namespace. This is intended to support persistent PMIx servers by providing an  
12 opportunity for the host RM to tell the PMIx server library to release all memory for a completed  
13 job. Note that the library must not invoke the callback function prior to returning from the API.

1 **11.3.5 PMIx\_server\_register\_client**

2 **Summary**

3 Register a client process with the PMIx server library.

4 **Format**

5 *PMIx v1.0*

C

```
6     pmix_status_t
7     PMIx_server_register_client(const pmix_proc_t *proc,
8                                     uid_t uid, gid_t gid,
9                                     void *server_object,
10                                    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

```
10    IN  proc
11        pmix_proc_t structure (handle)
12    IN  uid
13        user id (integer)
14    IN  gid
15        group id (integer)
16    IN  server_object
17        (memory reference)
18    IN  cbfunc
19        Callback function pmix_op_cbfunc_t (function reference)
20    IN  cbdata
21        Data to be passed to the callback function (memory reference)
```

22 Returns one of the following:

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

30 **Description**

31 Register a client process with the PMIx server library.

32 The host server can also, if it desires, provide an object it wishes to be returned when a server  
33 function is called that relates to a specific process. For example, the host server may have an object  
34 that tracks the specific client. Passing the object to the library allows the library to provide that  
35 object to the host server during subsequent calls related to that client, such as a

`pmix_server_client_connected_fn_t` function. This allows the host server to access the object without performing a lookup based on the client's namespace and rank.

## Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process. The expected user ID and group ID of the child process allows the server library to properly authenticate clients as they connect by requiring the two values to match. Accordingly, the detected user and group ID's of the connecting process are not included in the [`pmix\_server\_client\_connected\_fn\_t`](#) server module function.

## Advice to PMIx library implementers

For security purposes, the PMIx server library should check the user and group ID's of a connecting process against those provided for the declared client process identifier via the `PMIx_server_register_client` prior to completing the connection.

### 11.3.6 PMIx server deregister client

## Summary

Deregister a client and purge all data relating to it.

## Format

PMIx v1.0

0

**IN** proc  
pmix\_proc\_t structure (handle)

**IN** **cbfunc**  
Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)  
**IN** **cbddata**  
Data to be passed to the callback function (memory reference)

## Description

The `PMIx_server_deregister_nspace` API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library must not invoke the callback function prior to returning from the API.

### 11.3.7 PMIx\_server\_setup\_fork

## Summary

Setup the environment of a child process to be forked by the host.

## Format

*PMIx v1.0*

C

```
pmix_status_t  
PMIx_server_setup_fork(const pmix_proc_t *proc,  
                      char ***env)
```

**IN** `proc`  
`pmix_proc_t` structure (handle)

**IN** **env** Environment array (array of strings)

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

## Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server.

## Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process.

The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose, and will also provide any environmental variables that were specified in the launch command (e.g., via [PMIx\\_Spawn](#)) plus other values (e.g., variables required to properly initialize the client's fabric library).

## 11.3.8 PMIx\_server\_dmodex request

## Summary

Define a function by which the host server can request modeX data from the local PMIx server.

## Format

C

```
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 chfunc

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.

## 11.3.9 PMIx\_server\_setup\_application

### 2 Summary

3 Provide a function by which the resource manager can request application-specific setup data prior  
4 to launch of a **job**.

### 5 Format

PMIx v2.0

C

```
6 pmix_status_t
7 PMIx_server_setup_application(const pmix_nspace_t nspace,
8                               pmix_info_t info[], size_t ninfo,
9                               pmix_setup_application_cbfunc_t cbfunc,
10                             void *cbdata)
```

C

#### 11 IN nspace

12 namespace (string)

#### 13 IN info

14 Array of info structures (array of handles)

#### 15 IN ninfo

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

#### 17 IN cbfunc

18 Callback function **pmix\_setup\_application\_cbfunc\_t** (function reference)

#### 19 IN cbdata

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

21 Returns one of the following:

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

### Required Attributes

26 PMIx libraries that support this operation are required to support the following:

27 **PMIX\_SETUP\_APP\_ENVARS** "pmix.setup.env" (bool)

28 Harvest and include relevant environmental variables

29 **PMIX\_SETUP\_APP\_NONENVARS** ""pmix.setup.nenv" (bool)

30 Include all relevant data other than environmental variables

31 **PMIX\_SETUP\_APP\_ALL** "pmix.setup.all" (bool)

```

1      Include all relevant data
2      PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)
3          Array of pmix_info_t describing requested fabric resources. This must include at least:
4              PMIX_ALLOC_FABRIC_ID, PMIX_ALLOC_FABRIC_TYPE, and
5              PMIX_ALLOC_FABRIC_ENDPTS, plus whatever other descriptors are desired.
6      PMIX_ALLOC_FABRIC_ID "pmix.alloc.netid" (char*)
7          The key to be used when accessing this requested fabric allocation. The allocation will be
8          returned/stored as a pmix_data_array_t of pmix_info_t indexed by this key and
9          containing at least one entry with the same key and the allocated resource description. The
10         type of the included value depends upon the fabric support. For example, a TCP allocation
11         might consist of a comma-delimited string of socket ranges such as
12         "32000-32100,33005,38123-38146". Additional entries will consist of any provided
13         resource request directives, along with their assigned values. Examples include:
14             PMIX_ALLOC_FABRIC_TYPE - the type of resources provided;
15             PMIX_ALLOC_FABRIC_PLANE - if applicable, what plane the resources were assigned
16             from; PMIX_ALLOC_FABRIC_QOS - the assigned QoS; PMIX_ALLOC_BANDWIDTH -
17             the allocated bandwidth; PMIX_ALLOC_FABRIC_SEC_KEY - a security key for the
18             requested fabric allocation. NOTE: the assigned values may differ from those requested,
19             especially if PMIX_INFO REQD was not set in the request.
20      PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
21          Fabric security key for the spawned job
22      PMIX_ALLOC_FABRIC_TYPE "pmix.alloc.nettype" (char*)
23          Type of desired transport (e.g., "tcp", "udp") being requested in an allocation request.
24      PMIX_ALLOC_FABRIC_PLANE "pmix.alloc.netplane" (char*)
25          ID string for the NIC (aka plane) to be used for the requested allocation (e.g., CIDR for
26          Ethernet)
27      PMIX_ALLOC_FABRIC_ENDPTS "pmix.alloc.endpts" (size_t)
28          Number of endpoints to allocate per process in the job
29      PMIX_ALLOC_FABRIC_ENDPTS_NODE "pmix.alloc.endpts.nd" (size_t)
30          Number of endpoints to allocate per node for the job

```

### Optional Attributes

PMIx libraries that support this operation may support the following:

```

31
32      PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
33          Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation
34          request.
35      PMIX_ALLOC_FABRIC_QOS "pmix.alloc.netqos" (char*)

```

1           Fabric quality of service level for the job being requested in an allocation request.

2     **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (**uint32\_t**)

3        Total session time (in seconds) being requested in an allocation request.

4       The following optional attributes may be provided by the host environment to identify the  
5       programming model (as specified by the user) being executed within the application. The PMIx  
6       server library may utilize this information to harvest/forward model-specific environmental  
7       variables, record the programming model associated with the application, etc.

8     • **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (**char\***)

9       Programming model being initialized (e.g., "MPI" or "OpenMP")

10    • **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (**char\***)

11      Programming model implementation ID (e.g., "OpenMPI" or "MPICH")

12    • **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (**char\***)

13      Programming model version string (e.g., "2.1.1")

## 14       **Description**

15       Provide a function by which the RM can request application-specific setup data (e.g., environmental  
16       variables, fabric configuration and security credentials) from supporting PMIx server library  
17       subsystems prior to initiating launch of a job.

### Advice to PMIx server hosts

18       Host environments are required to execute this operation prior to launching a job. In addition to  
19       supported directives, the *info* array must include a description of the **job** using the  
20       **PMIX\_NODE\_MAP** and **PMIX\_PROC\_MAP** attributes.

21       This is defined as a non-blocking operation in case contributing subsystems need to perform some  
22       potentially time consuming action (e.g., query a remote service) before responding. The returned  
23       data must be distributed by the RM and subsequently delivered to the local PMIx server on each  
24       node where application processes will execute, prior to initiating execution of those processes.

### Advice to PMIx library implementers

25       Support for harvesting of environmental variables and providing of local configuration information  
26       by the PMIx implementation is optional.

## 27     **11.3.10 PMIx\_Register\_attributes**

### 28       **Summary**

29       Register host environment attribute support for a function.

## 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_reqattr_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 *attr* 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

1 PMIx implementations are *required* to register all internally supported attributes for each API  
2 during initialization of the library (i.e., when the process calls their respective PMIx init function).  
3 Specifically, the implementation *must not* register supported attributes upon first call to a given API  
4 as this would prevent users from discovering supported attributes prior to first use of an API.

5 It is the implementation's responsibility to associate registered attributes for a given  
6 `pmix_server_module_t` function with their corresponding user-facing API. Supported  
7 attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by  
8 the level (see 3.4.30) at which the attribute is supported.

9 Note that attributes can/will be registered on an API for each level. It is *required* that the  
10 implementation support user queries for supported attributes on a per-level basis. Duplicate  
11 registrations at the *same* level for a function *shall* return an error - however, duplicate registrations  
12 at *different* levels *shall* be independently tracked.

### 11.3.11 `PMIx_server_setup_local_support`

#### Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

#### Format

PMIx v2.0

```
18 pmix_status_t
19 PMIx_server_setup_local_support(const pmix_nspace_t nspace,
20                                     pmix_info_t info[], size_t ninfo,
21                                     pmix_op_cbfunc_t cbfunc,
22                                     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` (function reference)  
IN `cbdata`  
Data to be passed to the callback function (memory reference)

1 Returns one of the following:

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

9 **Description**

10 Provide a function by which the local PMIx server can perform any application-specific operations  
11 prior to spawning local clients of a given application. For example, a fabric library might need to  
12 setup the local driver for “instant on” addressing. The data provided in the *info* array is the data  
13 returned to the host RM by the callback function executed as a result of a call to  
14 **PMIx\_server\_setup\_application**.

---

Advice to PMIx server hosts

---

15 Host environments are required to execute this operation prior to starting any local application  
16 processes from the specified namespace.

17 **11.3.12 PMIx\_server\_IOF\_deliver**

18 **Summary**

19 Provide a function by which the host environment can pass forwarded IO to the PMIx server library  
20 for distribution to its clients.

21 **Format**

PMIx v3.0

C

```
22 pmix_status_t
23 PMIx_server_IOF_deliver(const pmix_proc_t *source,
24                           pmix_iof_channel_t channel,
25                           const pmix_byte_object_t *bo,
26                           const pmix_info_t info[], size_t ninfo,
27                           pmix_op_cbfunc_t cbfunc, void *cbdata);
```

```
1   IN  source
2     Pointer to pmix_proc_t identifying source of the IO (handle)
3   IN  channel
4     IO channel of the data ( pmix_ifof_channel_t )
5   IN  bo
6     Pointer to pmix_byte_object_t containing the payload to be delivered (handle)
7   IN  info
8     Array of pmix_info_t metadata describing the data (array of handles)
9   IN  ninfo
10    Number of elements in the info array (size_t)
11   IN  cbfunc
12     Callback function pmix_op_cbfunc_t (function reference)
13   IN  cbdata
14     Data to be passed to the callback function (memory reference)

15 Returns one of the following:
16
17   • PMIX_SUCCESS , indicating that the request is being processed by the host environment - result
18     will be returned in the provided cbfunc. Note that the library must not invoke the callback
19     function prior to returning from the API.
20
21   • PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and
22     returned success - the cbfunc will not be called
23
24   • a PMIx error constant indicating either an error in the input or that the request was immediately
25     processed and failed - the cbfunc will not be called
```

### 23 **Description**

24 Provide a function by which the host environment can pass forwarded IO to the PMIx server library
25 for distribution to its clients. The PMIx server library is responsible for determining which of its
26 clients have actually registered for the provided data and delivering it. The *cbfunc* callback function
27 will be called once the PMIx server library no longer requires access to the provided data.

## 28 **11.3.13 PMIx\_server\_collect\_inventory**

### 29 **Summary**

30 Collect inventory of resources on a node

1      **Format**

2      *PMIx v3.0*      C  
3      pmix\_status\_t  
4      PMIx\_server\_collect\_inventory(const pmix\_info\_t directives[],  
5                            size\_t ndirs,  
6                            pmix\_info\_cbfunc\_t cbfunc,  
                             void \*cbdata);

7      **IN directives**

8      Array of `pmix_info_t` directing the request (array of handles)

9      **IN ndirs**

10     Number of elements in the *directives* array (`size_t`)

11     **IN cbfunc**

12     Callback function to return collected data (`pmix_info_cbfunc_t` function reference)

13     **IN cbdata**

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

15     Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant. In the event  
16     the function returns an error, the *cbfunc* will not be called.

17     **Description**

18     Provide a function by which the host environment can request its PMIx server library collect an  
19     inventory of local resources. Supported resources depends upon the PMIx implementation, but may  
20     include the local node topology and fabric interfaces.

17     **Advice to PMIx server hosts**

21     This is a non-blocking API as it may involve somewhat lengthy operations to obtain the requested  
22     information. Inventory collection is expected to be a rare event – at system startup and upon  
23     command from a system administrator. Inventory updates are expected to initiate a smaller  
24     operation involving only the changed information. For example, replacement of a node would  
25     generate an event to notify the scheduler with an inventory update without invoking a global  
26     inventory operation.

27     **11.3.14 PMIx\_server\_deliver\_inventory**

28     **Summary**

29     Pass collected inventory to the PMIx server library for storage

## Format

```
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_cfunc_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` (function reference)

IN cbdata

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

Returns one of the following:

#### • **PMTX SUCCESS** indicates

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

## Description

Provide a function by which the host environment can pass inventory information obtained from a node to the PMIx server library for storage. Inventory data is subsequently used by the PMIx server library for allocations in response to [PMIx\\_server\\_setup\\_application](#), and may be available to the library's host via the [PMIx\\_Get](#) API (depending upon PMIx implementation). The *cbfunc* callback function will be called once the PMIx server library no longer requires access to the provided data.

## 11.3.15 PMIx\_generate\_locality\_string

### Summary

Generate a PMIx locality string from a given cpuset

### Format

PMIx v4.0

C

```
5     pmix_status_t  
6     PMIx_generate_locality_string(const pmix_cpuset_t *cpuset,  
7                                     char **locality);
```

C

#### 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.

## 11.3.15.1 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`).

PMIx v4.0

C

```
28     typedef struct pmix_cpuset {  
29         char *source;  
30         void *bitmap;  
31     } pmix_cpuset_t;
```

C

## 11.3.16 Cpuset support macros

The following macros support the `pmix_cpuset_t` structure.

### 11.3.16.1 Initialize the cpuset structure

Initialize the `pmix_cpuset_t` fields

PMIx v4.0

C

`PMIX_CPUSET_CONSTRUCT(m)`

C

IN m

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

## 11.4 Server Function Pointers

PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the protocol. This method allows RMs to implement the server without being burdened with PMIx internal details. When a request is received from the client, the corresponding server function will be called with the information.

Any functions not supported by the RM can be indicated by a `NULL` for the function pointer. PMIx implementations are required to return a `PMIX_ERR_NOT_SUPPORTED` status to all calls to functions that require host environment support and are not backed by a corresponding server module entry.

The host RM will provide the function pointers in a `pmix_server_module_t` structure passed to `PMIx_server_init`. That module structure and associated function references are defined in this section.

### Advice to PMIx server hosts

For performance purposes, the host server is required to return as quickly as possible from all functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx server support library to handle multiple client requests as quickly and scalably as possible.

All data passed to the host server functions is "owned" by the PMIx server support library and must not be free'd. Data returned by the host server via callback function is owned by the host server, which is free to release it upon return from the callback

### 11.4.1 `pmix_server_module_t` Module

#### Summary

List of function pointers that a PMIx server passes to `PMIx_server_init` during startup.

## 1 Format

C

```
2     typedef struct pmix_server_module_3_0_0_t
3         /* v1x interfaces */
4             pmix_server_client_connected_fn_t    client_connected;
5             pmix_server_client_finalized_fn_t   client_finalized;
6             pmix_server_abort_fn_t            abort;
7             pmix_server_fencenb_fn_t          fence_nb;
8             pmix_server_dmodex_req_fn_t      direct_modex;
9             pmix_server_publish_fn_t         publish;
10            pmix_server_lookup_fn_t         lookup;
11            pmix_server_unpublish_fn_t      unpublish;
12            pmix_server_spawn_fn_t          spawn;
13            pmix_server_connect_fn_t        connect;
14            pmix_server_disconnect_fn_t     disconnect;
15            pmix_server_register_events_fn_t register_events;
16            pmix_server_deregister_events_fn_t deregister_events;
17            pmix_server_listener_fn_t       listener;
18            /* v2x interfaces */
19            pmix_server_notify_event_fn_t    notify_event;
20            pmix_server_query_fn_t          query;
21            pmix_server_tool_connection_fn_t tool_connected;
22            pmix_server_log_fn_t           log;
23            pmix_server_alloc_fn_t          allocate;
24            pmix_server_job_control_fn_t    job_control;
25            pmix_server_monitor_fn_t        monitor;
26            /* v3x interfaces */
27            pmix_server_get_cred_fn_t      get_credential;
28            pmix_server_validate_cred_fn_t validate_credential;
29            pmix_server_iоф_fn_t           iоф_pull;
30            pmix_server_stdin_fn_t         push_stdin;
31            /* v4x interfaces */
32            pmix_server_grp_fn_t           group;
33            pmix_server_fabric_fn_t        fabric;
34     pmix_server_module_t;
```

C

35 **11.4.2 pmix\_server\_client\_connected\_fn\_t**36 **Summary**

37 Notify the host server that a client connected to this server.

1      **Format**

2      *PMIx v1.0*

C

```
2      typedef pmix_status_t (*pmix_server_client_connected_fn_t) (
3               const pmix_proc_t *proc,
4               void* server_object,
5               pmix_op_cbfunc_t cbfunc,
6               void *cbdata)
```

C

7      **IN proc**  
8          *pmix\_proc\_t* structure (handle)  
9      **IN server\_object**  
10     object reference (memory reference)  
11      **IN cbfunc**  
12     Callback function *pmix\_op\_cbfunc\_t* (function reference)  
13      **IN cbdata**  
14     Data to be passed to the callback function (memory reference)

15     Returns one of the following:

- **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

23     **Description**

24     Notify the host environment that a client has called **PMIx\_Init** . Note that the client will be in a  
25     blocked state until the host server executes the callback function, thus allowing the PMIx server  
26     support library to release the client. The *server\_object* parameter will be the value of the  
27     *server\_object* parameter passed to **PMIx\_server\_register\_client** by the host server  
28     when registering the connecting client. If provided, an implementation of  
29     **pmix\_server\_client\_connected\_fn\_t** is only required to call the callback function  
30     designated. A host server can choose to not be notified when clients connect by setting  
31     **pmix\_server\_client\_connected\_fn\_t** to **NULL**.

32     It is possible that only a subset of the clients in a namespace call **PMIx\_Init** . The server's  
33     **pmix\_server\_client\_connected\_fn\_t** implementation should not depend on being  
34     called once per rank in a namespace or delay calling the callback function until all ranks have  
35     connected. However, if a rank makes any PMIx calls, it must first call **PMIx\_Init** and therefore  
36     the server's **pmix\_server\_client\_connected\_fn\_t** will be called before any other  
37     server functions specific to the rank.

## Advice to PMIx server hosts

This operation is an opportunity for a host environment to update the status of the ranks it manages. It is also a convenient and well defined time to perform initialization necessary to support further calls into the server related to that rank.

### 11.4.3 pmix\_server\_client\_finalized\_fn\_t

#### Summary

Notify the host environment that a client called `PMIx_Finalize`.

#### Format

PMIx v1.0

C

```
8     typedef pmix_status_t (*pmix_server_client_finalized_fn_t)(
9             const pmix_proc_t *proc,
10            void* server_object,
11            pmix_op_cbfunc_t cbfunc,
12            void *cbdata)
```

C

13 **IN proc**  
14 `pmix_proc_t` structure (handle)  
15 **IN server\_object**  
16 object reference (memory reference)  
17 **IN cbfunc**  
18 Callback function `pmix_op_cbfunc_t` (function reference)  
19 **IN cbdata**  
20 Data to be passed to the callback function (memory reference)

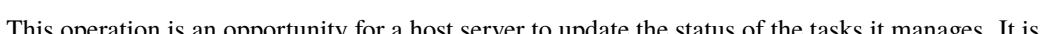
21 Returns one of the following:

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

1           **Description**

2       Notify the host environment that a client called **PMIx\_Finalize**. Note that the client will be in  
3       a blocked state until the host server executes the callback function, thus allowing the PMIx server  
4       support library to release the client. The `server_object` parameter will be the value of the  
5       `server_object` parameter passed to **PMIx\_server\_register\_client** by the host server  
6       when registering the connecting client. If provided, an implementation of  
7       **pmix\_server\_client\_finalized\_fn\_t** is only required to call the callback function  
8       designated. A host server can choose to not be notified when clients finalize by setting  
9       **pmix\_server\_client\_finalized\_fn\_t** to **NULL**.

10      Note that the host server is only being informed that the client has called **PMIx\_Finalize**. The  
11     client might not have exited. If a client exits without calling **PMIx\_Finalize**, the server support  
12     library will not call the **pmix\_server\_client\_finalized\_fn\_t** implementation.

13            **Advice to PMIx server hosts** 

14      This operation is an opportunity for a host server to update the status of the tasks it manages. It is  
also a convenient and well defined time to release resources used to support that client.  


15   **11.4.4 pmix\_server\_abort\_fn\_t**

16           **Summary**

17      Notify the host environment that a local client called **PMIx\_Abort**.

18           **Format**

19       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_cfunc_t cbfunc,
    void *cbdata)
```

```

1   IN  proc
2     pmix_proc_t structure identifying the process requesting the abort (handle)
3   IN  server_object
4     object reference (memory reference)
5   IN  status
6     exit status (integer)
7   IN  msg
8     exit status message (string)
9   IN  procs
10    Array of pmix_proc_t structures identifying the processes to be terminated (array of
11    handles)
12  IN  nprocs
13    Number of elements in the procs array (integer)
14  IN  cbfunc
15    Callback function pmix_op_cbfunc_t (function reference)
16  IN  cbdata
17    Data to be passed to the callback function (memory reference)

18 Returns one of the following:
19
20  • PMIX_SUCCESS , indicating that the request is being processed by the host environment - result
21    will be returned in the provided cbfunc. Note that the host must not invoke the callback function
22    prior to returning from the API.
23
24  • PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and
25    returned success - the cbfunc will not be called
26
27  • PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the
28    request, even though the function entry was provided in the server module - the cbfunc will not
    be called
29
30  • a PMIx error constant indicating either an error in the input or that the request was immediately
31    processed and failed - the cbfunc will not be called

```

## 29 Description

30 A local client called **PMIx\_Abort** . Note that the client will be in a blocked state until the host  
31 server executes the callback function, thus allowing the PMIx server library to release the client.  
32 The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all  
33 processes in the client's namespace are to be terminated.

## 34 11.4.5 pmix\_server\_fencenb\_fn\_t

### 35 Summary

36 At least one client called either **PMIx\_Fence** or **PMIx\_Fence\_nb** .

## Format

```
1 PMIx v1.0
2
3 typedef pmix_status_t (*pmix_server_fencenb_fn_t)(
4     const pmix_proc_t procs[],
5     size_t nprocs,
6     const pmix_info_t info[],
7     size_t ninfo,
8     char *data, size_t ndata,
9     pmix_modex_cbfunc_t cbfunc,
void *cbdata)
```

C

10 **IN procs**  
11 Array of **pmix\_proc\_t** structures identifying operation participants(array of handles)

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

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

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

18 **IN data**  
19 (string)

20 **IN ndata**  
21 (integer)

22 **IN cbfunc**  
23 Callback function **pmix\_modex\_cbfunc\_t** (function reference)

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

26 Returns one of the following:

- 27 • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
28 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
29 prior to returning from the API.
- 30 • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
31 returned *success* - the *cbfunc* will not be called
- 32 • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
33 request, even though the function entry was provided in the server module - the *cbfunc* will not  
34 be called
- 35 • a PMIx error constant indicating either an error in the input or that the request was immediately  
36 processed and failed - the *cbfunc* will not be called

## Required Attributes

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.

2

---

3 The following attributes are required to be supported by all host environments:

4 **PMIX\_COLLECT\_DATA** "pmix.collect" (bool)

5 Collect data and return it at the end of the operation.

## Optional Attributes

6 The following attributes are optional for host environments:

7 **PMIX\_TIMEOUT** "pmix.timeout" (int)

8 Time in seconds before the specified operation should time out (0 indicating infinite) in  
9 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
10 the target process from ever exposing its data.

11 **PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (char\*)

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

16 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)

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

## Advice to PMIx server hosts

18 Host environment are required to return **PMIX\_ERR\_NOT\_SUPPORTED** if passed an attributed  
19 marked as **PMIX\_INFO\_REQD** that they do not support, even if support for that attribute is  
20 optional.

1            **Description**

2        All local clients in the provided array of *procs* called either **PMIx\_Fence** or **PMIx\_Fence\_nb**.  
3        In either case, the host server will be called via a non-blocking function to execute the specified  
4        operation once all participating local processes have contributed. All processes in the specified  
5        *procs* array are required to participate in the **PMIx\_Fence / PMIx\_Fence\_nb** operation. The  
6        callback is to be executed once every daemon hosting at least one participant has called the host  
7        server's **pmix\_server\_fencenb\_fn\_t** function.

8            **Advice to PMIx library implementers**

9        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.

10            **Advice to PMIx server hosts**

11        The host will receive a single call for each collective operation. It is the responsibility of the host to  
12        identify the nodes containing participating processes, execute the collective across all participating  
13        nodes, and notify the local PMIx server library upon completion of the global collective. Data  
14        received from each node must be simply concatenated to form an aggregated unit, as shown in the  
following example:

15        **C**

```
16        uint8_t *blob1, *blob2, *total;
17        size_t sz_blob1, sz_blob2, sz_total;
18
19        sz_total = sz_blob1 + sz_blob2;
20        total = (uint8_t*)malloc(sz_total);
21        memcpy(total, blob1, sz_blob1);
22        memcpy(&total[sz_blob1], blob2, sz_blob2);
```

23            **C**

22        Note that the ordering of the data blobs does not matter. The host is responsible for free'ing the  
23        *data* object passed to it by the PMIx server library.

24        The provided data is to be collectively shared with all PMIx servers involved in the fence operation,  
25        and returned in the modec *cbfunc*. A **NULL** data value indicates that the local processes had no data  
26        to contribute.

27        The array of *info* structs is used to pass user-requested options to the server. This can include  
28        directives as to the algorithm to be used to execute the fence operation. The directives are optional  
29        unless the **PMIX\_INFO\_REQD** flag has been set - in such cases, the host RM is required to return  
30        an error if the directive cannot be met.

## 11.4.6 pmix\_server\_dmodex\_req\_fn\_t

### Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc.

### Format

PMIx v1.0

```
6     typedef pmix_status_t (*pmix_server_dmodex_req_fn_t)(
7         const pmix_proc_t *proc,
8         const pmix_info_t info[],
9         size_t ninfo,
10        pmix_modex_cbfunc_t cbfunc,
11        void *cbdata)
```

C

C

12 **IN proc**  
13      **pmix\_proc\_t** structure identifying the process whose data is being requested (handle)

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

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

18 **IN cbfunc**  
19      Callback function **pmix\_modex\_cbfunc\_t** (function reference)

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

22 Returns one of the following:

- 23      • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
24           will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
25           prior to returning from the API.
- 26      • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
27           request, even though the function entry was provided in the server module - the *cbfunc* will not  
28           be called
- 29      • a PMIx error constant indicating either an error in the input or that the request was immediately  
30           processed and failed - the *cbfunc* will not be called

### Required Attributes

31 PMIx libraries are required to pass any provided attributes to the host environment for processing.

## Optional Attributes

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

2 **PMIX\_TIMEOUT "pmix.timeout" (int)**

3 Time in seconds before the specified operation should time out (*0* indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

### 6 **Description**

7 Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
8 hosts the specified proc to obtain and return any information that process posted via calls to  
9 **PMIx\_Put** and **PMIx\_Commit**.

10 The array of *info* structs is used to pass user-requested options to the server. This can include a  
11 timeout to preclude an indefinite wait for data that may never become available. The directives are  
12 optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an  
13 error if the directive cannot be met.

## 14 **11.4.7 pmix\_server\_publish\_fn\_t**

### 15 **Summary**

16 Publish data per the PMIx API specification.

### 17 **Format**

18 *PMIx v1.0*

19 C

```
20     typedef pmix_status_t (*pmix_server_publish_fn_t)(
21         const pmix_proc_t *proc,
22         const pmix_info_t info[],
23         size_t ninfo,
24         pmix_op_cbfunc_t cbfunc,
25         void *cbdata)
```

26 **IN proc**  
27     **pmix\_proc\_t** structure of the process publishing the data (handle)  
28 **IN info**  
29     Array of info structures (array of handles)  
30 **IN ninfo**  
31     Number of elements in the *info* array (integer)  
32 **IN cbfunc**  
33     Callback function **pmix\_op\_cbfunc\_t** (function reference)

1   **IN   cbdata**

2   Data to be passed to the callback function (memory reference)

3   Returns one of the following:

- 4   • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
5   will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
6   prior to returning from the API.
- 7   • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
8   returned *success* - the *cbfunc* will not be called
- 9   • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
10   request, even though the function entry was provided in the server module - the *cbfunc* will not  
11   be called
- 12   • a PMIx error constant indicating either an error in the input or that the request was immediately  
13   processed and failed - the *cbfunc* will not be called

▼----- Required Attributes -----▼

14   PMIx libraries are required to pass any provided attributes to the host environment for processing.  
15   In addition, the following attributes are required to be included in the passed *info* array:

16   **PMIX\_USERID** "pmix.euid" (**uint32\_t**)

17       Effective user id.

18   **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)

19       Effective group id.

---

20   Host environments that implement this entry point are required to support the following attributes:

21   **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)

22       Value for calls to publish/lookup/unpublish or for monitoring event notifications.

23   **PMIX\_PERSISTENCE** "pmix.persist" (**pmix\_persistence\_t**)

24       Value for calls to **PMIx\_Publish**.

▲----- Optional Attributes -----▲

25   The following attributes are optional for host environments that support this operation:

26   **PMIX\_TIMEOUT** "pmix.timeout" (**int**)

27       Time in seconds before the specified operation should time out (0 indicating infinite) in  
28       error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
29       the target process from ever exposing its data.

1           **Description**

2         Publish data per the **PMIx\_Publish** specification. The callback is to be executed upon  
3         completion of the operation. The default data range is left to the host environment, but expected to  
4         be **PMIX\_RANGE\_SESSION**, and the default persistence **PMIX\_PERSIST\_SESSION** or their  
5         equivalent. These values can be specified by including the respective attributed in the *info* array.

6         The persistence indicates how long the server should retain the data.

7           **Advice to PMIx server hosts**

8         The host environment is not required to guarantee support for any specific range - i.e., the  
9         environment does not need to return an error if the data store doesn't support a specified range so  
10        long as it is covered by some internally defined range. However, the server must return an error (a)  
11        if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of  
12        published info by the original publisher - it is left to the discretion of the host environment to allow  
13        info-key-based flags to modify this behavior.

14        The **PMIX\_USERID** and **PMIX\_GRPID** of the publishing process will be provided to support  
15        authorization-based access to published information and must be returned on any subsequent  
16        lookup request.

16      **11.4.8 pmix\_server\_lookup\_fn\_t**

17           **Summary**

18         Lookup published data.

19           **Format**

PMIx v1.0           **C**

```
20        typedef pmix_status_t (*pmix_server_lookup_fn_t)(
21                   const pmix_proc_t *proc,
22                   char ***keys,
23                   const pmix_info_t info[],
24                   size_t ninfo,
25                   pmix_lookup_cbfunc_t cbfunc,
26                   void *cbdata)
```

```

1   IN  proc
2     pmix_proc_t structure of the process seeking the data (handle)
3   IN  keys
4     (array of strings)
5   IN  info
6     Array of info structures (array of handles)
7   IN  ninfo
8     Number of elements in the info array (integer)
9   IN  cbfunc
10    Callback function pmix_lookup_cbfunc_t (function reference)
11   IN  cldata
12    Data to be passed to the callback function (memory reference)

13 Returns one of the following:
14
15  • PMIX_SUCCESS , indicating that the request is being processed by the host environment - result
16    will be returned in the provided cbfunc. Note that the host must not invoke the callback function
17    prior to returning from the API.
18
19  • PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and
20    returned success - the cbfunc will not be called
21
22  • PMIX_ERR_NOT_SUPPORTED , indicating that the host environment does not support the
23    request, even though the function entry was provided in the server module - the cbfunc will not
24    be called
25
26  • a PMIx error constant indicating either an error in the input or that the request was immediately
27    processed and failed - the cbfunc will not be called

```

### Required Attributes

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:

```

26  PMIX_USERID  "pmix.euid" (uint32_t)
27    Effective user id.

28  PMIX_GRPID  "pmix.egid" (uint32_t)
29    Effective group id.

```

---

Host environments that implement this entry point are required to support the following attributes:

```

32  PMIX_RANGE  "pmix.range" (pmix_data_range_t)
33    Value for calls to publish/lookup/unpublish or for monitoring event notifications.

34  PMIX_WAIT  "pmix.wait" (int)

```

1 Caller requests that the PMIx server wait until at least the specified number of values are  
2 found (0 indicates all and is the default).

## Optional Attributes

3 The following attributes are optional for host environments that support this operation:

4 **PMIX\_TIMEOUT** "pmix.timeout" (int)

5 Time in seconds before the specified operation should time out (0 indicating infinite) in  
6 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
7 the target process from ever exposing its data.

## Description

9 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys  
10 identifying the data being requested.

11 The array of *info* structs is used to pass user-requested options to the server. The default data range  
12 is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**. This can include a  
13 wait flag to indicate that the server should wait for all data to become available before executing the  
14 callback function, or should immediately callback with whatever data is available. In addition, a  
15 timeout can be specified on the wait to preclude an indefinite wait for data that may never be  
16 published.

## Advice to PMIx server hosts

17 The **PMIX\_USERID** and **PMIX\_GRPID** of the requesting process will be provided to support  
18 authorization-based access to published information. The host environment is not required to  
19 guarantee support for any specific range - i.e., the environment does not need to return an error if  
20 the data store doesn't support a specified range so long as it is covered by some internally defined  
21 range.

## 11.4.9 pmix\_server\_unpublish\_fn\_t

### Summary

Delete data from the data store.

1      **Format**

2      *PMIx v1.0*

C

```
2      typedef pmix_status_t (*pmix_server_unpublish_fn_t) (
3               const pmix_proc_t *proc,
4               char **keys,
5               const pmix_info_t info[],
6               size_t ninfo,
7               pmix_op_cbfunc_t cbfunc,
8               void *cbdata)
```

C

9      **IN proc**

10     **pmix\_proc\_t** structure identifying the process making the request (handle)

11     **IN keys**

12     (array of strings)

13     **IN info**

14     Array of info structures (array of handles)

15     **IN ninfo**

16     Number of elements in the *info* array (integer)

17     **IN cbfunc**

18     Callback function **pmix\_op\_cbfunc\_t** (function reference)

19     **IN cbdata**

20     Data to be passed to the callback function (memory reference)

21     Returns one of the following:

- 22     • **PMIX\_SUCCESS** , indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24       prior to returning from the API.
- 25     • **PMIX\_OPERATION\_SUCCEEDED** , indicating that the request was immediately processed and  
26       returned *success* - the *cbfunc* will not be called
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED** , indicating that the host environment does not support the  
28       request, even though the function entry was provided in the server module - the *cbfunc* will not  
29       be called
- 30     • a PMIx error constant indicating either an error in the input or that the request was immediately  
31       processed and failed - the *cbfunc* will not be called

32     **Required Attributes**

33     PMIx libraries are required to pass any provided attributes to the host environment for processing.  
In addition, the following attributes are required to be included in the passed *info* array:

34     **PMIX\_USERID "pmix.euid" (uint32\_t)**

35       Effective user id.

1   **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2       Effective group id.

---

4       Host environments that implement this entry point are required to support the following attributes:

5   **PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)  
6       Value for calls to publish/lookup/unpublish or for monitoring event notifications.



### Optional Attributes

7       The following attributes are optional for host environments that support this operation:

8   **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
9       Time in seconds before the specified operation should time out (0 indicating infinite) in  
10      error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
11      the target process from ever exposing its data.



## Description

13      Delete data from the data store. The host server will be passed a **NULL**-terminated array of string  
14      keys, plus potential directives such as the data range within which the keys should be deleted. The  
15      default data range is left to the host environment, but expected to be **PMIX\_RANGE\_SESSION**.  
16      The callback is to be executed upon completion of the delete procedure.

## Advice to PMIx server hosts

17      The **PMIX\_USERID** and **PMIX\_GRP\_ID** of the requesting process will be provided to support  
18      authorization-based access to published information. The host environment is not required to  
19      guarantee support for any specific range - i.e., the environment does not need to return an error if  
20      the data store doesn't support a specified range so long as it is covered by some internally defined  
21      range.



## 11.4.10 pmix\_server\_spawn\_fn\_t

### Summary

24      Spawn a set of applications/processes as per the **PMIX\_Spawn** API.

1           **Format**2       *PMIx v1.0*

C

```
2       typedef pmix_status_t (*pmix_server_spawn_fn_t)(
3                           const pmix_proc_t *proc,
4                           const pmix_info_t job_info[],
5                           size_t ninfo,
6                           const pmix_app_t apps[],
7                           size_t napps,
8                           pmix_spawn_cbfunc_t cbfunc,
9                           void *cbdata)
```

C

10      **IN proc**  
11        *pmix\_proc\_t* structure of the process making the request (handle)

12      **IN job\_info**  
13        Array of info structures (array of handles)

14      **IN ninfo**  
15        Number of elements in the *jobinfo* array (integer)

16      **IN apps**  
17        Array of *pmix\_app\_t* structures (array of handles)

18      **IN napps**  
19        Number of elements in the *apps* array (integer)

20      **IN cbfunc**  
21        Callback function *pmix\_spawn\_cbfunc\_t* (function reference)

22      **IN cbdata**  
23        Data to be passed to the callback function (memory reference)

24      Returns one of the following:

- **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

1 PMIx libraries are required to pass any provided attributes to the host environment for processing.  
2 In addition, the following attributes are required to be included in the passed *info* array:

3 **PMIX\_USERID** "pmix.euid" (**uint32\_t**)

4     Effective user id.

5 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)

6     Effective group id.

7  
8 Host environments that provide this module entry point are required to pass the **PMIX\_SPAWNED**  
9 and **PMIX\_PARENT\_ID** attributes to all PMIx servers launching new child processes so those  
10 values can be returned to clients upon connection to the PMIx server. In addition, they are required  
11 to support the following attributes when present in either the *job\_info* or the *info* array of an  
12 element of the *apps* array:

13 **PMIX\_WDIR** "pmix.wdir" (**char\***)

14     Working directory for spawned processes.

15 **PMIX\_SET\_SESSION\_CWD** "pmix.ssncwd" (**bool**)

16     Set the application's current working directory to the session working directory assigned by  
17     the RM - when accessed using **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for  
18     the rank to discover the session working directory assigned to the provided namespace

19 **PMIX\_PREFIX** "pmix.prefix" (**char\***)

20     Prefix to use for starting spawned processes.

21 **PMIX\_HOST** "pmix.host" (**char\***)

22     Comma-delimited list of hosts to use for spawned processes.

23 **PMIX\_HOSTFILE** "pmix.hostfile" (**char\***)

24     Hostfile to use for spawned processes.

## Optional Attributes

25 The following attributes are optional for host environments that support this operation:

26 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (**char\***)

27     Hostfile listing hosts to add to existing allocation.

28 **PMIX\_ADD\_HOST** "pmix.addhost" (**char\***)

29     Comma-delimited list of hosts to add to the allocation.

30 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (**bool**)

31     Preload binaries onto nodes.

32 **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (**char\***)

```

1      Comma-delimited list of files to pre-position on nodes.
2      PMIX_PERSONALITY "pmix.pers" (char*)
3          Name of personality to use.
4      PMIX_MAPPER "pmix.mapper" (char*)
5          Mapping mechanism to use for placing spawned processes - when accessed using
6          PMIx_Get, use the PMIX_RANK_WILDCARD value for the rank to discover the mapping
7          mechanism used for the provided namespace.
8      PMIX_DISPLAY_MAP "pmix.dispmap" (bool)
9          Display process mapping upon spawn.
10     PMIX_PPR "pmix.ppr" (char*)
11         Number of processes to spawn on each identified resource.
12     PMIX_MAPBY "pmix.mapby" (char*)
13         Process mapping policy - when accessed using PMIx_Get, use the
14         PMIX_RANK_WILDCARD value for the rank to discover the mapping policy used for the
15         provided namespace
16     PMIX_RANKBY "pmix.rankby" (char*)
17         Process ranking policy - when accessed using PMIx_Get, use the
18         PMIX_RANK_WILDCARD value for the rank to discover the ranking algorithm used for the
19         provided namespace
20     PMIX_BINDTO "pmix.bindto" (char*)
21         Process binding policy - when accessed using PMIx_Get, use the
22         PMIX_RANK_WILDCARD value for the rank to discover the binding policy used for the
23         provided namespace
24     PMIX_NON_PMI "pmix.nonpmi" (bool)
25         Spawns processes will not call PMIx_Init.
26     PMIX_STDIN_TGT "pmix.stdin" (uint32_t)
27         Spawns process rank that is to receive any forwarded stdin.
28     PMIX_FWD_STDIN "pmix.fwd.stdin" (pmix_rank_t)
29         The requester intends to push information from its stdin to the indicated process. The
30         local spawn agent should, therefore, ensure that the stdin channel to that process remains
31         available. A rank of PMIX_RANK_WILDCARD indicates that all processes in the spawned
32         job are potential recipients.
33     PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
34         Requests that the stdout of the spawned processes be forwarded to the requester for
35         handling
36     PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)

```

Requests that the **stderr** of the spawned processes be forwarded to the requester for handling

**PMIX\_DEBUGGER\_DAEMONS "pmix.debugger" (bool)**

Included in the *info* 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 application output with the identity of the source process.

**PMIX\_TIMESTAMP\_OUTPUT "pmix.tsout" (bool)**

Timestamp output from applications.

**PMIX\_MERGE\_STDERR\_STDOUT "pmix.mergeerrout" (bool)**

Merge **stdout** and **stderr** streams from application processes.

**PMIX\_OUTPUT\_TO\_FILE "pmix.outfile" (char\*)**

Direct application output (both stdout and stderr) into files of form "<filename>.rank"

**PMIX\_INDEX\_ARGV "pmix.indxargv" (bool)**

Mark the **argv** with the rank of the process.

**PMIX\_CPUS\_PER\_PROC "pmix.cpuperproc" (uint32\_t)**

Number of cpus to assign to each rank - when accessed using **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for the rank to discover the cpus/process assigned to the provided namespace

**PMIX\_NO\_PROCS\_ON\_HEAD "pmix.nolocal" (bool)**

Do not place processes on the head node.

**PMIX\_NO\_OVERSUBSCRIBE "pmix.noover" (bool)**

Do not oversubscribe the cpus.

**PMIX\_REPORT\_BINDINGS "pmix.repbind" (bool)**

Report bindings of the individual processes.

**PMIX\_CPU\_LIST "pmix.cpulist" (char\*)**

List of cpus to use for this job - when accessed using **PMIx\_Get**, use the **PMIX\_RANK\_WILDCARD** value for the rank to discover the cpu list used for the provided namespace

**PMIX\_JOB\_RECOVERABLE "pmix.recover" (bool)**

Application supports recoverable operations.

```
1 PMIX_JOB_CONTINUOUS "pmix.continuous" (bool)
2 Application is continuous, all failed processes should be immediately restarted.
3 PMIX_MAX_RESTARTS "pmix.maxrestarts" (uint32_t)
4 Maximum number of times to restart a job - when accessed using PMIx_Get , use the
5 PMIX_RANK_WILDCARD value for the rank to discover the max restarts for the provided
6 namespace
7 PMIX_TIMEOUT "pmix.timeout" (int)
8 Time in seconds before the specified operation should time out (0 indicating infinite) in
9 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
10 the target process from ever exposing its data.
```

## 11 Description

12 Spawn a set of applications/processes as per the PMIx\_Spawn API. Note that applications are not  
13 required to be MPI or any other programming model. Thus, the host server cannot make any  
14 assumptions as to their required support. The callback function is to be executed once all processes  
15 have been started. An error in starting any application or process in this request shall cause all  
16 applications and processes in the request to be terminated, and an error returned to the originating  
17 caller.

18 Note that a timeout can be specified in the job\_info array to indicate that failure to start the  
19 requested job within the given time should result in termination to avoid hangs.

### 20 11.4.11 pmix\_server\_connect\_fn\_t

#### 21 Summary

22 Record the specified processes as *connected*.

#### 23 Format

PMIx v1.0

C

```
24     typedef pmix_status_t (*pmix_server_connect_fn_t) (
25             const pmix_proc_t procs[],
26             size_t nprocs,
27             const pmix_info_t info[],
28             size_t ninfo,
29             pmix_op_cbfunc_t cbfunc,
30             void *cbdata)
```

```

1   IN  procs
2     Array of pmix_proc_t structures identifying participants (array of handles)
3   IN  nprocs
4     Number of elements in the procs array (integer)
5   IN  info
6     Array of info structures (array of handles)
7   IN  ninfo
8     Number of elements in the info array (integer)
9   IN  cbfunc
10    Callback function pmix_op_cbfunc_t (function reference)
11   IN  cbdata
12     Data to be passed to the callback function (memory reference)

13 Returns one of the following:
14
15  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
16    will be returned in the provided cbfunc. Note that the host must not invoke the callback function
17    prior to returning from the API.
18
19  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
20    returned success - the cbfunc will not be called
21
22  • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
23    request, even though the function entry was provided in the server module - the cbfunc will not
24    be called
25
26  • a PMIx error constant indicating either an error in the input or that the request was immediately
27    processed and failed - the cbfunc will not be called

```

### Required Attributes

24 PMIx libraries are required to pass any provided attributes to the host environment for processing.

### Optional Attributes

25 The following attributes are optional for host environments that support this operation:

```

26  PMIX_TIMEOUT "pmix.timeout" (int)
27    Time in seconds before the specified operation should time out (0 indicating infinite) in
28    error. The timeout parameter can help avoid “hangs” due to programming errors that prevent
29    the target process from ever exposing its data.

30  PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

```

1 Comma-delimited list of algorithms to use for the collective operation. PMIx does not  
2 impose any requirements on a host environment's collective algorithms. Thus, the  
3 acceptable values for this attribute will be environment-dependent - users are encouraged to  
4 check their host environment for supported values.

5 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)

6 If **true**, indicates that the requested choice of algorithm is mandatory.

## 7 **Description**

8 Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The  
9 callback is to be executed once every daemon hosting at least one participant has called the host  
10 server's **pmix\_server\_connect\_fn\_t** function, and the host environment has completed any  
11 supporting operations required to meet the terms of the PMIx definition of *connected* processes.

### Advice to PMIx library implementers

12 The PMIx server library is required to aggregate participation by local clients, passing the request  
13 to the host environment once all local participants have executed the API.

### Advice to PMIx server hosts

14 The host will receive a single call for each collective operation. It is the responsibility of the host to  
15 identify the nodes containing participating processes, execute the collective across all participating  
16 nodes, and notify the local PMIx server library upon completion of the global collective.

## 17 **11.4.12 pmix\_server\_disconnect\_fn\_t**

### 18 **Summary**

19 Disconnect a previously connected set of processes.

1      **Format**

2      *PMIx v1.0*

C

```
2      typedef pmix_status_t (*pmix_server_disconnect_fn_t)(
3               const pmix_proc_t procs[],
4               size_t nprocs,
5               const pmix_info_t info[],
6               size_t ninfo,
7               pmix_op_cbfunc_t cbfunc,
8               void *cbdata)
```

C

9      **IN procs**

10     Array of **pmix\_proc\_t** structures identifying participants (array of handles)

11     **IN nprocs**

12     Number of elements in the *procs* array (integer)

13     **IN info**

14     Array of info structures (array of handles)

15     **IN ninfo**

16     Number of elements in the *info* array (integer)

17     **IN cbfunc**

18     Callback function **pmix\_op\_cbfunc\_t** (function reference)

19     **IN cbdata**

20     Data to be passed to the callback function (memory reference)

21     Returns one of the following:

- 22     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
24       prior to returning from the API.
- 25     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26       returned *success* - the *cbfunc* will not be called
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28       request, even though the function entry was provided in the server module - the *cbfunc* will not  
29       be called
- 30     • a PMIx error constant indicating either an error in the input or that the request was immediately  
31       processed and failed - the *cbfunc* will not be called

32      **Required Attributes**

32      PMIx libraries are required to pass any provided attributes to the host environment for processing.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT "pmix.timeout" (int)**

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

### 6 Description

7 Disconnect a previously connected set of processes. The callback is to be executed once every  
8 daemon hosting at least one participant has called the host server’s has called the  
9 **pmix\_server\_disconnect\_fn\_t** function, and the host environment has completed any  
10 required supporting operations.

### Advice to PMIx library implementers

11 The PMIx server library is required to aggregate participation by local clients, passing the request  
12 to the host environment once all local participants have executed the API.

### Advice to PMIx server hosts

13 The host will receive a single call for each collective operation. It is the responsibility of the host to  
14 identify the nodes containing participating processes, execute the collective across all participating  
15 nodes, and notify the local PMIx server library upon completion of the global collective.

16 A **PMIX\_ERR\_INVALID\_OPERATION** error must be returned if the specified set of *procs* was  
17 not previously *connected* via a call to the **pmix\_server\_connect\_fn\_t** function.

## 11.4.13 **pmix\_server\_register\_events\_fn\_t**

### Summary

Register to receive notifications for the specified events.

## Format

```
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 `mix_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 sbfunc

Callback function **pmix op cbfunc t** (function reference)

IN chdata

Data to be passed to the callback function (memory reference)

Returns one of

www.english-test.net

- **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.

1   **PMIX\_GRP\_ID** "pmix.egid" (**uint32\_t**)  
2       Effective group id.



3   **Description**

4   Register to receive notifications for the specified status codes. The *info* array included in this API is  
5   reserved for possible future directives to further steer notification.



Advice to PMIx library implementers

6   The PMIx server library must track all client registrations for subsequent notification. This module  
7   function shall only be called when:

- 8   • the client has requested notification of an environmental code (i.e., a PMIx code in the range  
9      beyond **PMIX\_ERR\_SYS\_OTHER**) or a code that lies outside the defined PMIx range of  
10     constants; and
- 11   • the PMIx server library has not previously requested notification of that code - i.e., the host  
12     environment is to be contacted only once a given unique code value
- 



Advice to PMIx server hosts

13   The host environment is required to pass to its PMIx server library all non-environmental events  
14   that directly relate to a registered namespace without the PMIx server library explicitly requesting  
15   them. Environmental events are to be translated to their nearest PMIx equivalent code as defined in  
16   the range between **PMIX\_ERR\_SYS\_BASE** and **PMIX\_ERR\_SYS\_OTHER** (inclusive).



17 **11.4.14 pmix\_server\_deregister\_events\_fn\_t**

18   **Summary**

19   Deregister to receive notifications for the specified events.

## Format

C

*PMIx v1.0* ▾ C  
typedef pmix\_status\_t (\*pmix\_server\_deregister\_events\_fn\_t)(  
 pmix\_status\_t \*codes,  
 size\_t ncodes,  
 pmix\_op\_cbfunc\_t cbfunc,  
 void \*cbdata)

IN codes

Array of **pmix\_status\_t** values (array of handles)

IN ncodes

Number of elements in the *codes* array (integer)

## IN chfunc

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_ERR_SYS_BASE` and `PMIX_ERR_SYS_OTHER`, inclusive) or codes that lies outside the defined PMIx range of constants; and

- 1     • no client (including the server library itself) remains registered for notifications on any included  
2        code - i.e., a code should be included in this call only when no registered notifications against it  
3        remain.
- 

4 **11.4.15 pmix\_server\_notify\_event\_fn\_t**

5 **Summary**

6 Notify the specified processes of an event.

7 **Format**

PMIx v2.0

8 `typedef pmix_status_t (*pmix_server_notify_event_fn_t)(pmix_status_t code,`  
9                    `const pmix_proc_t *source,`  
10                    `pmix_data_range_t range,`  
11                    `pmix_info_t info[],`  
12                    `size_t ninfo,`  
13                    `pmix_op_cbfunc_t cbfunc,`  
14                    `void *cbdata);`

C

15 **IN code**

16     The `pmix_status_t` event code being referenced structure (handle)

17 **IN source**

18     `pmix_proc_t` of process that generated the event (handle)

19 **IN range**

20     `pmix_data_range_t` range over which the event is to be distributed (handle)

21 **IN info**

22     Optional array of `pmix_info_t` structures containing additional information on the event  
23        (array of handles)

24 **IN ninfo**

25     Number of elements in the *info* array (integer)

26 **IN cbfunc**

27     Callback function `pmix_op_cbfunc_t` (function reference)

28 **IN cbdata**

29     Data to be passed to the callback function (memory reference)

30     Returns one of the following:

- 31     • `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result  
32        will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
33        prior to returning from the API.
- 34     • `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and  
35        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)

Value for calls to publish/lookup/unpublish or for monitoring event notifications.

## Description

Notify the specified processes (described through a combination of *range* and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. The process generating the event is provided in the *source* parameter, and any further descriptive information is included in the *info* array.

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.

#### 11.4.16 pmix server listener fn t

## Summary

Register a socket the host server can monitor for connection requests.

## Format

```
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** **cbddata**  
(memory reference)

Returns **PMIX\_SUCCESS** indicating that the request is accepted, or a negative value corresponding to a PMIx error constant indicating that the request has been rejected.

## Description

Register a socket the host environment can monitor for connection requests, harvest them, and then call the PMIx server library's internal callback function for further processing. A listener thread is essential to efficiently harvesting connection requests from large numbers of local clients such as occur when running on large SMPs. The host server listener is required to call accept on the incoming connection request, and then pass the resulting socket to the provided cbfunc. A **NULL** for this function will cause the internal PMIx server to spawn its own listener thread.

## 11.4.17 pmix\_server\_query\_fn\_t

## Summary

Query information from the resource manager.

## Format

*PMIx v2.0*

```
typedef pmix_status_t (*pmix_server_query_fn_t) (
    pmix_proc_t *proct,
    pmix_query_t *queries, size_t nqueries,
    pmix_info_cbfunc_t cbfunc,
    void *cbdata)
```

**IN** `proc`  
    `pmix_proc_t` structure of the requesting process (handle)  
**IN** `queries`  
    Array of `pmix_query_t` structures (array of handles)

```
1   IN  nqueries
2     Number of elements in the queries array (integer)
3   IN  cbfunc
4     Callback function pmix_info_cbfunc_t (function reference)
5   IN  cbdatal
6     Data to be passed to the callback function (memory reference)
7
8 Returns one of the following:
9
10 • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result
11 will be returned in the provided cbfunc. Note that the host must not invoke the callback function
12 prior to returning from the API.
13
14 • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and
15 returned success - the cbfunc will not be called
16
17 • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the
18 request, even though the function entry was provided in the server module - the cbfunc will not
19 be called
20
21 • a PMIx error constant indicating either an error in the input or that the request was immediately
22 processed and failed - the cbfunc will not be called
```

### 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:

```
20  PMIX_USERID  "pmix.euid" (uint32_t)
21    Effective user id.
22
23  PMIX_GRPID  "pmix.egid" (uint32_t)
24    Effective group id.
```

### Optional Attributes

The following attributes are optional for host environments that support this operation:

```
25  PMIX_QUERY_NAMESPACES  "pmix.qry.ns" (char*)
26    Request a comma-delimited list of active namespaces. NO QUALIFIERS
27
28  PMIX_QUERY_JOB_STATUS  "pmix.qry.jst" (pmix_status_t)
29    Status of a specified, currently executing job. REQUIRED QUALIFIER: PMIX_NSPACE
30    indicating the namespace whose status is being queried
31
32  PMIX_QUERY_QUEUE_LIST  "pmix.qry qlst" (char*)
33    Request a comma-delimited list of scheduler queues. NO QUALIFIERS
34
35  PMIX_QUERY_QUEUE_STATUS  "pmix.qry.qst" (char*)
```

```

1 Returns status of a specified scheduler queue, expressed as a string. SUPPORTED
2 QUALIFIERS: PMIX_ALLOC_QUEUE naming specific queue whose status is being
3 requested

4 PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
5 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
6 process in the specified namespace, ordered by process job rank. REQUIRED QUALIFIER:
7 PMIX_NSPACE indicating the namespace whose process table is being queried

8 PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
9 Returns a (pmix_data_array_t) array of pmix_proc_info_t, one entry for each
10 process in the specified namespace executing on the same node as the requester, ordered by
11 process job rank. REQUIRED QUALIFIER: PMIX_NSPACE indicating the namespace
12 whose process table is being queried

13 PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
14 Return a comma-delimited list of supported spawn attributes. NO QUALIFIERS

15 PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
16 Return a comma-delimited list of supported debug attributes. NO QUALIFIERS

17 PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
18 Return information on memory usage for the processes indicated in the qualifiers.
19 SUPPORTED QUALIFIERS: PMIX_NSPACE and PMIX_RANK, or PMIX_PROCID of
20 specific process(es) whose memory usage is being requested

21 PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
22 Constrain the query to local information only. NO QUALIFIERS

23 PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
24 Report only average values for sampled information. NO QUALIFIERS

25 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
26 Report minimum and maximum values. NO QUALIFIERS

27 PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)
28 String identifier of the allocation whose status is being requested. NO QUALIFIERS

29 PMIX_TIME_REMAINING "pmix.time.remaining" (char*)
30 Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
31 SUPPORTED QUALIFIERS: PMIX_NSPACE of the namespace whose info is being
32 requested (defaults to allocation containing the caller)

```



1           **Description**  
2         Query information from the host environment. The query will include the namespace/rank of the  
3         process that is requesting the info, an array of `pmix_query_t` describing the request, and a  
4         callback function/data for the return.

5            **Advice to PMIx library implementers** 

6         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.  


7   **11.4.18 pmix\_server\_tool\_connection\_fn\_t**

8           **Summary**  
9         Register that a tool has connected to the server.

10          **Format**

PMIx v2.0

11          **C**   
12         `typedef void (*pmix_server_tool_connection_fn_t) (`  
13                 `pmix_info_t info[], size_t ninfo,`  
14                 `pmix_tool_connection_cbfunc_t cbfunc,`  
               `void *cbdata)`

15          **IN info**  
16         Array of `pmix_info_t` structures (array of handles)  
17          **IN ninfo**  
18         Number of elements in the *info* array (integer)  
19          **IN cbfunc**  
20         Callback function `pmix_tool_connection_cbfunc_t` (function reference)  
21          **IN cbdata**  
22         Data to be passed to the callback function (memory reference)

23            **Required Attributes** 

24         PMIx libraries are required to pass the following attributes in the *info* array:

25         `PMIX_USERID "pmix.euid" (uint32_t)`  
               Effective user id.  
26         `PMIX_GRP_ID "pmix.egid" (uint32_t)`  
               Effective group id.  


## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)

3 Requests that the **stdout** of the spawned processes be forwarded to the requester for  
4 handling

5 **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)

6 Requests that the **stderr** of the spawned processes be forwarded to the requester for  
7 handling

8 **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (pmix\_rank\_t)

9 The requester intends to push information from its **stdin** to the indicated process. The  
10 local spawn agent should, therefore, ensure that the **stdin** channel to that process remains  
11 available. A rank of **PMIX\_RANK\_WILDCARD** indicates that all processes in the spawned  
12 job are potential recipients.

## 13 Description

14 Register that a tool has connected to the server, and request that the tool be assigned a  
15 namespace/rank identifier for further interactions. The **pmix\_info\_t** array is used to pass  
16 qualifiers for the connection request, including the effective uid and gid of the calling tool for  
17 authentication purposes.

## Advice to PMIx server hosts

18 The host environment is solely responsible for authenticating and authorizing the connection, and  
19 for authorizing all subsequent tool requests. The host must not execute the callback function prior  
20 to returning from the API.

## 21 11.4.19 pmix\_server\_log\_fn\_t

### 22 Summary

23 Log data on behalf of a client.

1      **Format**

2      **typedef void (\*pmix\_server\_log\_fn\_t)(**  
3                  **const pmix\_proc\_t \*client,**  
4                  **const pmix\_info\_t data[], size\_t ndata,**  
5                  **const pmix\_info\_t directives[], size\_t ndirs,**  
6                  **pmix\_op\_cbfunc\_t cbfunc, void \*cbdata)**

7      **IN client**  
8                  **pmix\_proc\_t** structure (handle)  
9      **IN data**  
10                 Array of info structures (array of handles)  
11      **IN ndata**  
12                 Number of elements in the *data* array (integer)  
13      **IN directives**  
14                 Array of info structures (array of handles)  
15      **IN ndirs**  
16                 Number of elements in the *directives* array (integer)  
17      **IN cbfunc**  
18                 Callback function **pmix\_op\_cbfunc\_t** (function reference)  
19      **IN cbdata**  
20                 Data to be passed to the callback function (memory reference)

21                 ----- Required Attributes -----

22      PMIx libraries are required to pass any provided attributes to the host environment for processing.  
23      In addition, the following attributes are required to be included in the passed *info* array:

24      **PMIX\_USERID** "pmix.euid" (**uint32\_t**)  
25                 Effective user id.  
26      **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
27                 Effective group id.

28      Host environments that provide this module entry point are required to support the following  
29      attributes:

30      **PMIX\_LOG\_STDERR** "pmix.log.stderr" (**char\***)  
31                 Log string to **stderr**.  
32      **PMIX\_LOG\_STDOUT** "pmix.log.stdout" (**char\***)  
33                 Log string to **stdout**.  
34      **PMIX\_LOG\_SYSLOG** "pmix.log.syslog" (**char\***)

1 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,  
2 otherwise to local syslog

### Optional Attributes

3 The following attributes are optional for host environments that support this operation:

4 **PMIX\_LOG\_MSG** "pmix.log.msg" (**pmix\_byte\_object\_t**)

5 Message blob to be sent somewhere.

6 **PMIX\_LOG\_EMAIL** "pmix.log.email" (**pmix\_data\_array\_t**)

7 Log via email based on **pmix\_info\_t** containing directives.

8 **PMIX\_LOG\_EMAIL\_ADDR** "pmix.log.emaddr" (**char\***)

9 Comma-delimited list of email addresses that are to receive the message.

10 **PMIX\_LOG\_EMAIL\_SUBJECT** "pmix.log.emsub" (**char\***)

11 Subject line for email.

12 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (**char\***)

13 Message to be included in email.

### Description

15 Log data on behalf of a client. This function is not intended for output of computational results, but  
16 rather for reporting status and error messages. The host must not execute the callback function prior  
17 to returning from the API.

## 11.4.20 pmix\_server\_alloc\_fn\_t

### Summary

Request allocation operations on behalf of a client.

1      **Format**

2      **typedef pmix\_status\_t (\*pmix\_server\_alloc\_fn\_t)(**  
3           **const pmix\_proc\_t \*client,**  
4           **pmix\_alloc\_directive\_t directive,**  
5           **const pmix\_info\_t data[], size\_t ndata,**  
6           **pmix\_info\_cfunc\_t cbfunc, void \*cbdata)**

7      **IN client**  
8          **pmix\_proc\_t** structure of process making request (handle)  
9      **IN directive**  
10       Specific action being requested (**pmix\_alloc\_directive\_t**)  
11      **IN data**  
12       Array of info structures (array of handles)  
13      **IN ndata**  
14       Number of elements in the *data* array (integer)  
15      **IN cbfunc**  
16       Callback function **pmix\_info\_cfunc\_t** (function reference)  
17      **IN cbdata**  
18       Data to be passed to the callback function (memory reference)

19      Returns one of the following:

- 20      • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
21       will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
22       prior to returning from the API.
- 23      • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
24       returned *success* - the *cbfunc* will not be called
- 25      • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
26       request, even though the function entry was provided in the server module - the *cbfunc* will not  
27       be called
- 28      • a PMIx error constant indicating either an error in the input or that the request was immediately  
29       processed and failed - the *cbfunc* will not be called

30      **Required Attributes**

31      PMIx libraries are required to pass any provided attributes to the host environment for processing.  
32      In addition, the following attributes are required to be included in the passed *info* array:

33      **PMIX\_USERID "pmix.euid" (uint32\_t)**  
34       Effective user id.  
35      **PMIX\_GRPID "pmix.egid" (uint32\_t)**  
36       Effective group id.

---

1  
2 Host environments that provide this module entry point are required to support the following  
3 attributes:

4 **PMIX\_ALLOC\_ID** "pmix.alloc.id" (**char\***)

5 A string identifier (provided by the host environment) for the resulting allocation which can  
6 later be used to reference the allocated resources in, for example, a call to **PMIx\_Spawn**.

7 **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (**uint64\_t**)

8 The number of nodes being requested in an allocation request

9 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (**uint64\_t**)

10 Number of cpus being requested in an allocation request.

11 **PMIX\_ALLOC\_TIME** "pmix.alloc.time" (**uint32\_t**)

12 Total session time (in seconds) being requested in an allocation request.

### Optional Attributes

13 The following attributes are optional for host environments that support this operation:

14 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (**char\***)

15 Regular expression of the specific nodes being requested in an allocation request

16 **PMIX\_ALLOC\_NUM\_CPU\_LIST** "pmix.alloc.ncpulist" (**char\***)

17 Regular expression of the number of cpus for each node being requested in an allocation  
18 request.

19 **PMIX\_ALLOC\_CPU\_LIST** "pmix.alloc.cpulist" (**char\***)

20 Regular expression of the specific cpus being requested in an allocation request.

21 **PMIX\_ALLOC\_MEM\_SIZE** "pmix.alloc.msize" (**float**)

22 Number of Megabytes[base2] of memory (per process) being requested in an allocation  
23 request.

24 **PMIX\_ALLOC\_FABRIC** "pmix.alloc.net" (**array**)

25 Array of **pmix\_info\_t** describing requested fabric resources. This must include at least:

26 **PMIX\_ALLOC\_FABRIC\_ID**, **PMIX\_ALLOC\_FABRIC\_TYPE**, and

27 **PMIX\_ALLOC\_FABRIC\_ENDPTS**, plus whatever other descriptors are desired.

28 **PMIX\_ALLOC\_FABRIC\_ID** "pmix.alloc.netid" (**char\***)

1       The key to be used when accessing this requested fabric allocation. The allocation will be  
2       returned/stored as a **pmix\_data\_array\_t** of **pmix\_info\_t** indexed by this key and  
3       containing at least one entry with the same key and the allocated resource description. The  
4       type of the included value depends upon the fabric support. For example, a TCP allocation  
5       might consist of a comma-delimited string of socket ranges such as  
6       "32000-32100,33005,38123-38146". Additional entries will consist of any provided  
7       resource request directives, along with their assigned values. Examples include:  
8           **PMIX\_ALLOC\_FABRIC\_TYPE** - the type of resources provided;  
9           **PMIX\_ALLOC\_FABRIC\_PLANE** - if applicable, what plane the resources were assigned  
10          from; **PMIX\_ALLOC\_FABRIC\_QOS** - the assigned QoS; **PMIX\_ALLOC\_BANDWIDTH** -  
11          the allocated bandwidth; **PMIX\_ALLOC\_FABRIC\_SEC\_KEY** - a security key for the  
12          requested fabric allocation. NOTE: the assigned values may differ from those requested,  
13          especially if **PMIX\_INFO\_REQD** was not set in the request.

14       **PMIX\_ALLOC\_BANDWIDTH** "pmix.alloc.bw" (**float**)  
15       Fabric bandwidth (in Megabits[base2]/sec) for the job being requested in an allocation  
16       request.  
17       **PMIX\_ALLOC\_FABRIC\_QOS** "pmix.alloc.netqos" (**char\***)  
18       Fabric quality of service level for the job being requested in an allocation request.



## 19       **Description**

20       Request new allocation or modifications to an existing allocation on behalf of a client. Several  
21       broad categories are envisioned, including the ability to:

- 22
  - Request allocation of additional resources, including memory, bandwidth, and compute for an  
23           existing allocation. Any additional allocated resources will be considered as part of the current  
24           allocation, and thus will be released at the same time.
  - Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not  
26           affiliated with) the allocation of the requestor - thus the termination of one allocation will not  
27           impact the other.
  - Extend the reservation on currently allocated resources, subject to scheduling availability and  
29           priorities.
  - Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to  
31           the scheduler with a promise to return them upon subsequent request.

32       The callback function provides a *status* to indicate whether or not the request was granted, and to  
33       provide some information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
34       **pmix\_info\_t** structures.

## 35      **11.4.21 pmix\_server\_job\_control\_fn\_t**

### 36      **Summary**

37      Execute a job control action on behalf of a client.

## Format

```
typedef pmix_status_t (*pmix_server_job_control_fn_t) (const pmix_proc_t *requestor, const pmix_proc_t targets[], size_t ntargs, const pmix_info_t directives[], size_t ndirs, pmix_info_cbfunc_t cbfunc, void *cbdata)
```

|           |                    |                                                                |
|-----------|--------------------|----------------------------------------------------------------|
| <b>IN</b> | <b>requestor</b>   |                                                                |
|           | <b>pmix_proc_t</b> | structure of requesting process (handle)                       |
| <b>IN</b> | <b>targets</b>     |                                                                |
|           |                    | Array of proc structures (array of handles)                    |
| <b>IN</b> | <b>ntargets</b>    |                                                                |
|           |                    | Number of elements in the <i>targets</i> array (integer)       |
| <b>IN</b> | <b>directives</b>  |                                                                |
|           |                    | Array of info structures (array of handles)                    |
| <b>IN</b> | <b>ndirs</b>       |                                                                |
|           |                    | Number of elements in the <i>info</i> array (integer)          |
| <b>IN</b> | <b>cbfunc</b>      |                                                                |
|           |                    | Callback function <b>pmix_op_cbfunc_t</b> (function reference) |
| <b>IN</b> | <b>cbdata</b>      |                                                                |
|           |                    | 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.

```
1 PMIX_GRP_ID "pmix.egid" (uint32_t)
2 Effective group id.
3
4 Host environments that provide this module entry point are required to support the following
5 attributes:
6 PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
7 Provide a string identifier for this request. The user can provide an identifier for the
8 requested operation, thus allowing them to later request status of the operation or to
9 terminate it. The host, therefore, shall track it with the request for future reference.
10 PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
11 Pause the specified processes.
12 PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
13 Resume ("un-pause") the specified processes.
14 PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
15 Forcibly terminate the specified processes and cleanup.
16 PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
17 Send given signal to specified processes.
18 PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)
19 Politely terminate the specified processes.
```

## Optional Attributes

The following attributes are optional for host environments that support this operation:

```
21 PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
22 Cancel the specified request - the provided request ID must match the
23 PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control. An ID of
24 NULL implies cancel all requests from this requestor.
25 PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
26 Restart the specified processes using the given checkpoint ID.
27 PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
28 Checkpoint the specified processes and assign the given ID to it.
29 PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
30 Use event notification to trigger a process checkpoint.
31 PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
32 Use the given signal to trigger a process checkpoint.
33 PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
```

```

1      Time in seconds to wait for a checkpoint to complete.
2      PMIX_JOB_CTRL_CHECKPOINT_METHOD
3      "pmix.jctrl.ckmethod" (pmix_data_array_t)
4      Array of pmix_info_t declaring each method and value supported by this application.
5      PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
6      Regular expression identifying nodes that are to be provisioned.
7      PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnimsg" (char*)
8      Name of the image that is to be provisioned.
9      PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)
10     Indicate that the job can be pre-empted.

```



## 11 Description

12 Execute a job control action on behalf of a client. The *targets* array identifies the processes to  
13 which the requested job control action is to be applied. A **NULL** value can be used to indicate all  
14 processes in the caller's namespace. The use of **PMIX\_RANK\_WILDCARD** can also be used to  
15 indicate that all processes in the given namespace are to be included.

16 The directives are provided as **pmix\_info\_t** structures in the *directives* array. The callback  
17 function provides a *status* to indicate whether or not the request was granted, and to provide some  
18 information as to the reason for any denial in the **pmix\_info\_cbfunc\_t** array of  
19 **pmix\_info\_t** structures.

## 20 11.4.22 pmix\_server\_monitor\_fn\_t

### 21 Summary

22 Request that a client be monitored for activity.

### 23 Format

*PMIx v2.0*

C

```

24      typedef pmix_status_t (*pmix_server_monitor_fn_t)(
25          const pmix_proc_t *requestor,
26          const pmix_info_t *monitor, pmix_status_t error,
27          const pmix_info_t directives[], size_t ndirs,
28          pmix_info_cbfunc_t cbfunc, void *cbdata);

```

```

1   IN requestor
2     pmix_proc_t structure of requesting process (handle)
3   IN monitor
4     pmix_info_t identifying the type of monitor being requested (handle)
5   IN error
6     Status code to use in generating event if alarm triggers (integer)
7   IN directives
8     Array of info structures (array of handles)
9   IN ndirs
10    Number of elements in the info array (integer)
11   IN cbfunc
12     Callback function pmix_op_cbfunc_t (function reference)
13   IN cldata
14     Data to be passed to the callback function (memory reference)

```

15 Returns one of the following:

- 16 • **PMIX\_SUCCESS** , indicating that the request is being processed by the host environment - result  
17 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function  
18 prior to returning from the API.
- 19 • **PMIX\_OPERATION\_SUCCEEDED** , indicating that the request was immediately processed and  
20 returned *success* - the *cbfunc* will not be called
- 21 • **PMIX\_ERR\_NOT\_SUPPORTED** , indicating that the host environment does not support the  
22 request, even though the function entry was provided in the server module - the *cbfunc* will not  
23 be called
- 24 • a PMIx error constant indicating either an error in the input or that the request was immediately  
25 processed and failed - the *cbfunc* will not be called

26 This entry point is only called for monitoring requests that are not directly supported by the PMIx  
27 server library itself.

### Required Attributes

28 If supported by the PMIx server library, then the library must not pass any supported attributes to  
29 the host environment. Any attributes provided by the client that are not directly supported by the  
30 server library must be passed to the host environment if it provides this module entry. In addition,  
31 the following attributes are required to be included in the passed *info* array:

```

32 PMIX_USERID "pmix.euid" (uint32_t)
33   Effective user id.
34 PMIX_GRPID "pmix.egid" (uint32_t)
35   Effective group id.

```

1  
2 Host environments are not required to support any specific monitoring attributes.  
▲-----▲

3  
4 **Optional Attributes**  
5

6 The following attributes may be implemented by a host environment.  
7

8 **PMIX\_MONITOR\_ID** "pmix.monitor.id" (char\*)  
9 Provide a string identifier for this request.

10 **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (char\*)  
11 Identifier to be canceled (NULL means cancel all monitoring for this process).

12 **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (bool)  
13 The application desires to control the response to a monitoring event.

14 **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (void)  
15 Register to have the PMIx server monitor the requestor for heartbeats.

16 **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (uint32\_t)  
17 Time in seconds before declaring heartbeat missed.

18 **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (uint32\_t)  
19 Number of heartbeats that can be missed before generating the event.

20 **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (char\*)  
21 Register to monitor file for signs of life.

22 **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (bool)  
23 Monitor size of given file is growing to determine if the application is running.

24 **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (char\*)  
25 Monitor time since last access of given file to determine if the application is running.

26 **PMIX\_MONITOR\_FILE MODIFY** "pmix.monitor.fmod" (char\*)  
27 Monitor time since last modified of given file to determine if the application is running.

28 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)  
29 Time in seconds between checking the file.

30 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)  
31 Number of file checks that can be missed before generating the event.  
▲-----▲

32 **Description**

33 Request that a client be monitored for activity.

## 11.4.23 pmix\_server\_get\_cred\_fn\_t

### 2 Summary

3 Request a credential from the host environment

### 4 Format

PMIx v3.0

C

```
5     typedef pmix_status_t (*pmix_server_get_cred_fn_t) (
6             const pmix_proc_t *proc,
7             const pmix_info_t directives[],
8             size_t ndirs,
9             pmix_credential_cbfunc_t cbfunc,
10            void *cbdata);
```

C

11 IN proc

12 pmix\_proc\_t structure of requesting process (handle)

13 IN directives

14 Array of info structures (array of handles)

15 IN ndirs

16 Number of elements in the *info* array (integer)

17 IN cbfunc

18 Callback function to return the credential ([pmix\\_credential\\_cbfunc\\_t](#) function  
19 reference)

20 IN cbdata

21 Data to be passed to the callback function (memory reference)

22 Returns [PMIX\\_SUCCESS](#), [PMIX\\_ERR\\_NOT\\_SUPPORTED](#) indicating that the host environment  
23 does not support the request (even though the function entry was provided in the server module), or  
24 a negative value corresponding to a PMIx error constant. In the event the function returns an error,  
25 the *cbfunc* will not be called.

### Required Attributes

26 If the PMIx library does not itself provide the requested credential, then it is required to pass any  
27 attributes provided by the client to the host environment for processing. In addition, it must include  
28 the following attributes in the passed *info* array:

29 **PMIX\_USERID** "pmix.euid" (uint32\_t)

30 Effective user id.

31 **PMIX\_GRPID** "pmix.egid" (uint32\_t)

32 Effective group id.

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_CRED\_TYPE** "pmix.sec.ctype" (char\*)

3 When passed in **PMIx\_Get\_credential**, a prioritized, comma-delimited list of desired  
4 credential types for use in environments where multiple authentication mechanisms may be  
5 available. When returned in a callback function, a string identifier of the credential type.

6 **PMIX\_TIMEOUT** "pmix.timeout" (int)

7 Time in seconds before the specified operation should time out (0 indicating infinite) in  
8 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
9 the target process from ever exposing its data.

## Advice to PMIx library implementers

10 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
11 environment due to race condition considerations between completion of the operation versus  
12 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
13 directly in the PMIx server library must take care to resolve the race condition and should avoid  
14 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
15 created.

### Description

16 Request a credential from the host environment

### 11.4.24 pmix\_server\_validate\_cred\_fn\_t

#### Summary

19 Request validation of a credential

## Format

```
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)

1                   Effective user id.  
2 **PMIX\_GRPID** "pmix.egid" (**uint32\_t**)  
3                   Effective group id.

---

5 Host environments are not required to support any specific attributes.

▲-----▼ **Optional Attributes** -----▼

6 The following attributes are optional for host environments that support this operation:

7 **PMIX\_TIMEOUT** "pmix.timeout" (**int**)  
8         Time in seconds before the specified operation should time out (0 indicating infinite) in  
9         error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
10        the target process from ever exposing its data.

▲-----▼ **Advice to PMIx library implementers** -----▼

11 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
12 environment due to race condition considerations between completion of the operation versus  
13 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
14 directly in the PMIx server library must take care to resolve the race condition and should avoid  
15 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
16 created.

17 **Description**

18 Request validation of a credential obtained from the host environment via a prior call to the  
19 **pmix\_server\_get\_cred\_fn\_t** module entry.

20 **11.4.25 pmix\_server\_iоф\_fn\_t**

21 **Summary**

22 Request the specified IO channels be forwarded from the given array of processes.

1      **Format**

C

```
2      typedef pmix_status_t (*pmix_server_iof_fn_t)(
3                           const pmix_proc_t procs[], size_t nprocs,
4                           const pmix_info_t directives[], size_t ndirs,
5                           pmix_iof_channel_t channels,
6                           pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

7      **IN procs**

8      Array **pmix\_proc\_t** identifiers whose IO is being requested (handle)

9      **IN nprocs**

10     Number of elements in *procs* (**size\_t**)

11     **IN directives**

12     Array of **pmix\_info\_t** structures further defining the request (array of handles)

13     **IN ndirs**

14     Number of elements in the *info* array (integer)

15     **IN channels**

16     Bitmask identifying the channels to be forwarded (**pmix\_iof\_channel\_t**)

17     **IN cbfunc**

18     Callback function **pmix\_op\_cbfunc\_t** (function reference)

19     **IN cbdata**

20     Data to be passed to the callback function (memory reference)

21     Returns one of the following:

- 22     • **PMIX\_SUCCESS**, indicating that the request is being processed by the host environment - result  
23       will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
24       function prior to returning from the API.
- 25     • **PMIX\_OPERATION\_SUCCEEDED**, indicating that the request was immediately processed and  
26       returned *success* - the *cbfunc* will not be called
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED**, indicating that the host environment does not support the  
28       request, even though the function entry was provided in the server module - the *cbfunc* will not  
29       be called
- 30     • a PMIx error constant indicating either an error in the input or that the request was immediately  
31       processed and failed - the *cbfunc* will not be called

32     **Required Attributes**

33     The following attributes are required to be included in the passed *info* array:

34     **PMIX\_USERID "pmix.euid" (uint32\_t)**

35       Effective user id.

36     **PMIX\_GRP\_ID "pmix.egid" (uint32\_t)**

1                   Effective group id.  
2

---

3 Host environments that provide this module entry point are required to support the following  
4 attributes:

5 **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csizes" (uint32\_t)

6         The requested size of the PMIx server cache in bytes for each specified channel. By default,  
7         the server is allowed (but not required) to drop all bytes received beyond the max size.

8 **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)

9         In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
10        cache.

11 **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)

12         In an overflow situation, the PMIx server is to drop any new bytes received until room  
13        becomes available in the cache (default).

## Optional Attributes

14 The following attributes may be supported by a host environment.

15 **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)

16         Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
17        specified number of bytes is collected to avoid being called every time a block of IO arrives.  
18         The PMIx tool library will execute the callback and reset the collection counter whenever the  
19        specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
20        the callback upon a call to deregister the respective channel.

21 **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)

22         Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
23        size, this prevents IO from being held indefinitely while waiting for another payload to  
24        arrive.

## Description

25 Request the specified IO channels be forwarded from the given array of processes. An error shall be  
26        returned in the callback function if the requested service from any of the requested processes cannot  
27        be provided.  
28

## Advice to PMIx library implementers

29 The forwarding of stdin is a *push* process - processes cannot request that it be *pulled* from some  
30        other source. Requests including the **PMIX\_FWD\_STDIN\_CHANNEL** channel will return a  
31        **PMIX\_ERR\_NOT\_SUPPORTED** error.

## 11.4.26 pmix\_server\_stdin\_fn\_t

### 2 Summary

3 Pass standard input data to the host environment for transmission to specified recipients.

### 4 Format

PMIx v3.0

C

```
5     typedef pmix_status_t (*pmix_server_stdin_fn_t)(  
6             const pmix_proc_t *source,  
7             const pmix_proc_t targets[],  
8             size_t ntargets,  
9             const pmix_info_t directives[],  
10            size_t ndirs,  
11            const pmix_byte_object_t *bo,  
12            pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

13 IN **source**  
14 pmix\_proc\_t structure of source process (handle)

15 IN **targets**  
16 Array of pmix\_proc\_t target identifiers (handle)

17 IN **ntargets**  
18 Number of elements in the *targets* array (integer)

19 IN **directives**  
20 Array of info structures (array of handles)

21 IN **ndirs**  
22 Number of elements in the *info* array (integer)

23 IN **bo**  
24 Pointer to pmix\_byte\_object\_t containing the payload (handle)

25 IN **cbfunc**  
26 Callback function pmix\_op\_cbfunc\_t (function reference)

27 IN **cbdata**  
28 Data to be passed to the callback function (memory reference)

29 Returns one of the following:

- 30 • PMIX\_SUCCESS , indicating that the request is being processed by the host environment - result  
31 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback  
32 function prior to returning from the API.
- 33 • PMIX\_OPERATION\_SUCCEEDED , indicating that the request was immediately processed and  
34 returned *success* - the *cbfunc* will not be called
- 35 • PMIX\_ERR\_NOT\_SUPPORTED , indicating that the host environment does not support the  
36 request, even though the function entry was provided in the server module - the *cbfunc* will not  
37 be called

- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the `cbfunc` will not be called

## Required Attributes

The following attributes are required to be included in the passed *info* array:

**PMIX\_USERID** "pmix.euid" (uint32\_t)

## Effective user id.

**PMIX\_GRP\_ID** "pmix.egid" (uint32\_t)

Effective group id.

## Description

Passes `stdin` to the host environment for transmission to specified recipients. The host environment is responsible for forwarding the data to all locations that host the specified *targets* and delivering the payload to the PMIx server library connected to those clients.

#### **11.4.27 pmix\_server\_grp\_fn\_t**

## Summary

Request group operations (construct, destruct, etc.) on behalf of a set of processes.

## Format

PMIx v4.0

```
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 source (string)

**IN** **procs**  
Character string identifying the group (string)  
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)

```
1   IN  ndirs  
2     Number of elements in the info array (integer)  
3   IN  cbfunc  
4     Callback function pmix_info_cbfunc_t (function reference)  
5   IN  cbdata  
6     Data to be passed to the callback function (memory reference)  
7  
8 Returns one of the following:  
9  
10  • PMIX_SUCCESS, indicating that the request is being processed by the host environment - result  
11    will be returned in the provided cbfunc. Note that the library must not invoke the callback  
12    function prior to returning from the API.  
13  
14  • PMIX_OPERATION_SUCCEEDED, indicating that the request was immediately processed and  
15    returned success - the cbfunc will not be called  
16  
17  • PMIX_ERR_NOT_SUPPORTED, indicating that the host environment does not support the  
18    request, even though the function entry was provided in the server module - the cbfunc will not  
19    be called  
20  
21  • a PMIx error constant indicating either an error in the input or that the request was immediately  
22    processed and failed - the cbfunc will not be called
```

## Optional Attributes

The following attributes may be supported by a host environment.

**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

**PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

Group operation only involves local processes. PMIx implementations are *required* to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false

**PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (pmix\_byte\_object\_t)

Data collected to be shared during group construction

**PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false

**PMIX\_RANGE** "pmix.range" (**pmix\_data\_range\_t**)  
Value for calls to publish/lookup/unpublish or for monitoring event notifications.

The following attributes may be included in the host's response:

**PMIX\_GROUP\_ID** "pmix.grp.id" (**char\***)  
User-provided group identifier

**PMIX\_GROUP\_MEMBERSHIP** "pmix.grp.mbrs" (**pmix\_data\_array\_t\***)  
Array of group member ID's

**PMIX\_GROUP\_CONTEXT\_ID** "pmix.grp.ctxid" (**size\_t**)  
Context identifier assigned to the group by the host RM.

**PMIX\_GROUP\_ENDPT\_DATA** "pmix.grp.endpt" (**pmix\_byte\_object\_t**)  
Data collected to be shared during group construction

## Description

Perform the specified operation across the identified processes, plus any special actions included in the directives. Return the result of any special action requests in the callback function when the operation is completed. Actions may include a request ([PMIX\\_GROUP\\_ASSIGN\\_CONTEXT\\_ID](#)) that the host assign a unique numerical (size\_t) ID to this group - if given, the [PMIX\\_RANGE](#) attribute will specify the range across which the ID must be unique (default to [PMIX\\_RANGE\\_SESSION](#)).

## 11.4.28 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);
```

```

1   IN requestor
2     pmix_proc_t identifying the requestor (handle)
3   IN op
4     pmix_fabric_operation_t value indicating operation the host is requested to perform
5     (integer)
6   IN directives
7     Array of info structures (array of handles)
8   IN ndirs
9     Number of elements in the info array (integer)
10  IN cbfunc
11    Callback function pmix_info_cbfunc_t (function reference)
12  IN cbdata
13    Data to be passed to the callback function (memory reference)

```

14 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

25 The following directives are required to be supported by all hosts to aid users in identifying the  
26 fabric and (if applicable) the device to whom the operation references:

```

27 PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string)
28   Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)
29 PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)
30   An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)
31 PMIX_FABRIC_PLANE "pmix.fab.plane" (char*)
32   ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
33   for information, specifies the plane whose information is to be returned. When used directly
34   in a request, returns a pmix_data_array_t of string identifiers for all fabric planes in
35   the system.

```

1           **PMIX\_FABRIC\_DEVICE\_INDEX** "pmix.fabdev.idx" (**uint32\_t**)  
2           System-unique index of a particular fabric device (NIC).



3           **Description**

4           Perform the specified operation. Return the result of any requests in the callback function when the  
5           operation is completed. Operations may, for example, include a request for fabric information. See  
6           **pmix\_fabric\_t** for a list of expected information to be included in the response. Note that  
7           requests for device index are to be returned in the callback function's array of **pmix\_info\_t**  
8           using the **PMIX\_FABRIC\_DEVICE\_INDEX** attribute.

## CHAPTER 12

# Fabric Support Definitions

---

As the drive for performance continues, interest has grown in both scheduling algorithms that take into account network locality of the allocated resources, and in optimizing collective communication patterns by structuring them to follow fabric topology. Several interfaces have been defined that are specifically intended to support WLMs (also known as *schedulers*) by providing access to information of potential use to scheduling algorithms - e.g., information on communication costs between different points on the fabric.

In contrast, hierarchical collective operations require each process have global information about both its peers and the fabric. For example, one might aggregate the contribution from all processes on a node, then again across all nodes on a common switch, and finally across all switches. Creating such optimized patterns relies on detailed knowledge of the fabric location of each participant.

PMIx supports these efforts by defining datatypes and attributes by which fabric coordinates for processes and devices can be obtained from the host SMS. When used in conjunction with the PMIx *instant on* methods, this results in the ability of a process to obtain the fabric coordinate of all other processes without incurring additional overhead associated with the publish/exchange of that information.

## 12.1 Fabric Support Constants

The following constants are defined for use in fabric-related events.

**PMIX\_FABRIC\_UPDATE\_PENDING** The PMIx server library has been alerted to a change in the fabric that requires updating of one or more registered `pmix_fabric_t` objects.

**PMIX\_FABRIC\_UPDATED** The PMIx server library has completed updating the entries of all affected `pmix_fabric_t` objects registered with the library. Access to the entries of those objects may now resume.

## 12.2 Fabric Support Datatypes

Several datatype definitions have been created to support fabric-related operations and information.

## 12.2.1 Fabric Coordinate Structure

2 The `pmix_coord_t` structure describes the fabric coordinates of a specified process in a given  
3 view

PMIx v4.0

```
4     typedef struct pmix_coord {
5         char *fabric;
6         char *plane;
7         pmix_coord_view_t view;
8         uint32_t *coord;
9         size_t dims;
10    } pmix_coord_t;
```

11 All coordinate values shall be expressed as unsigned integers due to their units being defined in  
12 fabric devices and not physical distances. The coordinate is therefore an indicator of connectivity  
13 and not relative communication distance.

14 The fabric and plane fields are assigned by the fabric provider to help the user identify the fabric to  
15 which the coordinates refer. Note that providers are not required to assign any particular value to  
16 the fields and may choose to leave the fields blank. Example entries include {"Ethernet", "mgmt"}  
17 or {"infiniband", "data1"}.

### Advice to PMIx library implementers

18 Note that the `pmix_coord_t` structure does not imply nor mandate any requirement on how the  
19 coordinate data is to be stored within the PMIx library. Implementers are free to store the  
20 coordinate in whatever format they choose.

21 A fabric coordinate is usually associated with a given fabric device - e.g., a particular NIC on a  
22 node. Thus, while the fabric coordinate of a device must be unique in a given view, the coordinate  
23 may be shared by multiple processes on a node. If the node contains multiple fabric devices, then  
24 either the device closest to the binding location of a process shall be used as its coordinate, or (if the  
25 process is unbound or its binding is not known) all devices on the node shall be reported as a  
26 `pmix_data_array_t` of `pmix_coord_t` structures.

27 Nodes with multiple fabric devices can also have those devices configured as multiple **fabric**  
28 **planes**. In such cases, a given process (even if bound to a specific location) may be associated  
29 with a coordinate on each plane. The resulting set of fabric coordinates shall be reported as a  
30 `pmix_data_array_t` of `pmix_coord_t` structures. The caller may request a coordinate  
31 from a specific fabric plane by passing the `PMIX_FABRIC_PLANE` attribute as a  
32 directive/qualifier to the `PMIx_Get` or `PMIx_Query_info_nb` call.

## 12.2.2 Fabric Coordinate Support Macros

2 The following macros are provided to support the `pmix_coord_t` structure.

### 12.2.2.1 Initialize the coord structure

4 Initialize the `pmix_coord_t` fields

PMIx v4.0

C

5 `PMIX_COORD_CONSTRUCT(m)`

C

6 IN m

7 Pointer to the structure to be initialized (pointer to `pmix_coord_t`)

### 12.2.2.2 Destruct the coord structure

9 Destruct the `pmix_coord_t` fields

PMIx v4.0

C

10 `PMIX_COORD_DESTRUCT(m)`

C

11 IN m

12 Pointer to the structure to be destructed (pointer to `pmix_coord_t`)

### 12.2.2.3 Create a coord array

14 Allocate and initialize a `pmix_coord_t` array

PMIx v4.0

C

15 `PMIX_COORD_CREATE(m, n)`

C

16 INOUT m

17 Address where the pointer to the array of `pmix_coord_t` structures shall be stored (handle)

18 IN n

19 Number of structures to be allocated (`size_t`)

### 12.2.2.4 Release a coord array

21 Release an array of `pmix_coord_t` structures

PMIx v4.0

C

22 `PMIX_COORD_FREE(m, n)`

C

23 IN m

24 Pointer to the array of `pmix_coord_t` structures (handle)

25 IN n

26 Number of structures in the array (`size_t`)

### 12.2.3 Fabric Coordinate Views

PMIx v4.0

```
2     typedef uint8_t pmix_coord_view_t;
3     #define PMIX_COORD_VIEW_UNDEF      0x00
4     #define PMIX_COORD_LOGICAL_VIEW   0x01
5     #define PMIX_COORD_PHYSICAL_VIEW  0x02
```

C

C

Fabric coordinates can be reported based on different *views* according to user preference at the time of request. The following views have been defined:

- 8 **PMIX\_COORD\_VIEW\_UNDEF** The coordinate view has not been defined.
- 9 **PMIX\_COORD\_LOGICAL\_VIEW** The coordinates are provided in a *logical* view, typically
  - 10 given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by
  - 11 the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains,
  - 12 and other similar factors employed by that fabric.
- 13 **PMIX\_COORD\_PHYSICAL\_VIEW** The coordinates are provided in a *physical* view based on
  - 14 the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative
  - 15 position of that interface on the specific fabric cabling.

#### Advice to PMIx library implementers

PMIx library implementers are advised to avoid declaring the above constants as actual **enum** values in order to allow host environments to add support for possibly proprietary coordinate views.

If the requester does not specify a view, coordinates shall default to the *logical* view.

### 12.2.4 Fabric Link State

The **pmix\_link\_state\_t** is a **uint32\_t** type for fabric link states.

PMIx v4.0

```
22    typedef uint8_t pmix_link_state_t;
```

C

C

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.

- 26 **PMIX\_LINK\_STATE\_UNKNOWN** The port state is unknown or not applicable.
- 27 **PMIX\_LINK\_DOWN** The port is inactive.
- 28 **PMIX\_LINK\_UP** The port is active.

## 12.2.5 Fabric Operation Constants

2 *PMIx v4.0* The `pmix_fabric_operation_t` structure is an enumerated type for specifying fabric  
3 operations used in the PMIx server module's `pmix_server_fabric_fn_t` API. All values  
4 were originally defined in version 4 of the standard unless otherwise marked.

5 **PMIX\_FABRIC\_REQUEST\_INFO** Request information on a specific fabric - if the fabric isn't  
6 specified as per `PMIx_Fabric_register`, then return information on the system default  
7 fabric. Information to be returned is described in `pmix_fabric_t`.

8 **PMIX\_FABRIC\_UPDATE\_INFO** Update information on a specific fabric - the index of the  
9 fabric (`PMIX_FABRIC_INDEX`) to be updated must be provided.

10 **PMIX\_FABRIC\_GET\_VERTEX\_INFO** Request information on a specific NIC within the  
11 identified fabric - the index of the device (`PMIX_FABRIC_DEVICE_INDEX`) and of the  
12 fabric (`PMIX_FABRIC_INDEX`) must be provided. If the NIC identifier is not specified,  
13 then return vertex info on all NICs in the fabric. Information to be included on each vertex is  
14 described in `pmix_fabric_t`.

### Advice to users

15 Requesting information on every NIC in the fabric may be an expensive operation in terms of  
16 both memory footprint and time.

17 **PMIX\_FABRIC\_GET\_DEVICE\_INDEX** Request the fabric-wide index (returned as  
18 `PMIX_FABRIC_DEVICE_INDEX`) for a specific NIC within the identified fabric based on  
19 the provided vertex information. The index of the fabric must be provided.

## 12.2.6 Fabric registration structure

21 The `pmix_fabric_t` structure is used by a WLM to interact with fabric-related PMIx interfaces,  
22 and to provide information about the fabric for use in scheduling algorithms or other purposes.

PMIx v4.0

C

```
23     typedef struct pmix_fabric_s {
24         char *name;
25         size_t index;
26         pmix_info_t *info;
27         size_t ninfo;
28         void *module;
29     } pmix_fabric_t;;
```

1 Note that in this structure:

- 2 • *name* is an optional user-supplied string name identifying the fabric being referenced by this
- 3 struct. If provided, the field must be a **NULL**-terminated string composed of standard
- 4 alphanumeric values supported by common utilities such as *strcmp*;
- 5 • *index* is a PMIx-provided number identifying this object;
- 6 • *info* is an array of **pmix\_info\_t** containing information (provided by the PMIx library) about
- 7 the fabric;
- 8 • *ninfo* is the number of elements in the *info* array
- 9 • *module* points to an opaque object reserved for use by the PMIx server library.

10 Note that only the *name* field is provided by the user - all other fields are provided by the PMIx  
 11 library and must not be modified by the user. The *info* array contains a varying amount of  
 12 information depending upon both the PMIx implementation and information available from the  
 13 fabric vendor. At a minimum, it must contain (ordering is arbitrary):

### Required Attributes

- 14 **PMIX\_FABRIC\_VENDOR** "pmix.fab.vndr" (**string**)  
 15 Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)
- 16 **PMIX\_FABRIC\_IDENTIFIER** "pmix.fab.id" (**string**)  
 17 An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)
- 18 **PMIX\_FABRIC\_NUM\_VERTICES** "pmix.fab.nverts" (**size\_t**)  
 19 Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and  
 20 columns) in the cost matrix

21 and may optionally contain one or more of the following:

### Optional Attributes

- 22 **PMIX\_FABRIC\_COST\_MATRIX** "pmix.fab.cm" (**pointer**)  
 23 Pointer to a two-dimensional array of point-to-point relative communication costs expressed  
 24 as **uint16\_t** values
- 25 **PMIX\_FABRIC\_GROUPS** "pmix.fab.grp" (**string**)  
 26 A string delineating the group membership of nodes in the system, where each fabric group  
 27 consists of the group number followed by a colon and a comma-delimited list of nodes in  
 28 that group, with the groups delimited by semi-colons (e.g.,  
 29 0:node000,node002,node004,node006;1:node001,node003,node005,node007)
- 30 **PMIX\_FABRIC\_DIMS** "pmix.fab.dims" (**uint32\_t**)

1 Number of dimensions in the specified fabric plane/view. If no plane is specified in a  
 2 request, then the dimensions of all planes in the system will be returned as a  
 3 **pmix\_data\_array\_t** containing an array of **uint32\_t** values. Default is to provide  
 4 dimensions in *logical* view.

5 **PMIX\_FABRIC\_PLANE** "pmix.fab.plane" (**char\***)  
 6 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
 7 for information, specifies the plane whose information is to be returned. When used directly  
 8 in a request, returns a **pmix\_data\_array\_t** of string identifiers for all fabric planes in  
 9 the system.

10 **PMIX\_FABRIC\_SHAPE** "pmix.fab.shape" (**pmix\_data\_array\_t\***)  
 11 The size of each dimension in the specified fabric plane/view, returned in a  
 12 **pmix\_data\_array\_t** containing an array of **uint32\_t** values. The size is defined as  
 13 the number of elements present in that dimension - e.g., the number of NICs in one  
 14 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of  
 15 each plane in the system will be returned in an array of fabric shapes. Default is to provide  
 16 the shape in *logical* view.

17 **PMIX\_FABRIC\_SHAPE\_STRING** "pmix.fab.shapestr" (**string**)  
 18 Network shape expressed as a string (e.g., "10x12x2").  
 19 While unusual due to scaling issues, implementations may include an array of  
 20 **PMIX\_FABRIC\_DEVICE** elements describing the vertex information for each NIC in the system.  
 21 Each element shall contain a **pmix\_data\_array\_t** of **pmix\_info\_t** values describing the  
 22 device. Each array may contain one or more of the following (ordering is arbitrary):

23 **PMIX\_FABRIC\_DEVICE\_NAME** "pmix.fabdev.nm" (**string**)  
 24 The operating system name associated with the device. This may be a logical fabric interface  
 25 name (e.g. eth0 or eno1) or an absolute filename.

26 **PMIX\_FABRIC\_DEVICE\_VENDOR** "pmix.fabdev.vndr" (**string**)  
 27 Indicates the name of the vendor that distributes the NIC.

28 **PMIX\_FABRIC\_DEVICE\_ID** "pmix.fabdev.devid" (**string**)  
 29 This is a vendor-provided identifier for the device or product.

30 **PMIX\_HOSTNAME** "pmix.hname" (**char\***)  
 31 Name of the host (e.g., where a specified process is running, or a given device is located).

32 **PMIX\_FABRIC\_DEVICE\_DRIVER** "pmix.fabdev.driver" (**string**)  
 33 The name of the driver associated with the device

34 **PMIX\_FABRIC\_DEVICE\_FIRMWARE** "pmix.fabdev.fmwr" (**string**)  
 35 The device's firmware version

36 **PMIX\_FABRIC\_DEVICE\_ADDRESS** "pmix.fabdev.addr" (**string**)  
 37 The primary link-level address associated with the NIC, such as a Media Access  
 38 Control (MAC) address. If multiple addresses are available, only one will be reported.

```

1   PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
2     The maximum transfer unit of link level frames or packets, in bytes.
3
4   PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
5     The active link data rate, given in bits per second.
6
7   PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )
8     The last available physical port state. Possible values are PMIX_LINK_STATE_UNKNOWN,
9     PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not
10    applicable (unknown), inactive (down), or active (up).
11
12  PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
13    Specifies the type of fabric interface currently active on the device, such as Ethernet or
14    InfiniBand.
15
16  PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
17    The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
18
19  PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
20    A node-level unique identifier for a Peripheral Component Interconnect (PCI) device.
21    Provided only if the device is located on a PCI bus. The identifier is constructed as a
22    four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit
23    device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus, an
24    example identifier might be "abc1:0f:23:01". The combination of node identifier (
25    PMIX_HOSTNAME or PMIX_NODEID) and PMIX_FABRIC_DEVICE_PCI_DEVID
26    shall be unique within the system.

```



## 22 12.2.6.1 Initialize the fabric structure

23 Initialize the **pmix\_fabric\_t** fields

*PMIx v4.0*

24 **PMIX\_FABRIC\_CONSTRUCT** (**m**)

25 **IN m**

26 Pointer to the structure to be initialized (pointer to **pmix\_fabric\_t**)

## 12.3 Fabric Support Attributes

The following attributes are used by the library supporting the system's WLM to either access or return fabric-related information (e.g., as part of the `pmix_fabric_t` structure).

```
4 PMIX_SERVER_SCHEDULER "pmix.srv.sched" (bool)
5     Server requests access to WLM-supporting features - passed solely to the
6     PMIx_server_init API to indicate that the library is to be initialized for scheduler
7     support.
8 PMIX_FABRIC_COST_MATRIX "pmix.fab.cm" (pointer)
9     Pointer to a two-dimensional array of point-to-point relative communication costs expressed
10    as uint16_t values
11 PMIX_FABRIC_GROUPS "pmix.fab.grp" (string)
12    A string delineating the group membership of nodes in the system, where each fabric group
13    consists of the group number followed by a colon and a comma-delimited list of nodes in
14    that group, with the groups delimited by semi-colons (e.g.,
15    0:node000,node002,node004,node006;1:node001,node003,node005,node007)
16    The following attributes may be returned by calls to the scheduler-related APIs or in response to
17    queries (e.g., PMIx_Get or PMIx_Query_info) made by processes or tools.
18 PMIX_FABRIC_VENDOR "pmix.fab.vndr" (string)
19     Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)
20 PMIX_FABRIC_IDENTIFIER "pmix.fab.id" (string)
21     An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)
22 PMIX_FABRIC_INDEX "pmix.fab.idx" (size_t)
23     The index of the fabric as returned in pmix_fabric_t
24 PMIX_FABRIC_NUM_VERTICES "pmix.fab.nverts" (size_t)
25     Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and
26     columns) in the cost matrix
27 PMIX_FABRIC_COORDINATE "pmix.fab.coord" (pmix_data_array_t)
28     Fabric coordinate(s) of the specified process in the view and/or plane provided by the
29     requester. If only one NIC has been assigned to the specified process, then the array will
30     contain only one address. Otherwise, the array will contain the coordinates of all NICs
31     available to the process in order of least to greatest distance from the process (NICs equally
32     distant from the process will be listed in arbitrary order).
33 PMIX_FABRIC_VIEW "pmix.fab.view" (pmix_coord_view_t)
34     Fabric coordinate view to be used for the requested coordinate - see
35     pmix_coord_view_t for the list of accepted values.
36 PMIX_FABRIC_DIMS "pmix.fab.dims" (uint32_t)
37     Number of dimensions in the specified fabric plane/view. If no plane is specified in a
38     request, then the dimensions of all planes in the system will be returned as a
39     pmix_data_array_t containing an array of uint32_t values. Default is to provide
40     dimensions in logical view.
41 PMIX_FABRIC_PLANE "pmix.fab.plane" (char*)
```

1 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request  
 2 for information, specifies the plane whose information is to be returned. When used directly  
 3 in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in  
 4 the system.  
 5 **PMIX\_FABRIC\_SWITCH** "pmix.fab.switch" (`char*`)  
 6 ID string of a fabric switch. When used as a modifier in a request for information, specifies  
 7 the switch whose information is to be returned. When used directly in a request, returns a  
 8 `pmix_data_array_t` of string identifiers for all fabric switches in the system.  
 9 **PMIX\_FABRIC\_ENDP** "pmix.fab.endpt" (`pmix_data_array_t`)  
 10 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given  
 11 process (e.g., in the case where multiple NICs are associated with a package to which the  
 12 process is bound), the returned values will be provided in a `pmix_data_array_t` - the  
 13 returned data type of the individual values in the array varies by fabric provider.  
 14 **PMIX\_FABRIC\_SHAPE** "pmix.fab.shape" (`pmix_data_array_t*`)  
 15 The size of each dimension in the specified fabric plane/view, returned in a  
 16 `pmix_data_array_t` containing an array of `uint32_t` values. The size is defined as  
 17 the number of elements present in that dimension - e.g., the number of NICs in one  
 18 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of  
 19 each plane in the system will be returned in an array of fabric shapes. Default is to provide  
 20 the shape in *logical* view.  
 21 **PMIX\_FABRIC\_SHAPE\_STRING** "pmix.fab.shapestr" (`string`)  
 22 Network shape expressed as a string (e.g., "10x12x2").  
 23 **PMIX\_SWITCH\_PEERS** "pmix.speers" (`string`)  
 24 Comma-delimited string of peers that share the same switch as the process specified in the  
 25 call to `PMIx_Get`. Single-NIC environments will return a string. Multi-NIC environments  
 26 will return an array of results, each element consisting of an array containing the  
 27 `PMIX_FABRIC_DEVICE_INDEX` of the local NIC and the list of peers sharing the switch  
 28 to which that NIC is connected.  
 29 The following attributes are used to describe devices (a.k.a., NICs) attached to the fabric.  
 30 **PMIX\_FABRIC\_DEVICE** "pmix.fabdev" (`pmix_data_array_t`)  
 31 An array of `pmix_info_t` describing a particular fabric device (NIC).  
 32 **PMIX\_FABRIC\_DEVICE\_INDEX** "pmix.fabdev.idx" (`uint32_t`)  
 33 System-unique index of a particular fabric device (NIC).  
 34 **PMIX\_FABRIC\_DEVICE\_NAME** "pmix.fabdev.nm" (`string`)  
 35 The operating system name associated with the device. This may be a logical fabric interface  
 36 name (e.g. eth0 or eno1) or an absolute filename.  
 37 **PMIX\_FABRIC\_DEVICE\_VENDOR** "pmix.fabdev.vndr" (`string`)  
 38 Indicates the name of the vendor that distributes the NIC.  
 39 **PMIX\_FABRIC\_DEVICE\_BUS\_TYPE** "pmix.fabdev.btyp" (`string`)  
 40 The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").  
 41 **PMIX\_FABRIC\_DEVICE\_ID** "pmix.fabdev.devid" (`string`)  
 42 This is a vendor-provided identifier for the device or product.  
 43 **PMIX\_FABRIC\_DEVICE\_DRIVER** "pmix.fabdev.driver" (`string`)

```

1      The name of the driver associated with the device
2      PMIX_FABRIC_DEVICE_FIRMWARE "pmix.fabdev.fmwr" (string)
3      The device's firmware version
4      PMIX_FABRIC_DEVICE_ADDRESS "pmix.fabdev.addr" (string)
5      The primary link-level address associated with the NIC, such as a MAC address. If multiple
6      addresses are available, only one will be reported.
7      PMIX_FABRIC_DEVICE_MTU "pmix.fabdev.mtu" (size_t)
8      The maximum transfer unit of link level frames or packets, in bytes.
9      PMIX_FABRIC_DEVICE_SPEED "pmix.fabdev.speed" (size_t)
10     The active link data rate, given in bits per second.
11     PMIX_FABRIC_DEVICE_STATE "pmix.fabdev.state" ( pmix_link_state_t )
12
13     The last available physical port state. Possible values are PMIX_LINK_STATE_UNKNOWN,
14     PMIX_LINK_DOWN, and PMIX_LINK_UP, to indicate if the port state is unknown or not
15     applicable (unknown), inactive (down), or active (up).
16     PMIX_FABRIC_DEVICE_TYPE "pmix.fabdev.type" (string)
17     Specifies the type of fabric interface currently active on the device, such as Ethernet or
18     InfiniBand.
19     PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)
20     A node-level unique identifier for a PCI device. Provided only if the device is located on a
21     PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
22     the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
23     zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
24     combination of node identifier ( PMIX_HOSTNAME or PMIX_NODEID ) and
25     PMIX_FABRIC_DEVICE_PCI_DEVID shall be unique within the system.

```

## 12.4 Fabric Support Functions

27 The following APIs allow the WLM to request specific services from the fabric subsystem via the  
28 PMIx library.

### Advice to PMIx server hosts

29 Due to their high cost in terms of execution, memory consumption, and interactions with other  
30 SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation  
31 of these APIs be restricted to a single PMIx server in a system that is supporting the SMS  
32 component responsible for the scheduling of allocations (i.e., the system **scheduler**). The  
33 **PMIX\_SERVER\_SCHEDULER** attribute can be used for this purpose to control the execution path.  
34 Clients, tools, and other servers utilizing these functions are advised to have their requests  
35 forwarded to the server supporting the scheduler using the **pmix\_server\_fabric\_fn\_t**  
36 server module function, as needed.

## 12.4.1 PMIx\_Fabric\_register

### Summary

Register for access to fabric-related information.

### Format

PMIx v4.0

```
5     pmix_status_t
6     PMIx_Fabric_register(pmix_fabric_t *fabric,
7                           const pmix_info_t directives[],
8                           size_t ndirs)
```

#### IN fabric

address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at will - PMIx does not utilize this field (handle)

#### IN directives

an optional array of values indicating desired behaviors and/or fabric to be accessed. If `NULL`, then the highest priority available fabric will be used (array of handles)

#### IN ndirs

Number of elements in the *directives* array (integer)

Returns `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

### Required Attributes

The following directives are required to be supported by all PMIx libraries to aid users in identifying the fabric whose data is being sought:

`PMIX_FABRIC_PLANE` "`pmix.fab.plane`" (`char*`)

ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request for information, specifies the plane whose information is to be returned. When used directly in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in the system.

`PMIX_FABRIC_IDENTIFIER` "`pmix.fab.id`" (`string`)

An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)

`PMIX_FABRIC_VENDOR` "`pmix.fab.vndr`" (`string`)

Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)

1           **Description**

2       Register for access to fabric-related information, including the communication cost matrix. This  
3       call must be made prior to requesting information from a fabric. The caller may request access to a  
4       particular fabric using the vendor, type, or identifier, or to a specific **fabric plane** via the  
5       **PMIX\_FABRIC\_PLANE** attribute - otherwise, the default fabric will be returned.

6       For performance reasons, the PMIx library does not provide thread protection for accessing the  
7       information in the **pmix\_fabric\_t** structure. Instead, the PMIx implementation shall provide  
8       two methods for coordinating updates to the provided fabric information:

- 9       • Users may periodically poll for updates using the **PMIx\_Fabric\_update** API
- 10      • Users may register for **PMIX\_FABRIC\_UPDATE\_PENDING** events indicating that an update to  
11       the cost matrix is pending. When received, users are required to terminate or pause any actions  
12       involving access to the cost matrix before returning from the event. Completion of the  
13       **PMIX\_FABRIC\_UPDATE\_PENDING** event handler indicates to the PMIx library that the  
14       fabric object's entries are available for updating. This may include releasing and re-allocating  
15       memory as the number of vertices may have changed (e.g., due to addition or removal of one or  
16       more NICs). When the update has been completed, the PMIx library will generate a  
17       **PMIX\_FABRIC\_UPDATED** event indicating that it is safe to begin using the updated fabric  
18       object(s).

19       There is no requirement that the caller exclusively use either one of these options. For example, the  
20       user may choose to both register for fabric update events, but poll for an update prior to some  
21       critical operation.

22     **12.4.2 PMIx\_Fabric\_register\_nb**

23       **Summary**

24       Register for access to fabric-related information.

25       **Format**

PMIx v4.0

```
26       pmix_status_t
27       PMIx_Fabric_register_nb(pmix_fabric_t *fabric,
28                                   const pmix_info_t directives[],
29                                   size_t ndirs,
30                                   pmix_op_cfunc_t cbfunc, void *cbdata)
```

31       **IN    fabric**

32       address of a **pmix\_fabric\_t** (backed by storage). User may populate the "name" field at  
33       will - PMIx does not utilize this field (handle)

34       **IN    directives**

35       an optional array of values indicating desired behaviors and/or fabric to be accessed. If **NULL**,  
36       then the highest priority available fabric will be used (array of handles)

```
1   IN  ndirs  
2     Number of elements in the directives array (integer)  
3   IN  cbfunc  
4     Callback function pmix_op_cbfunc_t (function reference)  
5   IN  cbdata  
6     Data to be passed to the callback function (memory reference)  
7  
8 Returns one of the following:  
9  
10  • PMIX_SUCCESS indicating that the request has been accepted for processing and the provided  
11    callback function will be executed upon completion of the operation. Note that the library must  
12    not invoke the callback function prior to returning from the API.  
13  
14  • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this  
15    case, the provided callback function will not be executed
```

### 13 **Description**

```
14 Non-blocking form of PMIx_Fabric_register. The caller is not allowed to access the  
15  provided pmix_fabric_t until the callback function has been executed, at which time the  
16  fabric information will have been loaded into the provided structure.
```

## 17 **12.4.3 PMIx\_Fabric\_update**

### 18 **Summary**

```
19 Update fabric-related information.
```

### 20 **Format**

PMIx v4.0

```
21  pmix_status_t  
22  PMIx_Fabric_update(pmix_fabric_t *fabric)
```

```
23  IN  fabric  
24    address of a pmix_fabric_t (backed by storage) (handle)
```

```
25 Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.
```

### 26 **Description**

```
27 Update fabric-related information. This call can be made at any time to request an update of the  
28  fabric information contained in the provided pmix_fabric_t object. The caller is not allowed  
29  to access the provided pmix_fabric_t until the call has returned.
```

## 30 **12.4.4 PMIx\_Fabric\_update\_nb**

### 31 **Summary**

```
32 Update fabric-related information.
```

1           **Format**

2            pmix\_status\_t  
3            PMIx\_Fabric\_update\_nb(pmix\_fabric\_t \*fabric,  
4                                pmix\_op\_cbfunc\_t cbfunc, void \*cbdata)

5            **IN    fabric**  
6            address of a **pmix\_fabric\_t** (handle)  
7            **IN    cbfunc**  
8            Callback function **pmix\_op\_cbfunc\_t** (function reference)  
9            **IN    cbdata**  
10          Data to be passed to the callback function (memory reference)

11         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

17           **Description**

18         Non-blocking form of **PMIx\_Fabric\_update**. The caller is not allowed to access the provided  
19         **pmix\_fabric\_t** until the callback function has been executed.

20         **12.4.5   PMIx\_Fabric\_deregister**

21           **Summary**

22         Deregister a fabric object.

23           **Format**

24            pmix\_status\_t PMIx\_Fabric\_deregister(pmix\_fabric\_t \*fabric)

25            **IN    fabric**  
26            address of a **pmix\_fabric\_t** (handle)

27         Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

28           **Description**

29         Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information  
30         (e.g., cost matrix) associated with it. Contents of the provided **pmix\_fabric\_t** will be  
31         invalidated upon function return.

## 12.4.6 PMIx\_Fabric\_deregister\_nb

### Summary

Deregister a fabric object.

### Format

PMIx v4.0

C

```
5 pmix_status_t PMIx_Fabric_deregister_nb(pmix_fabric_t *fabric,
6                                     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

7 IN **fabric**

8 address of a **pmix\_fabric\_t** (handle)

9 IN **cbfunc**

10 Callback function **pmix\_op\_cbfunc\_t** (function reference)

11 IN **cbdata**

12 Data to be passed to the callback function (memory reference)

13 Returns one of the following:

- 14 • **PMIX\_SUCCESS** indicating that the request has been accepted for processing and the provided  
15 callback function will be executed upon completion of the operation. Note that the library must  
16 not invoke the callback function prior to returning from the API.
- 17 • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this  
18 case, the provided callback function will not be executed

### Description

20 Non-blocking form of **PMIx\_Fabric\_deregister**. Provided *fabric* must not be accessed  
21 until after callback function has been executed.

## 12.4.7 PMIx\_Fabric\_get\_vertex\_info

### Summary

23 Given a communication cost matrix index for a specified fabric, return the corresponding vertex  
24 info.

### Format

PMIx v4.0

C

```
27 pmix_status_t
28 PMIx_Fabric_get_vertex_info(pmix_fabric_t *fabric, uint32_t index,
29                               pmix_info_t **info, size_t *ninfo)
```

```

1 IN  fabric          address of a pmix_fabric_t (handle)
2 IN  index           vertex index (i.e., communication cost matrix row or column number) (integer)
3
4 INOUT info        Address where a pointer to an array of pmix_info_t containing the results of the query
5      can be returned (memory reference)
6 INOUT ninfo       Address where the number of elements in info can be returned (handle)
7
8 Returns one of the following:
9
10 • PMIX_SUCCESS, indicating return of a valid value.
11 • PMIX_ERR_BAD_PARAM, indicating that the provided index is out of bounds.
12
13 • a PMIx error constant indicating either an error in the input or that the request failed.

```

## 14 Description

15 Query information about a specified vertex (fabric device, or NIC) in the system. The returned  
16 *status* indicates if requested data was found or not. The returned array of **pmix\_info\_t** will  
17 contain information on the specified vertex - the exact contents will depend on the PMIx  
18 implementation and the fabric vendor. At a minimum, it must contain sufficient information to  
19 uniquely identify the device within the system (ordering is arbitrary):

### 20 Required Attributes

```

21 PMIX_HOSTNAME "pmix.hname" (char*)
22     Name of the host (e.g., where a specified process is running, or a given device is located).
23     The PMIX_NODEID may be returned in its place, or in addition to the hostname.
24
25 PMIX_FABRIC_DEVICE_NAME "pmix.fabdev.nm" (string)
26     The operating system name associated with the device. This may be a logical fabric interface
27     name (e.g. eth0 or eno1) or an absolute filename.
28
29 PMIX_FABRIC_DEVICE_VENDOR "pmix.fabdev.vndr" (string)
30     Indicates the name of the vendor that distributes the NIC.
31
32 PMIX_FABRIC_DEVICE_BUS_TYPE "pmix.fabdev.btyp" (string)
33     The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
34
35 PMIX_FABRIC_DEVICE_PCI_DEVID "pmix.fabdev.pcidevid" (string)

```

1 A node-level unique identifier for a PCI device. Provided only if the device is located on a  
2 PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of  
3 the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in  
4 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The  
5 combination of node identifier ( **PMIX\_HOSTNAME** or **PMIX\_NODEID** ) and  
6 **PMIX\_FABRIC\_DEVICE\_PCI\_DEVID** shall be unique within the system. This item  
7 should be included if the device bus type is PCI - the equivalent should be provided for any  
8 other bus type.

9 The returned array may optionally contain one or more of the following:

#### Optional Attributes

10 **PMIX\_FABRIC\_DEVICE\_ID** "pmix.fabdev.devid" (**string**)

11 This is a vendor-provided identifier for the device or product.

12 **PMIX\_FABRIC\_DEVICE\_DRIVER** "pmix.fabdev.driver" (**string**)

13 The name of the driver associated with the device

14 **PMIX\_FABRIC\_DEVICE\_FIRMWARE** "pmix.fabdev.fmwr" (**string**)

15 The device's firmware version

16 **PMIX\_FABRIC\_DEVICE\_ADDRESS** "pmix.fabdev.addr" (**string**)

17 The primary link-level address associated with the NIC, such as a MAC address. If multiple  
18 addresses are available, only one will be reported.

19 **PMIX\_FABRIC\_DEVICE\_MTU** "pmix.fabdev.mtu" (**size\_t**)

20 The maximum transfer unit of link level frames or packets, in bytes.

21 **PMIX\_FABRIC\_DEVICE\_SPEED** "pmix.fabdev.speed" (**size\_t**)

22 The active link data rate, given in bits per second.

23 **PMIX\_FABRIC\_DEVICE\_STATE** "pmix.fabdev.state" ( **pmix\_link\_state\_t** )

24 The last available physical port state. Possible values are **PMIX\_LINK\_STATE\_UNKNOWN**,  
25 **PMIX\_LINK\_DOWN**, and **PMIX\_LINK\_UP**, to indicate if the port state is unknown or not  
26 applicable (unknown), inactive (down), or active (up).

27 **PMIX\_FABRIC\_DEVICE\_TYPE** "pmix.fabdev.type" (**string**)

28 Specifies the type of fabric interface currently active on the device, such as Ethernet or  
29 InfiniBand.

30 The caller is responsible for releasing the returned array.

#### 12.4.8 PMIx\_Fabric\_get\_vertex\_info\_nb

## Summary

Given a communication cost matrix index for a specified fabric, return the corresponding vertex info.

## Format

PMIx v4.0

```
pmix_status_t  
PMIx_Fabric_get_vertex_info_nb(pmix_fabric_t *fabric, uint32_t index,  
                                pmix_info_cbfnc_t cbfunc, void *cbdata)
```

IN fabric

address of a **pmix\_fabric\_t** (handle)

IN index

vertex index (i.e., communication cost matrix row or column number) (integer)

## IN cbfunc

Callback function **pmix\_info\_cbfunc\_t** (function reference)

IN chdata

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\\_get\\_vertex\\_info](#). Data will be returned in the provided callback function.

## **12.4.9 PMIx\_Fabric\_get\_device\_index**

## Summary

Given vertex info, return the corresponding communication cost matrix index.

1           **Format**

2        `pmix_status_t`  
3        `PMIx_Fabric_get_device_index(pmix_fabric_t *fabric,`  
4                            `const pmix_info_t vertex[], size_t ninfo,`  
5                            `uint32_t *index)`

C

C

6        **IN**`fabric`

7           address of a `pmix_fabric_t` (handle)

8        **IN**`vertex`

9           array of `pmix_info_t` containing info describing the vertex whose index is being queried  
10           (handle)

11       **IN**`ninfo`

12           number of elements in *vertex*

13       **OUT**`index`

14           pointer to the location where the index is to be returned (memory reference (handle))

15       Returns one of the following:

16       • `PMIX_SUCCESS`, indicating return of a valid value.

17       • a PMIx error constant indicating either an error in the input or that the request failed.

18           **Description**

19       Query the index number of a vertex corresponding to the provided description. The description  
20       must provide adequate information to uniquely identify the target vertex. At a minimum, this must  
21       include identification of the node hosting the device using either the `PMIX_HOSTNAME` or  
22       `PMIX_NODEID`, plus a node-level unique identifier for the device (e.g., the  
23       `PMIX_FABRIC_DEVICE_PCI_DEVID` for a PCI device).

24   **12.4.10   PMIx\_Fabric\_get\_device\_index\_nb**

25           **Summary**

26       Given vertex info, return the corresponding communication cost matrix index.

27           **Format**

PMIx v4.0

C

28        `pmix_status_t`  
29        `PMIx_Fabric_get_device_index_nb(pmix_fabric_t *fabric,`  
30                            `const pmix_info_t vertex[], size_t ninfo,`  
31                            `pmix_info_cbfunc_t cbfunc, void *cbdata)`

```
1   IN  fabric
2     address of a pmix_fabric_t (handle)
3   IN  vertex
4     array of pmix_info_t containing info describing the vertex whose index is being queried
5     (handle)
6   IN  ninfo
7     number of elements in vertex
8   IN  cbfunc
9     Callback function pmix_info_cbfunc_t (function reference)
10  IN  cldata
11    Data to be passed to the callback function (memory reference)
12
13 Returns one of the following:
14
15  • PMIX_SUCCESS indicating that the request has been accepted for processing and the provided
16    callback function will be executed upon completion of the operation. Note that the library must
17    not invoke the callback function prior to returning from the API.
18
19  • a non-zero PMIx error constant indicating a reason for the request to have been rejected. In this
20    case, the provided callback function will not be executed
```

## 18      **Description**

19      Non-blocking form of **PMIx\_Fabric\_get\_device\_index**. Index will be returned in the  
20      provided callback function via the **PMIX\_FABRIC\_INDEX** attribute.

## CHAPTER 13

# Process Sets and Groups

PMIx supports two slightly related, but functionally different concepts known as *process sets* and *process groups*. This chapter these two concepts and describes how they are utilized, along with their corresponding APIs.

## 13.1 Process Sets

A PMIx *Process Set* is a user-provided label associated with a given set of application processes. Processes can belong to multiple process *sets* at a time. Definition of a PMIx process set typically occurs at time of application execution - e.g., on a command line:

```
$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp
```

In this example, the processes in the first application will be labeled with a **PMIX\_PSET\_NAME** attribute of *ocean* while those in the second application will be labeled with an *ice* value. During the execution, application processes could lookup the process set attribute for any other process using **PMIx\_Get**. Alternatively, other executing applications could utilize the **PMIx\_Query\_info\_nb** API to obtain the number of declared process sets in the system, a list of their names, and other information about them. In other words, the *process set* identifier provides a label by which an application can derive information about a process and its application - it does *not*, however, confer any operational function.

Thus, process *sets* differ from process *groups* in several key ways:

- Process *sets* have no implied relationship between their members - i.e., a process in a process set has no concept of a “pset rank” as it would in a process *group*
- Process *set* identifiers are considered job-level information set at launch. No PMIx API is provided by which a user can change the process *set* value of a process on-the-fly. In contrast, PMIx process *groups* can only be defined dynamically by the application.
- Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are involved in an operation are translated by their PMIx server into their *native* identifier prior to the operation being passed to the host environment. For example, an application can define a process group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the group id of *foo*. If the application subsequently calls the **PMIx\_Fence** API with a process

1 identifier of {foo, PMIX\_RANK\_WILDCARD}, the PMIx server will replace that identifier  
2 with an array consisting of {210456, 0} and {210456, 1} - the host-assigned identifiers of the  
3 participating processes - prior to passing the request up to the host environment

- 4 • Process *groups* can request that the host environment assign a unique **size\_t** PGCID to the  
5 group at time of group construction. An MPI library may, for example, use the PGCID as the  
6 MPI communicator identifier for the group.

7 The two concepts do, however, overlap in one specific area. Process *groups* are included in the  
8 process *set* information returned by calls to **PMIx\_Query\_info\_nb**. Thus, a *process group* can  
9 effectively be considered an extended version of a *process set* that adds dynamic definition and  
10 operational context to the *process set* concept.

### Advice to PMIx library implementers

11 PMIx implementations are required to include all active *group* identifiers in the returned list of  
12 process *set* names provided in response to the appropriate **PMIx\_Query\_info\_nb** call.

## 13.2 Process Groups

14 PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for  
15 operational purposes such as passing events or participating in PMIx fence operations. As with  
16 processes that assemble via **PMIx\_Connect**, each member of the group is provided with both the  
17 job-level information of any other namespace represented in the group, and the contact information  
18 for all group members. However, *groups* differ from **PMIx\_Connect** assemblages in the  
19 following key areas:

- 20 • Relation to the host environment
- 21 – Calls to **PMIx\_Connect** are relayed to the host environment. This means that the host RM  
22 should treat the failure of any process in the specified assemblage as a reportable event and  
23 take appropriate action. However, the environment is not required to define a new identifier for  
24 the connected assemblage or any of its member processes, nor does it define a new rank for  
25 each process within that assemblage. In addition, the PMIx server does not provide any  
26 tracking support for the assemblage. Thus, the caller is responsible for addressing members of  
27 the connected assemblage using their RM-provided identifiers.
- 28 – Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed,  
29 the server creates a tracker that associates the specified processes with the user-provided group  
30 identifier, and assigns a new *group rank* based on their relative position in the array of  
31 processes provided in the call to **PMIx\_Group\_construct**. Members of the group can  
32 subsequently utilize the group identifier in PMIx function calls to address the group's  
33 members, using either **PMIX\_RANK\_WILDCARD** to refer to all of them or the group-level  
34 rank of specific members. The PMIx server will translate the specified processes into their

1 RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment  
2 has no visibility into the group's existence or membership.

## Advice to users

3 User-provided group identifiers must be distinct from anything provided by the RM so as to  
4 avoid collisions between group identifiers and RM-assigned namespaces. This can usually be  
5 accomplished through the use of an application-specific prefix – e.g., “myapp-foo”

### 6 • Construction procedure

- 7 – **PMIx\_Connect** calls require that every process call the API before completing – i.e., it is  
8 modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a  
9 given application thread can only be involved in one connect/accept operation at a time, and is  
10 blocked in that operation until all specified processes participate. In addition, there is no  
11 provision for replacing processes in the assemblage due to failure to participate, nor a  
12 mechanism by which a process might decline participation.
- 13 – PMIx Groups are designed to be more flexible in their construction procedure by relaxing  
14 these constraints. While a standard blocking form of constructing groups is provided, the event  
15 notification system is utilized to provide a designated *group leader* with the ability to replace  
16 participants that fail to participate within a given timeout period. This provides a mechanism  
17 by which the application can, if desired, replace members on-the-fly or allow the group to  
18 proceed with partial membership. In such cases, the final group membership is returned to all  
19 participants upon completion of the operation.

20 Additionally, PMIx supports dynamic definition of group membership based on an invite/join  
21 model. A process can asynchronously initiate construction of a group of any processes via the  
22 **PMIx\_Group\_invite** function call. Invitations are delivered via a PMIx event (using the  
23 **PMIX\_GROUP\_INVITED** event) to the invited processes which can then either accept or  
24 decline the invitation using the **PMIx\_Group\_join** API. The initiating process tracks  
25 responses by registering for the events generated by the call to **PMIx\_Group\_join**,  
26 timeouts, or process terminations, optionally replacing processes that decline the invitation,  
27 fail to respond in time, or terminate without responding. Upon completion of the operation,  
28 the final list of participants is communicated to each member of the new group.

### 29 • Destruct procedure

- 30 – Processes that assemble via **PMIx\_Connect** must all depart the assemblage together – i.e.,  
31 no member can depart the assemblage while leaving the remaining members in it. Even the  
32 non-blocking form of **PMIx\_Disconnect** retains this requirement in that members remain  
33 a part of the assemblage until all members have called **PMIx\_Disconnect\_nb**

- 1 – Members of a PMIx Group may depart the group at any time via the **PMIx\_Group\_leave**  
2 API. Other members are notified of the departure via the **PMIX\_GROUP\_LEFT** event to  
3 distinguish such events from those reporting process termination. This leaves the remaining  
4 members free to continue group operations. The **PMIx\_Group\_destruct** operation offers  
5 a collective method akin to **PMIx\_Disconnect** for deconstructing the entire group.

6 Note that applications supporting dynamic group behaviors such as asynchronous departure  
7 take responsibility for ensuring global consistency in the group definition prior to executing  
8 group collective operations - i.e., it is the application's responsibility to either ensure that  
9 knowledge of the current group membership is globally consistent across the participants, or to  
10 register for appropriate events to deal with the lack of consistency during the operation.

11 In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected*  
12 when constructed via **PMIx\_Connect**. The relevant APIs are explained below.

#### Advice to users

13 The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs  
14 must register for the corresponding events. Failure to do so will likely lead to operational failures.  
15 Users are recommended to utilize the **PMIX\_TIMEOUT** directive (or retain an internal timer) on  
16 calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have  
17 not registered for required events will never respond.

### 13.2.1 Group Operation Constants

19 *PMIx v4.0* The **pmix\_group\_operation\_t** structure is an enumerated type for specifying group  
20 operations. All values were originally defined in version 4 of the standard unless otherwise marked.

21 **PMIX\_GROUP\_DECLINE** Decline an invitation to join a PMIx group - provided for readability  
22 of user code  
23 **PMIX\_GROUP\_ACCEPT** Accept an invitation to join a PMIx group - provided for readability  
24 of user code  
25 **PMIX\_GROUP\_CONSTRUCT** Construct a group composed of the specified processes - used by  
26 a PMIx server library to direct host operation  
27 **PMIX\_GROUP\_DESTRUCT** Destruct the specified group - used by a PMIx server library to  
28 direct host operation

### 13.2.2 PMIx\_Group\_construct

#### Summary

Construct a PMIx process group

1      **Format**

C

```
2      pmix_status_t  
3      PMIx_Group_construct(const char grp[],  
4                        const pmix_proc_t procs[], size_t nprocs,  
5                        const pmix_info_t directives[], size_t ndirs,  
6                        pmix_info_t **results, size_t *nresults)
```

7      **IN    grp**

8       NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
9       group identifier (string)

10     **IN    procs**

11       Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the member processes  
12       (array of handles)

13     **IN    nprocs**

14       Number of elements in the *procs* array (**size\_t**)

15     **IN    directives**

16       Array of **pmix\_info\_t** structures (array of handles)

17     **IN    ndirs**

18       Number of elements in the *directives* array (**size\_t**)

19     **INOUT results**

20       Pointer to a location where the array of **pmix\_info\_t** describing the results of the  
21       operation is to be returned (pointer to handle)

22     **INOUT nresults**

23       Pointer to a **size\_t** location where the number of elements in *results* is to be returned  
24       (memory reference)

25       Returns one of the following:

- 26       • **PMIX\_SUCCESS** , indicating that the request has been successfully completed
- 27       • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this  
28       operation
- 29       • a PMIx error constant indicating either an error in the input or that the request failed to be  
30       completed

▼----- Required Attributes -----▼

31       The following attributes are *required* to be supported by all PMIx libraries that support this  
32       operation:

33       **PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (**bool**)

34           This process is the leader of the group

35       **PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (**bool**)

1           Participation is optional - do not return an error if any of the specified processes terminate  
2           without having joined. The default is false

3           **PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

4           Group operation only involves local processes. PMIx implementations are *required* to  
5           automatically scan an array of group members for local vs remote processes - if only local  
6           processes are detected, the implementation need not execute a global collective for the  
7           operation unless a context ID has been requested from the host environment. This can result  
8           in significant time savings. This attribute can be used to optimize the operation by indicating  
9           whether or not only local processes are represented, thus allowing the implementation to  
10          bypass the scan. The default is false

11          Host environments that support this operation are *required* to provide the following attributes:

12           **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

13          Requests that the RM assign a new context identifier to the newly created group. The  
14          identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
15          specified in the request. Thus, the value serves as a means of identifying the group within  
16          that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

17           **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

18          Notify remaining members when another member terminates without first leaving the group.  
19          The default is false

▲-----▼           **Optional Attributes**           ▼-----▲

20          The following attributes are optional for host environments that support this operation:

21           **PMIX\_TIMEOUT** "pmix.timeout" (int)

22          Time in seconds before the specified operation should time out (0 indicating infinite) in  
23          error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
24          the target process from ever exposing its data.

▲-----▼           **Advice to PMIx library implementers**           ▼-----▲

25          We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
26          environment due to race condition considerations between completion of the operation versus  
27          internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
28          directly in the PMIx server library must take care to resolve the race condition and should avoid  
29          passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
30          created.

## 1           Description

2           Construct a new group composed of the specified processes and identified with the provided group  
3           identifier. The group identifier is a user-defined, **NULL**-terminated character array of length less  
4           than or equal to **PMIX\_MAX\_NSLEN**. Only characters accepted by standard string comparison  
5           functions (e.g., *strncmp*) are supported. Processes may engage in multiple simultaneous group  
6           construct operations so long as each is provided with a unique group ID. The *directives* array can be  
7           used to pass user-level directives regarding timeout constraints and other options available from the  
8           PMIx server.

9           If the **PMIX\_GROUP\_NOTIFY\_TERMINATION** attribute is provided and has a value of **true**,  
10          then either the construct leader (if **PMIX\_GROUP\_LEADER** is provided) or all participants who  
11          register for the **PMIX\_GROUP\_MEMBER FAILED** event will receive events whenever a process  
12          fails or terminates prior to calling **PMIx\_Group\_construct** – i.e. if a *group leader* is  
13          declared, *only* that process will receive the event. In the absence of a declared leader, *all* specified  
14          group members will receive the event.

15          The event will contain the identifier of the process that failed to join plus any other information that  
16          the host RM provided. This provides an opportunity for the leader or the collective members to  
17          react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The  
18          decision is communicated to the PMIx library in the results array at the end of the event handler.  
19          This allows PMIx to properly adjust accounting for procedure completion. When construct is  
20          complete, the participating PMIx servers will be alerted to any change in participants and each  
21          group member will receive an updated group membership (marked with the  
22          **PMIX\_GROUP\_MEMBERSHIP** attribute) as part of the *results* array returned by this API.

23          Failure of the declared leader at any time will cause a **PMIX\_GROUP LEADER FAILED** event to  
24          be delivered to all participants so they can optionally declare a new leader. A new leader is  
25          identified by providing the **PMIX\_GROUP LEADER** attribute in the results array in the return of  
26          the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the  
27          new leader. Results of the leader selection will be communicated to all participants via a  
28          **PMIX\_GROUP LEADER SELECTED** event identifying the new leader. If no leader was selected,  
29          then the **pmix\_info\_t** provided to that event handler will include that information so the  
30          participants can take appropriate action.

31          Any participant that returns **PMIX\_GROUP CONSTRUCT ABORT** from either the  
32          **PMIX\_GROUP MEMBER FAILED** or the **PMIX\_GROUP LEADER FAILED** event handler will  
33          cause the construct process to abort, returning from the call with a  
34          **PMIX\_GROUP CONSTRUCT ABORT** status.

35          If the **PMIX\_GROUP NOTIFY TERMINATION** attribute is not provided or has a value of  
36          **false**, then the **PMIx\_Group\_construct** operation will simply return an error whenever a  
37          proposed group member fails or terminates prior to calling **PMIx\_Group\_construct**.

38          Providing the **PMIX\_GROUP OPTIONAL** attribute with a value of **true** directs the PMIx library  
39          to consider participation by any specified group member as non-required - thus, the operation will  
40          return **PMIX\_SUCCESS** if all members participate, or **PMIX\_ERR\_PARTIAL\_SUCCESS** if

1 some members fail to participate. The *results* array will contain the final group membership in the  
2 latter case. Note that this use-case can cause the operation to hang if the **PMIX\_TIMEOUT**  
3 attribute is not specified and one or more group members fail to call **PMIx\_Group\_construct**  
4 while continuing to execute. Also, note that no leader or member failed events will be generated  
5 during the operation.

6 Processes in a group under construction are not allowed to leave the group until group construction  
7 is complete. Upon completion of the construct procedure, each group member will have access to  
8 the job-level information of all namespaces represented in the group plus any information posted  
9 via **PMIx\_Put** (subject to the usual scoping directives) for every group member.

### Advice to PMIx library implementers

10 At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related  
11 information from each participating namespace plus any information posted by group members via  
12 **PMIx\_Put** (subject to scoping directives) is available to each member via calls to **PMIx\_Get**.

### Advice to PMIx server hosts

13 The collective nature of this API generally results in use of a fence-like operation by the backend  
14 host environment. Host environments that utilize the array of process participants as a *signature* for  
15 such operations may experience potential conflicts should both a **PMIx\_Group\_construct**  
16 and a **PMIx\_Fence** operation involving the same participants be simultaneously executed. As  
17 PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve  
18 any potential conflicts.

## 13.2.3 **PMIx\_Group\_construct\_nb**

### Summary

Non-blocking form of **PMIx\_Group\_construct**

1           **Format**2       *PMIx v4.0*

C

```
3       pmix_status_t
4       PMIx_Group_construct_nb(const char grp[],  
5                            const pmix_proc_t procs[], size_t nprocs,  
6                            const pmix_info_t directives[], size_t ndirs,  
                          pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

7       **IN** **grp**

8       NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
9       group identifier (string)

10      **IN** **procs**

11      Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the member processes  
12      (array of handles)

13      **IN** **nprocs**

14      Number of elements in the *procs* array (**size\_t**)

15      **IN** **directives**

16      Array of **pmix\_info\_t** structures (array of handles)

17      **IN** **ndirs**

18      Number of elements in the *directives* array (**size\_t**)

19      **IN** **cbfunc**

20      Callback function **pmix\_info\_cbfunc\_t** (function reference)

21      **IN** **cbdata**

22      Data to be passed to the callback function (memory reference)

23      Returns one of the following:

- **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

33      If executed, the status returned in the provided callback function will be one of the following  
34      constants:

- **PMIX\_SUCCESS** The operation succeeded and all specified members participated.

- **PMIX\_ERR\_PARTIAL\_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function
- **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request's failure

### Required Attributes

PMIx libraries that choose not to support this operation *must* return **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called.

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

**PMIX\_GROUP\_LEADER** "pmix.grp.ldr" (bool)

This process is the leader of the group

**PMIX\_GROUP\_OPTIONAL** "pmix.grp.opt" (bool)

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false

**PMIX\_GROUP\_LOCAL\_ONLY** "pmix.grp.lcl" (bool)

Group operation only involves local processes. PMIx implementations are *required* to automatically scan an array of group members for local vs remote processes - if only local processes are detected, the implementation need not execute a global collective for the operation unless a context ID has been requested from the host environment. This can result in significant time savings. This attribute can be used to optimize the operation by indicating whether or not only local processes are represented, thus allowing the implementation to bypass the scan. The default is false

Host environments that support this operation are *required* to provide the following attributes:

**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

**PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

Notify remaining members when another member terminates without first leaving the group. The default is false

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

## Advice to PMIx library implementers

6 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
7 environment due to race condition considerations between completion of the operation versus  
8 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
9 directly in the PMIx server library must take care to resolve the race condition and should avoid  
10 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
11 created.

## Description

13 Non-blocking version of the **PMIx\_Group\_construct** operation. The callback function will  
14 be called once all group members have called either **PMIx\_Group\_construct** or  
15 **PMIx\_Group\_construct\_nb**.

### 16 13.2.4 **PMIx\_Group\_destruct**

#### 17 Summary

18 Destruct a PMIx process group

1           **Format**

2        **pmix\_status\_t**  
3        **PMIx\_Group\_destruct**(const char grp[],  
4                            const pmix\_info\_t directives[], size\_t ndirs)

5        **IN**    **grp**

6            NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
7            identifier of the group to be destructed (string)

8        **IN**    **directives**

9            Array of **pmix\_info\_t** structures (array of handles)

10      **IN**    **ndirs**

11        Number of elements in the *directives* array (**size\_t**)

12        Returns one of the following:

- 13        • **PMIX\_SUCCESS**, indicating that the request has been successfully completed
- 14        • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this  
15            operation
- 16        • a PMIx error constant indicating either an error in the input or that the request failed to be  
17            completed

18           **Required Attributes**

19        For implementations and host environments that support the operation, there are no identified  
required attributes for this API.

20           **Optional Attributes**

21        The following attributes are optional for host environments that support this operation:

22        **PMIX\_TIMEOUT "pmix.timeout" (int)**

23           Time in seconds before the specified operation should time out (0 indicating infinite) in  
24           error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
         the target process from ever exposing its data.

## Advice to PMIx library implementers

We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

### Description

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling **PMIx\_Group\_destruct** or its non-blocking version unless the **PMIX\_GROUP\_NOTIFY\_TERMINATION** attribute was provided (with a value of **false**) at time of group construction. If notification was requested, then the **PMIX\_GROUP\_MEMBER\_FAILED** event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The **PMIx\_Group\_destruct** operation will subsequently return **PMIX\_SUCCESS** when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

## Advice to PMIx server hosts

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a **PMIx\_Group\_destruct** and a **PMIx\_Fence** operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

### 13.2.5 **PMIx\_Group\_destruct\_nb**

#### Summary

Non-blocking form of **PMIx\_Group\_destruct**

1           **Format**

2        *PMIx v4.0*

C

```
3        pmix_status_t
4        PMIx_Group_destruct_nb(const char grp[],  
5                            const pmix_info_t directives[], size_t ndirs,  
6                            pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

6        **IN**    **grp**

7           NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
8           identifier of the group to be destructed (string)

9        **IN**    **directives**

10          Array of **pmix\_info\_t** structures (array of handles)

11        **IN**    **ndirs**

12          Number of elements in the *directives* array (**size\_t**)

13        **IN**    **cbfunc**

14          Callback function **pmix\_op\_cbfunc\_t** (function reference)

15        **IN**    **cbdata**

16          Data to be passed to the callback function (memory reference)

17          Returns one of the following:

- **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

27          If executed, the status returned in the provided callback function will be one of the following  
28          constants:

- **PMIX\_SUCCESS** The operation was successfully completed
- **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM  
31          does not.
- a non-zero PMIx error constant indicating a reason for the request's failure

## Required Attributes

1 PMIx libraries that choose not to support this operation *must* return  
2 **PMIX\_ERR\_NOT\_SUPPORTED** when the function is called. For implementations and host  
3 environments that support the operation, there are no identified required attributes for this API.

## Optional Attributes

4 The following attributes are optional for host environments that support this operation:

### **PMIX\_TIMEOUT "pmix.timeout" (int)**

5 Time in seconds before the specified operation should time out (0 indicating infinite) in  
6 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
7 the target process from ever exposing its data.

## Advice to PMIx library implementers

9 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
10 environment due to race condition considerations between completion of the operation versus  
11 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
12 directly in the PMIx server library must take care to resolve the race condition and should avoid  
13 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
14 created.

## Description

15 Non-blocking version of the **PMIx\_Group\_destruct** operation. The callback function will be  
16 called once all members of the group have executed either **PMIx\_Group\_destruct** or  
17 **PMIx\_Group\_destruct\_nb**.

## 13.2.6 **PMIx\_Group\_invite**

### Summary

20 Asynchronously construct a PMIx process group  
21

1      **Format**

2      *pmix\_status\_t*  
3      **PMIx\_Group\_invite**(const char grp[],  
4                            const pmix\_proc\_t procs[], size\_t nprocs,  
5                            const pmix\_info\_t directives[], size\_t ndirs,  
6                            pmix\_info\_t \*\*results, size\_t \*nresult)

C

7      **IN    grp**

8       NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
9       group identifier (string)

10     **IN    procs**

11     Array of **pmix\_proc\_t** structures containing the PMIx identifiers of the processes to be  
12     invited (array of handles)

13     **IN    nprocs**

14     Number of elements in the *procs* array (**size\_t**)

15     **IN    directives**

16     Array of **pmix\_info\_t** structures (array of handles)

17     **IN    ndirs**

18     Number of elements in the *directives* array (**size\_t**)

19     **INOUT results**

20     Pointer to a location where the array of **pmix\_info\_t** describing the results of the  
21     operation is to be returned (pointer to handle)

22     **INOUT nresults**

23     Pointer to a **size\_t** location where the number of elements in *results* is to be returned  
24     (memory reference)

25     Returns one of the following:

- 26     • **PMIX\_SUCCESS** , indicating that the request has been successfully completed
- 27     • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this  
28       operation
- 29     • a PMIx error constant indicating either an error in the input or that the request failed to be  
30       completed

31     ----- Required Attributes -----

32     The following attributes are *required* to be supported by all PMIx libraries that support this  
operation:

33     **PMIX\_GROUP\_OPTIONAL "pmix.grp.opt" (bool)**

34       Participation is optional - do not return an error if any of the specified processes terminate  
35       without having joined. The default is false

1 Host environments that support this operation are *required* to provide the following attributes:

2   **PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID** "pmix.grp.actxid" (bool)

3   Requests that the RM assign a new context identifier to the newly created group. The  
4   identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range  
5   specified in the request. Thus, the value serves as a means of identifying the group within  
6   that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

7   **PMIX\_GROUP\_NOTIFY\_TERMINATION** "pmix.grp.notterm" (bool)

8   Notify remaining members when another member terminates without first leaving the group.  
9   The default is false

### Optional Attributes

10 The following attributes are optional for host environments that support this operation:

11   **PMIX\_TIMEOUT** "pmix.timeout" (int)

12   Time in seconds before the specified operation should time out (0 indicating infinite) in  
13   error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
14   the target process from ever exposing its data.

### Advice to PMIx library implementers

15 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
16 environment due to race condition considerations between completion of the operation versus  
17 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
18 directly in the PMIx server library must take care to resolve the race condition and should avoid  
19 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
20 created.

1            **Description**

2     Explicitly invite the specified processes to join a group. The process making the  
3     **PMIx\_Group\_invite** call is automatically declared to be the *group leader*. Each invited  
4     process will be notified of the invitation via the **PMIX\_GROUP\_INVITED** event - the processes  
5     being invited must therefore register for the **PMIX\_GROUP\_INVITED** event in order to be notified  
6     of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of  
7     invite versus event registration is required.

8     The invitation event will include the identity of the inviting process plus the name of the group.  
9     When ready to respond, each invited process provides a response using either the blocking or  
10    non-blocking form of **PMIx\_Group\_join**. This will notify the inviting process that the  
11    invitation was either accepted (via the **PMIX\_GROUP\_INVITE\_ACCEPTED** event) or declined  
12    (via the **PMIX\_GROUP\_INVITE\_DECLINED** event). The **PMIX\_GROUP\_INVITE\_ACCEPTED**  
13    event is captured by the PMIx client library of the inviting process – i.e., the application itself does  
14    not need to register for this event. The library will track the number of accepting processes and  
15    alert the inviting process (by returning from the blocking form of **PMIx\_Group\_invite** or  
16    calling the callback function of the non-blocking form) when group construction completes.

17    The inviting process should, however, register for the **PMIX\_GROUP\_INVITE\_DECLINED** if the  
18    application allows invited processes to decline the invitation. This provides an opportunity for the  
19    application to either invite a replacement, declare “abort”, or choose to remove the declining  
20    process from the final group. The inviting process should also register to receive  
21    **PMIX\_GROUP\_INVITE\_FAILED** events whenever a process fails or terminates prior to  
22    responding to the invitation. Actions taken by the inviting process in response to these events must  
23    be communicated at the end of the event handler by returning the corresponding result so that the  
24    PMIx library can adjust accordingly.

25    Upon completion of the operation, all members of the new group will receive access to the job-level  
26    information of each other’s namespaces plus any information posted via **PMIx\_Put** by the other  
27    members.

28    The inviting process is automatically considered the leader of the asynchronous group construction  
29    procedure and will receive all failure or termination events for invited members prior to completion.  
30    The inviting process is required to provide a **PMIX\_GROUP\_CONSTRUCT\_COMPLETE** event  
31    once the group has been fully assembled – this event is used by the PMIx library as a trigger to  
32    release participants from their call to **PMIx\_Group\_join** and provides information (e.g., the  
33    final group membership) to be returned in the *results* array.

---

**Advice to users**

---

34    Applications are not allowed to use the group in any operations until group construction is  
35    complete. This is required in order to ensure consistent knowledge of group membership across all  
36    participants.

Failure of the inviting process at any time will cause a **PMIX\_GROUP\_LEADER\_FAILED** event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the **PMIX\_GROUP\_LEADER** attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a **PMIX\_GROUP\_LEADER\_SELECTED** event identifying the new leader. If no leader was selected, then the status code provided in the event handler will provide an error value so the participants can take appropriate action.

## 13.2.7 PMIx\_Group\_invite\_nb

### Summary

Non-blocking form of **PMIx\_Group\_invite**

### Format

PMIx v4.0

```
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.
- **PMIX\_ERR\_PARTIAL\_SUCCESS** The operation succeeded but not all specified members participated - the final group membership is included in the callback function
- **PMIX\_ERR\_NOT\_SUPPORTED** While the PMIx server supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request's failure

## Required Attributes

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

**PMIX\_GROUP\_OPTIONAL "pmix.grp.opt" (bool)**

Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false

Host environments that support this operation are *required* to provide the following attributes:

**PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID "pmix.grp.actxid" (bool)**

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size\_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX\_RANGE\_SESSION**.

**PMIX\_GROUP\_NOTIFY\_TERMINATION "pmix.grp.notterm" (bool)**

Notify remaining members when another member terminates without first leaving the group. The default is false

## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2 **PMIX\_TIMEOUT** "pmix.timeout" (int)

3 Time in seconds before the specified operation should time out (0 indicating infinite) in  
4 error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5 the target process from ever exposing its data.

## Advice to PMIx library implementers

6 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
7 environment due to race condition considerations between completion of the operation versus  
8 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
9 directly in the PMIx server library must take care to resolve the race condition and should avoid  
10 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
11 created.

### Description

13 Non-blocking version of the **PMIx\_Group\_invite** operation. The callback function will be  
14 called once all invited members of the group (or their substitutes) have executed either  
15 **PMIx\_Group\_join** or **PMIx\_Group\_join\_nb**.

## 16 13.2.8 **PMIx\_Group\_join**

### 17 Summary

18 Accept an invitation to join a PMIx process group

1           **Format**

C

```
2       pmix_status_t  
3       PMIx_Group_join(const char grp[],  
4                     const pmix_proc_t *leader,  
5                     pmix_group_operation_t opt,  
6                     const pmix_info_t directives[], size_t ndirs,  
7                     pmix_info_t **results, size_t *nresult)
```

C

8       **IN** **grp**

9       NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
10      group identifier (string)

11      **IN** **leader**

12      Process that generated the invitation (handle)

13      **IN** **opt**

14      Accept or decline flag (**pmix\_group\_operation\_t**)

15      **IN** **directives**

16      Array of **pmix\_info\_t** structures (array of handles)

17      **IN** **ndirs**

18      Number of elements in the *directives* array (**size\_t**)

19      **INOUT** **results**

20      Pointer to a location where the array of **pmix\_info\_t** describing the results of the  
21      operation is to be returned (pointer to handle)

22      **INOUT** **nresults**

23      Pointer to a **size\_t** location where the number of elements in *results* is to be returned  
24      (memory reference)

25      Returns one of the following:

- 26      • **PMIX\_SUCCESS**, indicating that the request has been successfully completed
- 27      • **PMIX\_ERR\_NOT\_SUPPORTED** The PMIx library and/or the host RM does not support this  
28        operation
- 29      • a PMIx error constant indicating either an error in the input or that the request failed to be  
30        completed

▼----- Required Attributes -----▼

31      There are no identified required attributes for implementers.



## Optional Attributes

1 The following attributes are optional for host environments that support this operation:

2   **PMIX\_TIMEOUT** "pmix.timeout" (int)

3   Time in seconds before the specified operation should time out (0 indicating infinite) in  
4   error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
5   the target process from ever exposing its data.

### Advice to PMIx library implementers

6 We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host  
7 environment due to race condition considerations between completion of the operation versus  
8 internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT**  
9 directly in the PMIx server library must take care to resolve the race condition and should avoid  
10 passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not  
11 created.

### Description

13 Respond to an invitation to join a group that is being asynchronously constructed. The process must  
14 have registered for the **PMIX\_GROUP\_INVITED** event in order to be notified of the invitation.  
15 When called, the event information will include the **pmix\_proc\_t** identifier of the process that  
16 generated the invitation along with the identifier of the group being constructed. When ready to  
17 respond, the process provides a response using either form of **PMIx\_Group\_join**.

### Advice to users

18 Since the process is alerted to the invitation in a PMIx event handler, the process *must not* use the  
19 blocking form of this call unless it first “thread shifts” out of the handler and into its own thread  
20 context. Likewise, while it is safe to call the non-blocking form of the API from the event handler,  
21 the process *must not* block in the handler while waiting for the callback function to be called.

1 Calling this function causes the inviting process (aka the *group leader*) to be notified that the  
2 process has either accepted or declined the request. The blocking form of the API will return once  
3 the group has been completely constructed or the group's construction has failed (as described  
4 below) – likewise, the callback function of the non-blocking form will be executed upon the same  
5 conditions.

6 Failure of the leader during the call to **PMIx\_Group\_join** will cause a  
7 **PMIX\_GROUP\_LEADER\_FAILED** event to be delivered to all invited participants so they can  
8 optionally declare a new leader. A new leader is identified by providing the  
9 **PMIX\_GROUP\_LEADER** attribute in the results array in the return of the event handler. Only one  
10 process is allowed to return that attribute, declaring itself as the new leader. Results of the leader  
11 selection will be communicated to all participants via a **PMIX\_GROUP\_LEADER\_SELECTED**  
12 event identifying the new leader. If no leader was selected, then the status code provided in the  
13 event handler will provide an error value so the participants can take appropriate action.

14 Any participant that returns **PMIX\_GROUP\_CONSTRUCT\_ABORT** from the leader failed event  
15 handler will cause all participants to receive an event notifying them of that status. Similarly, the  
16 leader may elect to abort the procedure by either returning **PMIX\_GROUP\_CONSTRUCT\_ABORT**  
17 from the handler assigned to the **PMIX\_GROUP\_INVITE\_ACCEPTED** or  
18 **PMIX\_GROUP\_INVITE\_DECLINED** codes, or by generating an event for the abort code. Abort  
19 events will be sent to all invited participants.

## 20 13.2.9 **PMIx\_Group\_join\_nb**

### 21 **Summary**

22 Non-blocking form of **PMIx\_Group\_join**

### 23 **Format**

24 *PMIx v4.0*

C

```
25 pmix_status_t
26 PMIx_Group_join_nb(const char grp[],           C
27                      const pmix_proc_t *leader,
28                      pmix_group_operation_t opt,
29                      const pmix_info_t directives[], size_t ndirs,
30                      pmix_info_cbfunc_t cbfunc, void *cbdata)
```

31 **IN grp**

32 NULL-terminated character array of maximum size **PMIX\_MAX\_NSLEN** containing the  
33 group identifier (string)

34 **IN leader**

Process that generated the invitation (handle)

```

1   IN  opt
2     Accept or decline flag ( pmix_group_operation_t )
3   IN  directives
4     Array of pmix_info_t structures (array of handles)
5   IN  ndirs
6     Number of elements in the directives array (size_t)
7   IN  cbfunc
8     Callback function pmix_info_cbfunc_t (function reference)
9   IN  cbdata
10    Data to be passed to the callback function (memory reference)

11 Returns one of the following:
12
13 • PMIX_SUCCESS , indicating that the request is being processed - result will be returned in the
14   provided cbfunc. Note that the library must not invoke the callback function prior to returning
15   from the API.
16
17 • PMIX_OPERATION_SUCCEEDED , indicating that the request was immediately processed and
18   returned success - the cbfunc will not be called
19
20 • PMIX_ERR_NOT_SUPPORTED The PMIx library does not support this operation - the cbfunc
21   will not be called
22
23 • a PMIx error constant indicating either an error in the input or that the request was immediately
24   processed and failed - the cbfunc will not be called

25 If executed, the status returned in the provided callback function will be one of the following
26 constants:
27
28 • PMIX_SUCCESS The operation succeeded and group membership is in the callback function
29   parameters
30
31 • PMIX_ERR_NOT_SUPPORTED While the PMIx server supports this operation, the host RM
32   does not.
33
34 • a non-zero PMIx error constant indicating a reason for the request's failure

```

### Required Attributes

There are no identified required attributes for implementers.

### Optional Attributes

The following attributes are optional for host environments that support this operation:

`PMIX_TIMEOUT "pmix.timeout" (int)`

Time in seconds before the specified operation should time out (0 indicating infinite) in  
error. The timeout parameter can help avoid “hangs” due to programming errors that prevent  
the target process from ever exposing its data.

## Advice to PMIx library implementers

We recommend that implementation of the **PMIX\_TIMEOUT** attribute be left to the host environment due to race condition considerations between completion of the operation versus internal timeout in the PMIx server library. Implementers that choose to support **PMIX\_TIMEOUT** directly in the PMIx server library must take care to resolve the race condition and should avoid passing **PMIX\_TIMEOUT** to the host environment so that multiple competing timeouts are not created.

### Description

Non-blocking version of the **PMIx\_Group\_join** operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either **PMIx\_Group\_join** or **PMIx\_Group\_join\_nb**.

## 13.2.10 PMIx\_Group\_leave

### Summary

Leave a PMIx process group

### Format

PMIx v4.0

```
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.

1           **Description**

2     Leave a PMIx Group. Calls to [PMIx\\_Group\\_leave](#) (or its non-blocking form) will cause a  
3     **PMIX\_GROUP\_LEFT** event to be generated notifying all members of the group of the caller's  
4     departure. The function will return (or the non-blocking function will execute the specified callback  
5     function) once the event has been locally generated and is not indicative of remote receipt.

6           **Advice to users**

7     The PMIx\_Group\_leave API is intended solely for asynchronous departures of individual processes  
8     from a group as it is not a scalable operation – i.e., when a process determines it should no longer  
9     be a part of a defined group, but the remainder of the group retains a valid reason to continue in  
10    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.

11    **13.2.11 PMIx\_Group\_leave\_nb**

12    **Summary**

13    Non-blocking form of [PMIx\\_Group\\_leave](#)

14    **Format**

PMIx v4.0

15           pmix\_status\_t  
16           **PMIx\_Group\_leave\_nb**(const char grp[],  
17                            const pmix\_info\_t directives[], size\_t ndirs,  
18                            pmix\_op\_cbfunc\_t cbfunc, void \*cbdata)

19    **IN grp**

20        NULL-terminated character array of maximum size [PMIX\\_MAX\\_NSLEN](#) containing the  
21        group identifier (string)

22    **IN directives**

23        Array of [pmix\\_info\\_t](#) structures (array of handles)

24    **IN ndirs**

25        Number of elements in the *directives* array ([size\\_t](#))

26    **IN cbfunc**

27        Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

28    **IN cbdata**

29        Data to be passed to the callback function (memory reference)

30        Returns one of the following:

- 31        • [PMIX\\_SUCCESS](#), indicating that the request is being processed - result will be returned in the  
32        provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning  
33        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

# Tools and Debuggers

---

The term *tool* widely refers to non-computational programs executed by the user or system administrator on a command line. Tools almost always 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 and monitoring 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 14.5 later in the chapter.

## 14.1 Connection Mechanisms

The key to supporting tools lies in providing mechanisms by which a tool can connect to a PMIx server. Application processes are able to connect because their local RM daemon provides them with the necessary contact information upon execution. A command-line tool, however, isn't spawned by an RM daemon, and therefore lacks the information required for rendezvous with a PMIx server.

Once a tool has started, it initializes PMIx as a tool (via `PMIx_tool_init`) if its access is restricted to PMIx-based informational services such as `PMIx_Query_info`. However, if the

1 tool intends to start jobs, then it must include the **PMIX\_LAUNCHER** attribute to inform the library  
2 of that intent so that the library can initialize and provide access to the corresponding support.

3 Support for tools requires that the PMIx server be initialized with an appropriate attribute  
4 indicating that tool connections are to be allowed. Separate attributes are provided to "fine-tune"  
5 this permission by allowing the environment to independently enable (or disable) connections from  
6 tools executing on nodes other than the one hosting the server itself. The PMIx server library shall  
7 provide an opportunity for the host environment to authenticate and approve each connection  
8 request from a specific tool by calling the **pmix\_server\_tool\_connection\_fn\_t** "hook"  
9 provided in the server module for that purpose. Servers in environments that do not provide this  
10 "hook" shall automatically reject all tool connection requests.

11 Tools can connect to any local or remote PMIx server provided they are either explicitly given the required connection information, or are able to discover it via one of several defined rendezvous  
12 protocols. Connection discovery centers around the existence of *rendezvous files* containing the  
13 necessary connection information, as illustrated in Fig. 14.1.  
14

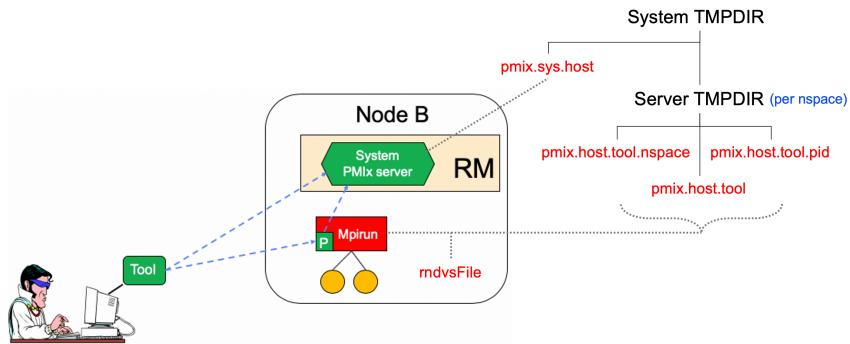


Figure 14.1.: Tool rendezvous files

15 The contents of each rendezvous file are specific to a given PMIx implementation, but should at  
16 least contain the namespace and rank of the server along with its connection URI. Note that tools  
17 linked to one PMIx implementation are therefore unlikely to successfully connect to PMIx server  
18 libraries from another implementation.

19 The top of the directory tree is defined by either the **PMIX\_SYSTEM\_TMPDIR** attribute (if given)  
20 or the **TMPDIR** environmental variable. PMIx servers that are designated as *system servers* by  
21 including the **PMIX\_SERVER\_SYSTEM\_SUPPORT** attribute when calling  
22 **PMIx\_server\_init** will create a rendezvous file in this top-level directory. The filename will  
23 be of the form *pmix.sys.hostname*, where *hostname* is the string returned by the **gethostname**  
24 system call. Note that only one PMIx server on a node can be designated as the system server.

25 Non-system PMIx servers will create a set of three rendezvous files in the directory defined by  
26 either the **PMIX\_SERVER\_TMPDIR** attribute or the **TMPDIR** environmental variable:

- *pmix.host.tool.nspace* where *host* is the string returned by the **gethostname** system call and *nspace* is the namespace of the server
- *pmix.host.tool.pid* where *host* is the string returned by the **gethostname** system call and *pid* is the PID of the server
- *pmix.host.tool* where *host* is the string returned by the **gethostname** system call. Note that servers which are not given a namespace-specific **PMIX\_SERVER\_TMPDIR** attribute may not generate this file due to conflicts should multiple servers be present on the node.

The files are identical and may be implemented as symlinks to a single instance. The individual file names are composed so as to aid the search process should a tool wish to connect to a server identified by its namespace or PID.

Servers will additionally provide a rendezvous file in any given location if the path (either absolute or relative) and filename is specified either during **PMIx\_server\_init** using the **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE** attribute, or by the **PMIX\_LAUNCHER\_RENDEZVOUS\_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 14.2.2.

Rendezvous file ownerships are set to the User ID (UID) and Group ID (GID) of the server that created them, with permissions set according to the desires of the implementation and/or system administrator policy. All connection attempts are first governed by read access privileges to the target rendezvous file - thus, the combination of permissions, UID, and GID of the rendezvous files act as a first-level of security for tool access.

A tool may connect to as many servers at one time as the implementation supports, but is limited to designating only one such connection as its *primary* server. This is done to avoid confusion when the tool calls an API as to which server should service the request. The first server the tool connects to is automatically designated as the *primary* server.

Tools are allowed to change their primary server at any time via the **PMIx\_tool\_set\_server** API, and to connect/disconnect from a server as many times as desired. Note that standing requests (e.g., event registrations) with the current primary server may be lost and/or may not be transferred when transitioning to another primary server - PMIx implementors are not required to maintain or transfer state across tool-server connections.

Tool process identifiers are assigned by one of the following methods:

- If **PMIX\_TOOL\_NSPACE** is given, then the namespace of the tool will be assigned that value
- If **PMIX\_TOOL\_RANK** is given, then the rank of the tool will be assigned that value. Passing **PMIX\_TOOL\_NSPACE** without also specifying the rank will result in the rank being set to a default value of zero
- If a process ID has not been provided, then one will be assigned by the host environment upon connection to a server. Users should note that the tool's process ID will be *invalid* until a

connection has been established

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\_connect\_to\_server**.

### 14.1.1 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\_connect\_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.

- 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.
- 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 Transmission Control Protocol (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..
- If **PMIX\_SERVER\_NSPACE** is given, then the tool will search for a rendezvous file created by a PMIx server of the given namespace in the **PMIX\_SERVER\_TMPDIR** directory.
- 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.

- 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.

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.

### 14.1.2 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.
- 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.

### 14.1.3 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.

1 In cases where the user has only the one job of interest in operation on the local node (e.g., when  
2 engaged in an interactive session on the node from which the launcher was executed), the normal  
3 rendezvous file discovery method can often be used to successfully connect to the target job, even  
4 in the presence of jobs executed by other users. The permissions and security authorizations can, in  
5 many cases, reliably ensure that only the one connection can be made. However, this is not  
6 guaranteed in all cases.

7 The most common method, therefore, for attaching to a running job is to specify either the PID of  
8 the job's launcher or the namespace of the launcher's job (note that the launcher's namespace  
9 frequently differs from the namespace of the job it has launched). Unless the application processes  
10 themselves act as PMIx servers, connection must be to the servers in the daemons that oversee the  
11 application. This is typically either daemons specifically started by the job's launcher process, or  
12 daemons belonging to the host environment, that are responsible for starting the application's  
13 processes and oversee their execution.

14 Identifying the correct PID or namespace can be accomplished in a variety of ways, including:

- 15 • Using typical OS or host environment tools to obtain a listing of active jobs and perusing those to  
16 find the target launcher
- 17 • Using a PMIx-based tool attached to a system-level server to query the active jobs and their  
18 command lines, thereby identifying the application of interest and its associated launcher
- 19 • Manually recording the PID of the launcher upon starting the job

20 Once the namespace and/or PID of the target server has been identified, either of the previous  
21 methods can be used to connect to it.

#### 22 14.1.4 Tool initialization attributes

23 The following attributes are passed to the `PMIx_tool_init` API for use when initializing the  
24 PMIx library.

```
25 PMIX_TOOL_NSPACE "pmix.tool.nspace" (char*)
26           Name of the namespace to use for this tool.
27 PMIX_TOOL_RANK "pmix.tool.rank" (uint32_t)
28           Rank of this tool.
29 PMIX_LAUNCHER "pmix.tool.launcher" (bool)
30           Tool is a launcher and needs to create rendezvous files
```

#### 31 14.1.5 Tool initialization environmental variables

32 The following environmental variables are used during `PMIx_tool_init` to control various  
33 rendezvous-related operations when the tool is started manually (e.g., on a command line) or by a  
34 fork/exec-like operation.

35 `PMIX_LAUNCHER_PAUSE_FOR_TOOL`

1       Upon completing **PMIx\_server\_init**, the spawned launcher is to generate the  
2       **PMIX\_LAUNCHER\_READY** event and then pause until the spawning tool can connect to it  
3       and provide directives.  
4       **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE**  
5       Pathname of file where the launcher is to store its connection information so that the  
6       spawning tool can connect to it.

## 7     14.1.6 Tool connection attributes

8       These attributes are defined to assist PMIx-enabled tools to connect with a PMIx server by passing  
9       them into either the **PMIx\_tool\_init** or the **PMIx\_tool\_connect\_to\_server** APIs -  
10      thus, they are not typically accessed via the **PMIx\_Get** API.

11      **PMIX\_SERVER\_PIDINFO "pmix.srvr.pidinfo" (pid\_t)**  
12        PID of the target PMIx server for a tool.  
13      **PMIX\_CONNECT\_TO\_SYSTEM "pmix.cnct.sys" (bool)**  
14        The requester requires that a connection be made only to a local, system-level PMIx server.  
15      **PMIX\_CONNECT\_SYSTEM\_FIRST "pmix.cnct.sys.first" (bool)**  
16        Preferentially, look for a system-level PMIx server first.  
17      **PMIX\_SERVER\_URI "pmix.srvr.uri" (char\*)**  
18        URI of the PMIx server to be contacted.  
19      **PMIX\_SERVER\_HOSTNAME "pmix.srvr.host" (char\*)**  
20        Host where target PMIx server is located.  
21      **PMIX\_CONNECT\_MAX\_RETRIES "pmix.tool.mretries" (uint32\_t)**  
22        Maximum number of times to try to connect to PMIx server.  
23      **PMIX\_CONNECT\_RETRY\_DELAY "pmix.tool.retry" (uint32\_t)**  
24        Time in seconds between connection attempts to a PMIx server.  
25      **PMIX\_TOOL\_DO\_NOT\_CONNECT "pmix.tool.nocon" (bool)**  
26        The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.  
27      **PMIX\_TOOL\_CONNECT\_OPTIONAL "pmix.tool.conopt" (bool)**  
28        The tool shall connect to a server if available, but otherwise continue to operate unconnected  
29      **PMIX\_RECONNECT\_SERVER "pmix.tool.recon" (bool)**  
30        Tool is requesting to change server connections - superseded by the  
31        **PMIx\_tool\_connect\_to\_server** API  
32      **PMIX\_TOOL\_ATTACHMENT\_FILE "pmix.tool.attach" (char\*)**  
33        Pathname of file containing connection information to be used for attaching to a specific  
34        server  
35      **PMIX\_LAUNCHER\_RENDEZVOUS\_FILE "pmix.tool.lncrnd" (char\*)**  
36        Pathname of file where the launcher is to store its connection information so that the  
37        spawning tool can connect to it.  
38      **PMIX\_PRIMARY\_SERVER "pmix.pri.srvr" (bool)**  
39        The server to which the tool is connecting shall be designated the *primary* server once  
40        connection has been accomplished.

## 14.2 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.

### 14.2.1 Direct launch

In the direct-launch use-case (Fig. 14.2), 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 14.4):

- `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.

- `PMIX_FWD_STDOUT "pmix.fwd.stdout"` (`bool`)

Requests that the `stdout` of the spawned processes be forwarded to the requester for handling

- `PMIX_FWD_STDERR "pmix.fwd.stderr"` (`bool`)

Requests that the `stderr` of the spawned processes be forwarded to the requester for handling

- `PMIX_FWD_STDDIAG "pmix.fwd.stddiag"` (`bool`)

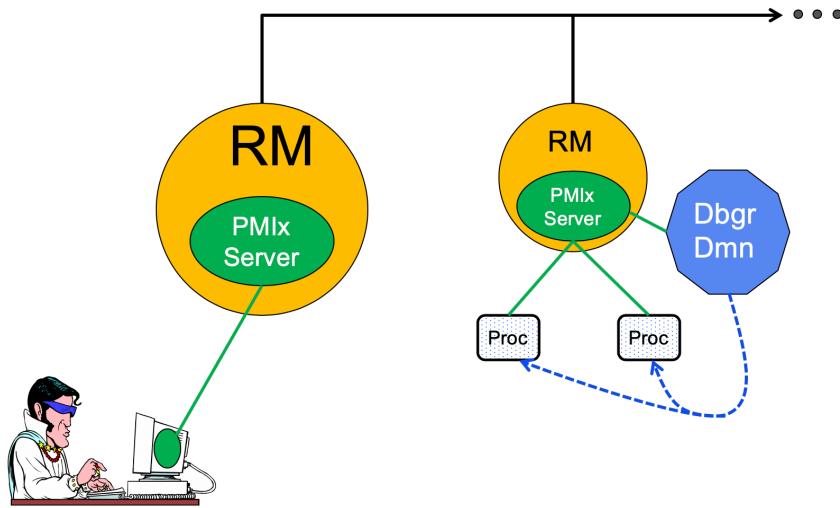


Figure 14.2.: Direct Launch

1 Requests that the diagnostic channel of the spawned processes be forwarded to the  
2 requester for handling, if that channel exists

- 3 • **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.cszie" (**uint32\_t**)  
4 The requested size of the PMIx server cache in bytes for each specified channel. By  
5 default, the server is allowed (but not required) to drop all bytes received beyond the max  
6 size.
- 7 • **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (**bool**)  
8 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
9 cache.
- 10 • **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (**bool**)  
11 In an overflow situation, the PMIx server is to drop any new bytes received until room  
12 becomes available in the cache (default).
- 13 • **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (**uint32\_t**)  
14 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until  
15 the specified number of bytes is collected to avoid being called every time a block of IO  
16 arrives. The PMIx tool library will execute the callback and reset the collection counter  
17 whenever the specified number of bytes becomes available. Any remaining buffered data  
18 will be *flushed* to the callback upon a call to deregister the respective channel.
- 19 • **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (**uint32\_t**)  
20 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
21 size, this prevents IO from being held indefinitely while waiting for another payload to

- 1                   arrive.
- 2     • **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (bool)  
   3        Requests that output be prefixed with the nspace,rank of the source and a string  
   4        identifying the channel (**stdout**, **stderr**, etc.)
- 5     • **PMIX\_IOF\_TIMESTAMP\_OUTPUT** "pmix.iof.ts" (bool)  
   6        Requests that output be marked with the time at which the data was received by the tool -  
   7        note that this will differ from the time at which the data was collected from the source
- 8     • **PMIX\_IOF\_XML\_OUTPUT** "pmix.iof.xml" (bool)  
   9        Requests that output be formatted in XML
- 10    • **PMIX\_NOHUP** "pmix.nohup" (bool)  
 11       Any processes started on behalf of the calling tool (or the specified namespace, if such  
 12       specification is included in the list of attributes) should continue after the tool disconnects  
 13       from its server
- 14    • **PMIX\_NOTIFY\_JOB\_EVENTS** "pmix.note.jev" (bool)  
 15       Requests that the PMIx server provide the requester with the  
 16       **PMIX\_EVENT\_JOB\_START**, **PMIX\_LAUNCH\_COMPLETE**, and  
 17       **PMIX\_EVENT\_JOB\_END** events. Each event is to include at least the namespace of the  
 18       corresponding job and a **PMIX\_EVENT\_TIMESTAMP** indicating the time the event  
 19       occurred. Note that these events can be captured and processed by registering a default  
 20       event handler instead of individual handlers and then processing the events based on the  
 21       returned status code. Another common method is to register one event handler for all  
 22       job-related events, with a separate handler for non-job events - see  
 23       **PMIx\_Register\_event\_handler** for details.
- 24    • **PMIX\_NOTIFY\_COMPLETION** "pmix.notecomp" (bool)  
 25       Requests that the PMIx generate the **PMIX\_EVENT\_JOB\_END** event for normal or  
 26       abnormal termination of the spawned job. The event shall include the returned status code  
 27       for the corresponding job and a **PMIX\_EVENT\_TIMESTAMP** indicating the time the  
 28       termination occurred.
- 29    • **PMIX\_LOG\_JOB\_EVENTS** "pmix.log.jev" (bool)  
 30       Requests that the PMIx server log the **PMIX\_EVENT\_JOB\_START**,  
 31       **PMIX\_LAUNCH\_COMPLETE**, and **PMIX\_EVENT\_JOB\_END** events using **PMIx\_Log**  
 32       instead of generating PMIx events, subject to the logging attributes of Section 3.4.22
- 33    • **PMIX\_DEBUG\_STOP\_ON\_EXEC** "pmix.dbg.exec" (bool)  
 34       Included in either the *pmix\_info\_t* array in a **pmix\_app\_t** description (if the directive  
 35       applies only to that application) or in the *job\_info* array if it applies to all applications in  
 36       the given spawn request. Indicates that the application is being spawned under a debugger,  
 37       and that the local launch agent is to pause the resulting application processes on first  
 38       instruction for debugger attach. The launcher (RM or IL) is to generate the  
 39       **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 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 when they reach some application-determined location and pause at that point until 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 application. 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 to the requester. Once the spawn request has succeeded in starting the specified processes, the request will return **PMIX\_SUCCESS** back to the requester. Upon termination of the spawned application, the host environment must generate a termination event if requested to do so - the termination event includes the status returned by the spawned application plus any other info provided by the host environment.

## 14.2.2 Indirect launch

In the indirect launch use-case, the application processes are started via an intermediate launcher (e.g., *mpexec*) that is itself started by the tool (see Fig 14.3). Thus, at a high level, this is a two-stage launch procedure to start the application: the tool starts the IL, which then starts the

1 applications. In practice, additional steps may be involved if, for example, the IL starts its own  
2 daemons to shepherd the application processes.

3 A key aspect of this operational mode is the avoidance of any requirement that the tool parse and/or  
4 understand the command line of the IL. Instead, the indirect launch support relies on PMIx  
5 definitions to abstract the tool-launcher interaction, and on the PMIx event notification system  
6 (Chapter 8) to coordinate between the tool and IL.

### Advice to users

7 The precise steps involved in starting and connecting to the IL differ somewhat depending on  
8 whether or not an external PMIx server (e.g., a system-level server provided by the host  
9 environment) is available. Tool developers wishing to orchestrate indirect launch themselves are  
10 referred to Section ?? for a detailed description of the process.

11 The tool spawns the IL using the same procedure for launching an application - it assembles the  
12 description of the IL (e.g., by parsing its command line) into a spawn request containing an array of  
13 `pmix_app_t` and `pmix_info_t` job-level information. An allocation of resources for the IL  
14 itself may or may not be required – if it is, then the allocation must be made in advance or the  
15 spawn request must include allocation request information.

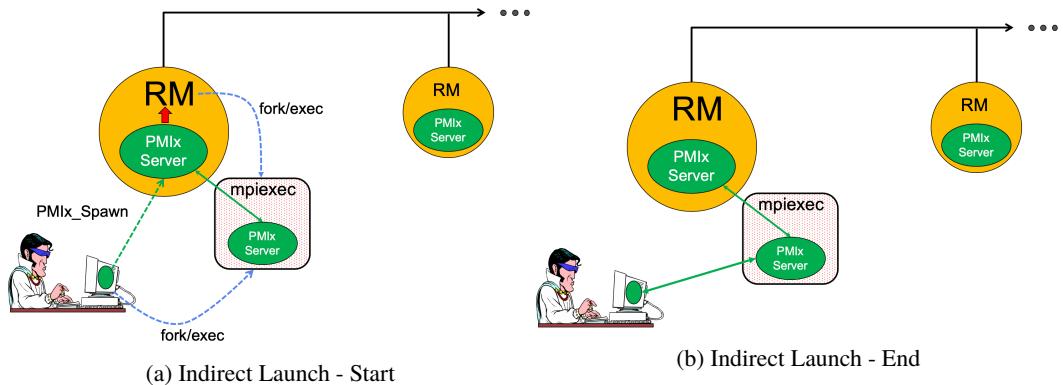


Figure 14.3.: Indirect launch procedure

16 The tool may optionally wish to include the following tool-specific attributes in the *job\_info*  
17 argument to `PMIx_Spawn` - note that these attributes refer to the behavior of the IL itself and not  
18 the eventual job to be launched:

- 19 • **PMIX\_FWD\_STDIN "pmix.fwd.stdin" (pmix\_rank\_t)**

20 The requester intends to push information from its `stdin` to the indicated process. The  
21 local spawn agent should, therefore, ensure that the `stdin` channel to that process  
22 remains available. A rank of `PMIX_RANK_WILDCARD` indicates that all processes in the  
23 spawned job are potential recipients.

- **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)  
Requests that the **stdout** of the spawned processes be forwarded to the requester for handling
- **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)  
Requests that the **stderr** of the spawned processes be forwarded to the requester for handling
- **PMIX\_FWD\_STDDIAG** "pmix.fwd.stddiag" (bool)  
Requests that the diagnostic channel of the spawned processes be forwarded to the requester for handling, if that channel exists
- **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csizes" (uint32\_t)  
The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.
- **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (bool)  
In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the cache.
- **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (bool)  
In an overflow situation, the PMIx server is to drop any new bytes received until room becomes available in the cache (default).
- **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (uint32\_t)  
Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the specified number of bytes is collected to avoid being called every time a block of IO arrives. The PMIx tool library will execute the callback and reset the collection counter whenever the specified number of bytes becomes available. Any remaining buffered data will be *flushed* to the callback upon a call to deregister the respective channel.
- **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (uint32\_t)  
Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
- **PMIX\_IOF\_TAG\_OUTPUT** "pmix.iof.tag" (bool)  
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)

- 1 Any processes started on behalf of the calling tool (or the specified namespace, if such  
 2 specification is included in the list of attributes) should continue after the tool disconnects  
 3 from its server
- 4 • **PMIX\_LAUNCHER\_DAEMON** "pmix.lnch.dmn" (**char\***)  
 5 Path to executable that is to be used as the backend daemon for the launcher. This replaces  
 6 the launcher's own daemon with the specified executable. Note that the user is therefore  
 7 responsible for ensuring compatibility of the specified executable and the host launcher.
  - 8 • **PMIX\_FORKEXEC\_AGENT** "pmix.frkex.agnt" (**char\***)  
 9 Path to executable that the launcher's backend daemons are to fork/exec in place of the  
 10 actual application processes. The fork/exec agent shall connect back (as a PMIx tool) to  
 11 the launcher's daemon to receive its spawn instructions, and is responsible for starting the  
 12 actual application process it replaced. See Section 14.4.3 for details.
  - 13 • **PMIX\_EXEC\_AGENT** "pmix.exec.agnt" (**char\***)  
 14 Path to executable that the launcher's backend daemons are to fork/exec in place of the  
 15 actual application processes. The launcher's daemon shall pass the full command line of  
 16 the application on the command line of the exec agent, which shall not connect back to the  
 17 launcher's daemon. The exec agent is responsible for exec'ing the specified application  
 18 process in its own place. See Section 14.4.3 for details
- 19 The tool then calls the **PMIx\_Spawn** API so that the PMIx library can either communicate the  
 20 spawn request to the server (if connected to one), or locally spawn the IL itself if not connected to a  
 21 server and the PMIx implementation includes self-spawn support. **PMIx\_Spawn** shall return an  
 22 error if neither of these conditions is met, or if the IL fails to successfully complete the necessary  
 23 rendezvous procedure (as detailed in Section ??).
- 24 Upon successful return from **PMIx\_Spawn** :
- 25 • The namespace of the IL shall be returned in the *nspace* parameter of the **PMIx\_Spawn** API (or  
 26 in the **pmix\_spawn\_cbfunc\_t** for the non-blocking form of that API)
  - 27 • The tool shall be connected to the IL with the IL acting as the tool's *primary* server
- 28 Once **PMIx\_Spawn** has returned, the tool can proceed to spawn the actual application according  
 29 to the procedure describe in Section 14.2.1.

### 30 14.2.3 Tool spawn-related attributes

31 Tools are free to utilize the spawn attributes available to applications (see 3.4.20) when  
 32 constructing a spawn request, but can also utilize the following attributes that are specific to  
 33 tool-based spawn operations:

#### 34 **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (**pmix\_rank\_t**)

35 The requester intends to push information from its **stdin** to the indicated process. The  
 36 local spawn agent should, therefore, ensure that the **stdin** channel to that process remains  
 37 available. A rank of **PMIX\_RANK\_WILDCARD** indicates that all processes in the spawned  
 38 job are potential recipients.

```

1   PMIX_FWD_STDOUT "pmix.fwd.stdout" (bool)
2       Requests that the stdout of the spawned processes be forwarded to the requester for
3           handling
4   PMIX_FWD_STDERR "pmix.fwd.stderr" (bool)
5       Requests that the stderr of the spawned processes be forwarded to the requester for
6           handling
7   PMIX_FWD_STDDIAG "pmix.fwd.stddiag" (bool)
8       Requests that the diagnostic channel of the spawned processes be forwarded to the requester
9           for handling, if that channel exists
10  PMIX_NOHUP "pmix.nohup" (bool)
11      Any processes started on behalf of the calling tool (or the specified namespace, if such
12          specification is included in the list of attributes) should continue after the tool disconnects
13          from its server
14  PMIX_LAUNCHER_DAEMON "pmix.lnch.dmn" (char*)
15      Path to executable that is to be used as the backend daemon for the launcher. This replaces
16          the launcher's own daemon with the specified executable. Note that the user is therefore
17          responsible for ensuring compatibility of the specified executable and the host launcher.
18  PMIX_FORKEXEC_AGENT "pmix.frkex.agnt" (char*)
19      Path to executable that the launcher's backend daemons are to fork/exec in place of the actual
20          application processes. The fork/exec agent shall connect back (as a PMIx tool) to the
21          launcher's daemon to receive its spawn instructions, and is responsible for starting the actual
22          application process it replaced. See Section 14.4.3 for details.
23  PMIX_EXEC_AGENT "pmix.exec.agnt" (char*)
24      Path to executable that the launcher's backend daemons are to fork/exec in place of the actual
25          application processes. The launcher's daemon shall pass the full command line of the
26          application on the command line of the exec agent, which shall not connect back to the
27          launcher's daemon. The exec agent is responsible for exec'ing the specified application
28          process in its own place. See Section 14.4.3 for details

```

## 29 14.2.4 Tool spawn-related constants

```

30  PMIX_LAUNCH_DIRECTIVE    Launcher directives have been received from a PMIx-enabled
31      tool
32  PMIX_LAUNCHER_READY     Application launcher (e.g., mpieexec) is ready to receive directives
33      from a PMIx-enabled tool

```

## 34 14.3 IO Forwarding

```

35 Underlying the operation of many tools is a common need to forward stdin from the tool to
36 targeted processes, and to return stdout/stderr from those processes to the tool (e.g., for
37 display on the user's console). Historically, each tool developer was responsible for creating their
38 own IO forwarding subsystem. However, the introduction of PMIx as a standard mechanism for

```

1 interacting between applications and the host environment has made it possible to relieve tool  
2 developers of this burden.

3 This section defines functions by which tools can request forwarding of input/output to/from other  
4 processes and serves as a design guide to:

- 5 • provide tool developers with an overview of the expected behavior of the PMIx IO forwarding  
6 support;  
7 • guide RM vendors regarding roles and responsibilities expected of the RM to support IO  
8 forwarding; and  
9 • provide insight into the thinking of the PMIx community behind the definition of the PMIx IO  
10 forwarding APIs

11 Note that the forwarding of IO via PMIx requires that both the host environment and the tool  
12 support PMIx, but does not impose any similar requirements on the application itself.

13 The responsibility of the host environment in forwarding of IO falls into the following areas:

- 14 • Capturing output from specified processes  
15 • Forwarding that output to the host of the PMIx server library that requested it  
16 • Delivering that payload to the PMIx server library via the [PMIX\\_SERVER\\_IOF\\_DELIVER](#)  
17 API for final dispatch to the requesting tool

18 It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the  
19 requesting client. This may require caching of output until a forwarding registration is received, as  
20 governed by the corresponding IO forwarding attributes of Section 14.3.3 that are supported by the  
21 implementation.

## 22 14.3.1 Forwarding stdout/stderr

23 At an appropriate point in its operation (usually during startup), a tool will utilize the  
24 [PMIX\\_TOOL\\_INIT](#) function to connect to a PMIx server. The PMIx server can be hosted by an  
25 RM daemon or could be embedded in a library-provided starter program such as *mpiexec* - in terms  
26 of IO forwarding, the operations remain the same either way. For purposes of this discussion, we  
27 will assume the server is in an RM daemon and that the application processes are directly launched  
28 by the RM, as shown in Fig 14.4.

29 Once the tool has connected to the target server, it can request that processes be spawned on its  
30 behalf or that output from a specified set of existing processes in a given executing application be  
31 forwarded to it. Requests to forward output from processes being spawned by the tool should be  
32 included in calls to the [PMIX\\_SPAWN](#) API using the [PMIX\\_FWD\\_STDOUT](#) and/or  
33 [PMIX\\_FWD\\_STDERR](#) attributes. Requests to capture output from existing processes can be made  
34 via the [PMIX\\_IOF\\_PULL](#) API.

35 Two modes are supported when forwarding standard output/error:

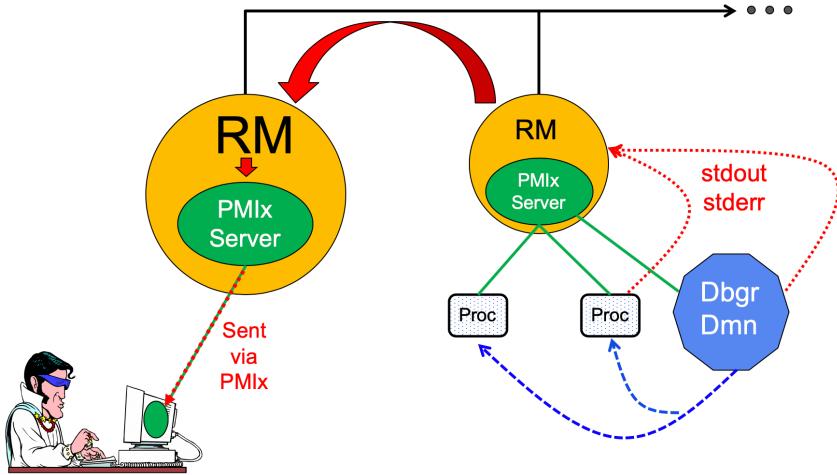


Figure 14.4.: Forwarding stdout/stderr

1     • **PMIX\_IOF\_COPY "pmix.iof.cpy" (bool)**

2         Requests that the host environment deliver a copy of the specified output stream(s) to the  
3         tool, letting the stream(s) continue to also be delivered to the default location. This allows  
4         the tool to tap into the output stream(s) without redirecting it from its current final  
5         destination

6     • **PMIX\_IOF\_REDIRECT "pmix.iof.redir" (bool)**

7         Requests that the host environment intercept the specified output stream(s) and deliver it  
8         to the requesting tool instead of its current final destination. This might be used, for  
9         example, during a debugging procedure to avoid injection of debugger-related output into  
10        the application's results file. The original output stream(s) destination is restored upon  
11        termination of the tool. This is the default mode of operation

12       When requesting to forward output, the tool can specify several formatting options to be used on  
13       the resulting output stream. These include:

14     • **PMIX\_IOF\_TAG\_OUTPUT "pmix.iof.tag" (bool)**

15         Requests that output be prefixed with the nspace, rank of the source and a string  
16         identifying the channel (**stdout**, **stderr**, etc.)

17     • **PMIX\_IOF\_TIMESTAMP\_OUTPUT "pmix.iof.ts" (bool)**

18         Requests that output be marked with the time at which the data was received by the tool -  
19         note that this will differ from the time at which the data was collected from the source

20     • **PMIX\_IOF\_XML\_OUTPUT "pmix.iof.xml" (bool)**

21         Requests that output be formatted in XML

1       The PMIx client in the tool is responsible for formatting the output stream. Note that output from  
2       multiple processes will often be interleaved due to variations in arrival time - ordering of output is  
3       not guaranteed across processes and/or nodes.

#### 4     14.3.2 Forwarding `stdin`

5       A tool is not necessarily a child of the RM as it may have been started directly from the command  
6       line. Thus, provision must be made for the tool to collect its `stdin` and pass it to the host RM (via  
7       the PMIx server) for forwarding. Two methods of support for forwarding of `stdin` are defined:

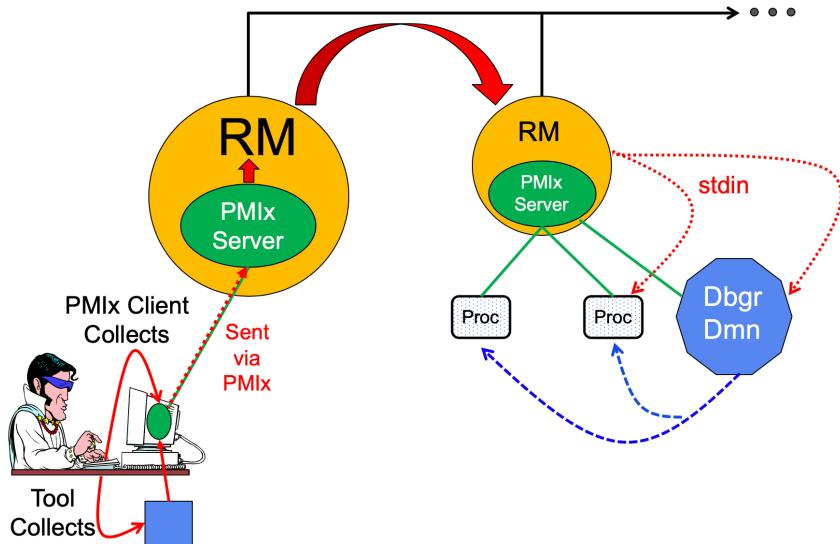


Figure 14.5.: Forwarding `stdin`

- 8       internal collection by the PMIx tool library itself. This is requested via the  
9       `PMIX_IOF_PUSH_STDIN` attribute in the `PMIx_IOF_push` call. When this mode is  
10      selected, the tool library begins collecting all `stdin` data and internally passing it to the local  
11      server for distribution to the specified target processes. All collected data is sent to the same  
12      targets until `stdin` is closed, or a subsequent call to `PMIx_IOF_push` is made that includes  
13      the `PMIX_IOF_COMPLETE` attribute indicating that forwarding of `stdin` is to be terminated.
- 14      external collection directly by the tool. It is assumed that the tool will provide its own  
15      code/mechanism for collecting its `stdin` as the tool developers may choose to insert some  
16      filtering and/or editing of the stream prior to forwarding it. In addition, the tool can directly  
17      control the targets for the data on a per-call basis – i.e., each call to `PMIx_IOF_push` can  
18      specify its own set of target recipients for that particular *blob* of data. Thus, this method provides  
19      maximum flexibility, but requires that the tool developer provide their own code to capture  
20      `stdin`.

1 Note that it is the responsibility of the RM to forward data to the host where the target process(es)  
2 are executing, and for the host daemon on that node to deliver the data to the **stdin** of target  
3 process(es).. The PMIx server on the remote node is not involved in this process. Systems that do  
4 not support forwarding of **stdin** shall return **PMIX\_ERR\_NOT\_SUPPORTED** in response to a  
5 forwarding request.

### Advice to users

6 Scalable forwarding of **stdin** represents a significant challenge. Most environments will at least  
7 handle a *send-to-1* model whereby **stdin** is forwarded to a single identified process, and  
8 occasionally an additional *send-to-all* model where **stdin** is forwarded to all processes in the  
9 application. Users are advised to check their host environment for available support as the  
10 distribution method lies outside the scope of PMIx.

11 **Stdin** buffering by the RM and/or PMIx library can be problematic. If any targeted recipient is  
12 slow reading data (or decides never to read data), then the data must be buffered in some  
13 intermediate daemon or the PMIx tool library itself. Thus, piping a large amount of data into  
14 **stdin** can result in a very large memory footprint in the system management stack or the tool.  
15 Best practices, therefore, typically focus on reading of input files by application processes as  
16 opposed to forwarding of **stdin**.

### 14.3.3 IO Forwarding attributes

18 The following attributes are used to control IO forwarding behavior at the request of tools.

19 **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.cszie" (**uint32\_t**)

20 The requested size of the PMIx server cache in bytes for each specified channel. By default,  
21 the server is allowed (but not required) to drop all bytes received beyond the max size.

22 **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (**bool**)

23 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
24 cache.

25 **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (**bool**)

26 In an overflow situation, the PMIx server is to drop any new bytes received until room  
27 becomes available in the cache (default).

28 **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (**uint32\_t**)

29 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
30 specified number of bytes is collected to avoid being called every time a block of IO arrives.  
31 The PMIx tool library will execute the callback and reset the collection counter whenever the  
32 specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
33 the callback upon a call to deregister the respective channel.

34 **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (**uint32\_t**)

35 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
36 size, this prevents IO from being held indefinitely while waiting for another payload to arrive.

37 **PMIX\_IOF\_COMPLETE** "pmix.iof.cmp" (**bool**)

```

1     Indicates that the specified IO channel has been closed by the source.
2 PMIX_IOF_TAG_OUTPUT "pmix.iof.tag" (bool)
3     Requests that output be prefixed with the nspace,rank of the source and a string identifying
4     the channel (stdout, stderr, etc.)
5 PMIX_IOF_TIMESTAMP_OUTPUT "pmix.iof.ts" (bool)
6     Requests that output be marked with the time at which the data was received by the tool -
7     note that this will differ from the time at which the data was collected from the source
8 PMIX_IOF_XML_OUTPUT "pmix.iof.xml" (bool)
9     Requests that output be formatted in XML
10 PMIX_IOF_PUSH_STDIN "pmix.iof.stdin" (bool)
11    Requests that the PMIx library collect the stdin of the requester and forward it to the
12    processes specified in the PMIx_IOF_push call. All collected data is sent to the same
13    targets until stdin is closed, or a subsequent call to PMIx_IOF_push is made that
14    includes the PMIX_IOF_COMPLETE attribute indicating that forwarding of stdin is to
15    be terminated.
16 PMIX_IOF_COPY "pmix.iof.cpy" (bool)
17    Requests that the host environment deliver a copy of the specified output stream(s) to the
18    tool, letting the stream(s) continue to also be delivered to the default location. This allows the
19    tool to tap into the output stream(s) without redirecting it from its current final destination
20 PMIX_IOF_REDIRECT "pmix.iof.redir" (bool)
21    Requests that the host environment intercept the specified output stream(s) and deliver it to
22    the requesting tool instead of its current final destination. This might be used, for example,
23    during a debugging procedure to avoid injection of debugger-related output into the
24    application's results file. The original output stream(s) destination is restored upon
25    termination of the tool

```

## 14.4 Debugger Support

27 Debuggers are a class of tool that merits special consideration due to their particular requirements  
28 for access to job-related information and control over process execution. The primary advantage of  
29 using PMIx for these purposes lies in the resulting portability of the debugger as it can be used with  
30 any system and/or programming model that supports PMIx. In addition to the general tool support  
31 described above, debugger support includes:

- 32 • Co-location, co-spawn, and communication wireup of debugger daemons for scalable launch.  
33 This includes providing debugger daemons with endpoint connection information across the  
34 daemons themselves.
- 35 • Identification of the job that is to be debugged. This includes automatically providing debugger  
36 daemons with the job-level information for their target job

37 Debuggers can also utilize the options in the **PMIx\_Spawn** API to exercise a degree of control  
38 over spawned jobs for debugging purposes. For example, a debugger can utilize the environmental  
39 parameter attributes of Section 3.4.24 to request **LD\_PRELOAD** of a memory interceptor library

1 prior to spawning an application process, or interject a custom fork/exec agent to shepherd the  
2 application process.

3 A key element of the debugging process is the ability of the debugger to require that processes  
4 *pause* at some well-defined point, thereby providing the debugger with an opportunity to attach and  
5 control execution. The actual implementation of the *pause* lies outside the scope of PMIx - it  
6 typically requires either the launcher or the application itself to implement the necessary  
7 operations. However, PMIx does provide several standard attributes by which the debugger can  
8 specify the desired attach point:

- 9     • **PMIX\_DEBUG\_STOP\_ON\_EXEC** "pmix.dbg.exec" (bool)  
10         Included in either the *pmix\_info\_t* array in a **pmix\_app\_t** description (if the directive  
11             applies only to that application) or in the *job\_info* array if it applies to all applications in  
12             the given spawn request. Indicates that the application is being spawned under a debugger,  
13             and that the local launch agent is to pause the resulting application processes on first  
14             instruction for debugger attach. The launcher (RM or IL) is to generate the  
15             **PMIX\_LAUNCH\_COMPLETE** event when all processes are stopped at the exec point.  
16             Launchers that cannot support this operation shall return an error from the **PMIx\_Spawn**  
17             API if this behavior is requested.
- 18     • **PMIX\_DEBUG\_STOP\_IN\_INIT** "pmix.dbg.init" (bool)  
19         Included in either the *pmix\_info\_t* array in a **pmix\_app\_t** description (if the directive  
20             applies only to that application) or in the *job\_info* array if it applies to all applications in  
21             the given spawn request. Indicates that the specified application is being spawned under a  
22             debugger. The PMIx client library in each resulting application process shall notify its  
23             PMIx server that it is pausing and then pause during **PMIx\_Init** of the spawned  
24             processes until receipt of the **PMIX\_DEBUGGER\_RELEASE** event. The launcher (RM or  
25             IL) is responsible for generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event  
26             when all processes have reached the pause point. PMIx implementations that do not  
27             support this operation shall return an error from **PMIx\_Init** if this behavior is  
28             requested. Launchers that cannot support this operation shall return an error from the  
29             **PMIx\_Spawn** API if this behavior is requested.
- 30     • **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** "pmix.dbg.notify" (bool)  
31         Included in either the *pmix\_info\_t* array in a **pmix\_app\_t** description (if the directive  
32             applies only to that application) or in the *job\_info* array if it applies to all applications in  
33             the given spawn request. Indicates that the specified application is being spawned under a  
34             debugger. The resulting application processes are to notify their server when they reach  
35             some application-determined location and pause at that point until receipt of the  
36             **PMIX\_DEBUGGER\_RELEASE** event. The launcher (RM or IL) is responsible for  
37             generating the **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** event when all processes have  
38             indicated they are at the pause point. Launchers that cannot support this operation shall  
39             return an error from the **PMIx\_Spawn** API if this behavior is requested.

40 Note that there is no mechanism by which the PMIx library or the launcher can verify that an  
41 application will recognize and support the **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** request.

1      Debuggers utilizing this attachment method must, therefore, be prepared to deal with the case  
2      where the application fails to recognize and/or honor the request.

3      If the PMIx implementation and/or the host environment support it, debuggers can utilize the  
4      **PMIx\_Query\_info** API to determine which features are available via the  
5      **PMIX\_QUERY\_ATTRIBUTE\_SUPPORT** attribute:

- 6      • **PMIX\_DEBUG\_STOP\_IN\_INIT** by checking **PMIX\_CLIENT\_ATTRIBUTES** for the  
7      **PMIx\_Init** API
- 8      • **PMIX\_DEBUG\_STOP\_ON\_EXEC** by checking **PMIX\_HOST\_ATTRIBUTES** for the  
9      **PMIx\_Spawn** API

10     The target namespace or process (as given by the debugger in the spawn request) shall be provided  
11     to each daemon in its job-level information via the **PMIX\_DEBUG\_TARGET** attribute. Debugger  
12     daemons are responsible for self-determining their specific target process(es), and can then utilize  
13     the **PMIx\_Query\_info** API to obtain information about them (see Fig 14.6) - e.g., to obtain the  
14     PIDs of the local processes to which they need to attach. PMIx provides the  
15     **pmix\_proc\_info\_t** structure for organizing information about a process' PID, location, and  
16     state. Debuggers may request information on a given job at two levels:

- 17     • **PMIX\_QUERY\_PROC\_TABLE** "pmix.qry.ptable" (**char\***)  
18       Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for  
19       each process in the specified namespace, ordered by process job rank. REQUIRED  
20       QUALIFIER: **PMIX\_NSPACE** indicating the namespace whose process table is being  
21       queried
- 22     • **PMIX\_QUERY\_LOCAL\_PROC\_TABLE** "pmix.qry.lptable" (**char\***)  
23       Returns a (**pmix\_data\_array\_t**) array of **pmix\_proc\_info\_t**, one entry for  
24       each process in the specified namespace executing on the same node as the requester,  
25       ordered by process job rank. REQUIRED QUALIFIER: **PMIX\_NSPACE** indicating the  
26       namespace whose process table is being queried

27     Note that the information provided in the returned proctable represents a snapshot in time.

28     Debugger daemons can be started in two ways - either at the same time the application is spawned,  
29     or separately at a later time.

### 30    14.4.1 Co-Location of Debugger Daemons

31     Debugging operations typically require the use of daemons that are located on the same node as the  
32     processes they are attempting to debug. The debugger can, of course, specify its own mapping  
33     method when issuing its spawn request or utilize its own internal launcher to place the daemons.  
34     However, when attaching to a running job, PMIx provides debuggers with a simplified method for  
35     requesting that the launcher associated with the job *co-locate* the required daemons. Debuggers can  
36     request *co-location* of their daemons by adding the following attributes to the **PMIx\_Spawn** used  
37     to spawn them:

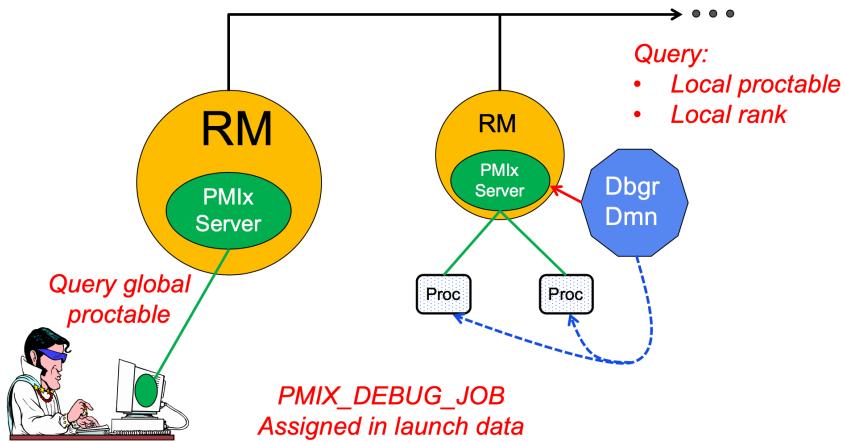


Figure 14.6.: Obtaining proctables

- 1     • **PMIX\_DEBUGGER\_DAEMONS** - indicating that the launcher is being asked to spawn debugger  
2       daemons
- 3     • **PMIX\_DEBUG\_TARGET** - indicating the job or process that is to be debugged. This allows the  
4       launcher to identify the processes to be debugged and their location.
- 5     • **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** - specifies the number of debugger daemons to be  
6       co-located per target process.
- 7     • **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** - specifies the number of debugger daemons to be  
8       co-located per node where at least one target process is executing.

9       Debugger daemons spawned in this manner shall be provided with the typical PMIx information for  
10      their own job, plus the target they are to debug via the **PMIX\_DEBUG\_TARGET** attribute. The  
11      debugger daemons spawned on a given node are responsible for self-determining their specific  
12      target process(es). Note that this method for starting the debugger precludes the use of PMIx  
13      attributes to *pause* the application prior to attaching the debugger. Users are referred to the  
14      **PMIx\_Job\_control** API for possible alternative methods.

## 15     14.4.2 Co-Spawn of Debugger Daemons

16       In the case where a job is being spawned under the control of a debugger, PMIx provides a shortcut  
17      method for spawning the debugger's daemons in parallel with the job. This requires that the  
18      debugger be specified as one of the **pmix\_app\_t** in the same spawn command used to start the  
19      job. The debugger application must include at least the **PMIX\_DEBUGGER\_DAEMONS** attribute  
20      identifying itself as a debugger, and may utilize either a mapping option to direct daemon  
21      placement, or one of the **PMIX\_DEBUG\_DAEMONS\_PER\_PROC** or  
22      **PMIX\_DEBUG\_DAEMONS\_PER\_NODE** directives.

The launcher is not to include the debugger daemons in the job-level info provided to the rest of the **pmix\_app\_t**s, nor in any calculated rank values (e.g., PMIX\_NODE\_RANK or PMIX\_LOCAL\_RANK) in those 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.

### 14.4.3 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. 14.7a), 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 14.4.2.

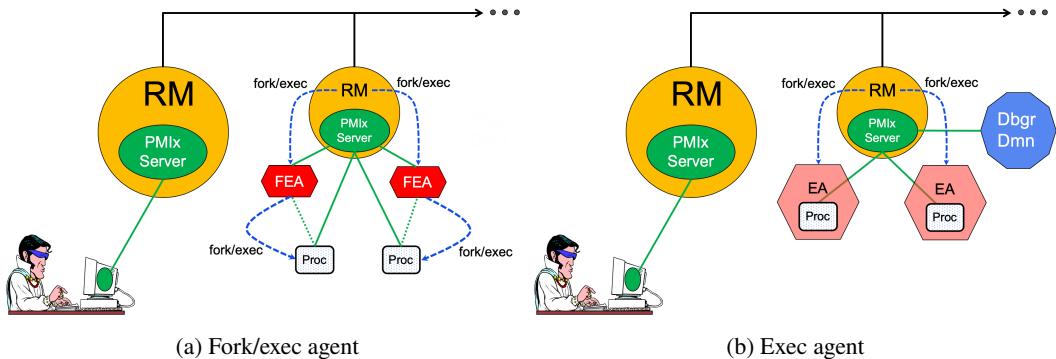


Figure 14.7.: Intermediate agents

In contrast, the **PMIX\_EXEC\_AGENT** attribute (Fig. 14.7b) 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.

## 14.4.4 Debugger-related constants

```
2 PMIX_DEBUG_WAITING_FOR_NOTIFY Job to be debugged is waiting for a release - this is
3 not a value accessed using the PMIx_Get API.
4 PMIX_DEBUGGER_RELEASE Release processes that are paused awaiting debugger attach
5 PMIX_ERR_DEBUGGER_RELEASE Error in debugger release - deprecated in favor of
6 PMIX_DEBUGGER_RELEASE
```

## 14.4.5 Debugger attributes

```
8 Attributes used to assist debuggers - these are values that can either be passed to the PMIx_Spawn
9 APIs or accessed by a debugger itself using the PMIx_Get API with the
10 PMIX_RANK_WILDCARD rank.
```

```
11 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.

```
18 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 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.

```
27 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 when they reach some application-determined location and pause at that point until 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.

```
36 PMIX_DEBUG_JOB "pmix.dbg.job" (char*)
```

Namespace of the job to be debugged - provided to the debugger upon launch (replaced by **PMIX\_DEBUG\_TARGET**)

```
39 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 *info* 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).  
**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).

## 14.5 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.

### 14.5.1 PMIx\_tool\_init

#### Summary

Initialize the PMIx library for operating as a tool, optionally connecting to a specified PMIx server

## Format

1 PMIx v2.0 C

2 pmix\_status\_t  
3 PMIx\_tool\_init(pmix\_proc\_t \*proc,  
4 pmix\_info\_t info[], size\_t ninfo)

5 INOUT proc  
6 pmix\_proc\_t structure (handle)  
7 IN info  
8 Array of pmix\_info\_t structures (array of handles)  
9 IN ninfo  
10 Number of element in the info array (size\_t)  
11 Returns PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant.

12 The following attributes are required to be supported by all PMIx libraries:

13 PMIX\_TOOL\_NSPACE "pmix.tool.nspace" (char\*)  
14 Name of the namespace to use for this tool.

15 PMIX\_TOOL\_RANK "pmix.tool.rank" (uint32\_t)  
16 Rank of this tool.

17 PMIX\_TOOL\_DO\_NOT\_CONNECT "pmix.tool.nocon" (bool)  
18 The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

19 PMIX\_TOOL\_ATTACHMENT\_FILE "pmix.tool.attach" (char\*)  
20 Pathname of file containing connection information to be used for attaching to a specific  
21 server

22 PMIX\_SERVER\_URI "pmix.srvr.uri" (char\*)  
23 URI of the PMIx server to be contacted.

24 PMIX\_TCP\_URI "pmix.tcp.uri" (char\*)  
25 The URI of the PMIx server to connect to, or a file name containing it in the form of  
26 file:<name of file containing it>.

27 PMIX\_SERVER\_PIDINFO "pmix.srvr.pidinfo" (pid\_t)  
28 PID of the target PMIx server for a tool.

29 PMIX\_SERVER\_NSPACE "pmix.srv.nspace" (char\*)  
30 Name of the namespace to use for this PMIx server.

31 PMIX\_CONNECT\_TO\_SYSTEM "pmix.cnct.sys" (bool)  
32 The requester requires that a connection be made only to a local, system-level PMIx server.

33 PMIX\_CONNECT\_SYSTEM\_FIRST "pmix.cnct.sys.first" (bool)

1 Preferentially, look for a system-level PMIx server first.

## Optional Attributes

2 The following attributes are optional for implementers of PMIx libraries:

3 **PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (uint32\_t)  
4 Time in seconds between connection attempts to a PMIx server.

5 **PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)  
6 Maximum number of times to try to connect to PMIx server.

7 **PMIX\_SOCKET\_MODE** "pmix.sockmode" (uint32\_t)  
8 POSIX mode\_t (9 bits valid) If the library supports socket connections, this attribute may  
9 be supported for setting the socket mode.

10 **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (char\*)  
11 If provided, directs that the TCP URI be reported and indicates the desired method of  
12 reporting: '-' for stdout, '+' for stderr, or filename. If the library supports TCP socket  
13 connections, this attribute may be supported for reporting the URI.

14 **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)  
15 Comma-delimited list of devices and/or CIDR notation to include when establishing the  
16 TCP connection. If the library supports TCP socket connections, this attribute may be  
17 supported for specifying the interfaces to be used.

18 **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)  
19 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the  
20 TCP connection. If the library supports TCP socket connections, this attribute may be  
21 supported for specifying the interfaces that are *not* to be used.

22 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)  
23 The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be  
24 supported for specifying the port to be used.

25 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)  
26 The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be  
27 supported for specifying the port to be used.

28 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)  
29 Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections,  
30 this attribute may be supported for disabling it.

31 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)  
32 Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections,  
33 this attribute may be supported for disabling it.

34 **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)

1           Pointer to libevent<sup>1</sup> **event\_base** to use in place of the internal progress thread.

## 2           **Description**

3           Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided  
4           **pmix\_proc\_t** struct. The *info* array is used to pass user requests pertaining to the initialization  
5           and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives  
6           are desired.

7           If called with the **PMIX\_TOOL\_DO\_NOT\_CONNECT** attribute, the PMIx tool library will fully  
8           initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later  
9           point in time, if desired, by calling the **PMIx\_tool\_connect\_to\_server** function. If  
10           provided, the *proc* structure will be set to a zero-length namespace and a rank of  
11           **PMIX\_RANK\_UNDEF** unless the **PMIX\_TOOL\_NSPACE** and **PMIX\_TOOL\_RANK** attributes are  
12           included in the *info* array.

13           In all other cases, the PMIx tool library will automatically attempt to connect to a PMIx server  
14           according to the precedence chain described in Section 14.1. If successful, the function will return  
15           **PMIX\_SUCCESS** and will fill the process structure (if provided) with the assigned namespace and  
16           rank of the tool. The server to which the tool connects will be designated its *primary* server. Note  
17           that each connection attempt in the above precedence chain will retry (with delay between each  
18           retry) a number of times according to the values of the corresponding attributes. Default is no  
19           retries.

20           Note that the PMIx tool library is referenced counted, and so multiple calls to **PMIx\_tool\_init**  
21           are allowed. If the tool is not connected to any server when this API is called, then the tool will  
22           attempt to connect to a server unless the **PMIX\_TOOL\_DO\_NOT\_CONNECT** is included in the call  
23           to API.

## 24          **14.5.2 PMIx\_tool\_finalize**

### 25          **Summary**

26           Finalize the PMIx tool library.

### 27          **Format**

28          PMIx v2.0

C

```
28          pmix_status_t
29          PMIx_tool_finalize(void)
```

C

30           Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

---

<sup>1</sup><http://libevent.org/>

1    **Description**

2    Finalize the PMIx tool library, closing all existing connections to servers. An error code will be  
3    returned if, for some reason, a connection cannot be cleanly terminated — in such cases, the  
4    connection is dropped.

5    **14.5.3 PMIx\_tool\_disconnect**

6    **Summary**

7    Disconnect the PMIx tool from the specified server connection while leaving the tool library  
8    initialized.

9    **Format**

PMIx v4.0

10    pmix\_status\_t  
11    PMIx\_tool\_disconnect (pmix\_proc\_t \*server)

12    **IN server**  
13       pmix\_proc\_t structure (handle)

14    Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

15    **Description**

16    Close the current connection to the specified server, if one has been made, while leaving the PMIx  
17    library initialized. An error code will be returned if, for some reason, the connection cannot be  
18    cleanly terminated - in this case, the connection is dropped. In either case, the library will remain  
19    initialized.

20    Note that if the server being disconnected is the current *primary* server, then all operations  
21    requiring support from a server will return the **PMIX\_ERR\_UNREACH** error until the tool either  
22    designates an existing connection to be the *primary* server or, if no other connections exist, the tool  
23    establishes a connection to a PMIx server.

24    **14.5.4 PMIx\_tool\_connect\_to\_server**

25    **Summary**

26    Establish a connection to a PMIx server.

1      **Format**

2      *pmix\_status\_t*  
 3      **PMIx\_tool\_connect\_to\_server**(*pmix\_proc\_t \*proc,*  
       *pmix\_info\_t info[], size\_t ninfo*)

5      **INOUT proc**  
 6         *pmix\_proc\_t* structure (handle)  
 7      **IN info**  
 8         Array of *pmix\_info\_t* structures (array of handles)  
 9      **IN ninfo**  
 10     Number of element in the *info* array (*size\_t*)  
 11     Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

**Required Attributes**

12    The following attributes are required to be supported by all PMIx libraries:

13    **PMIX\_TOOL\_ATTACHMENT\_FILE** "pmix.tool.attach" (*char\**)  
 14    Pathname of file containing connection information to be used for attaching to a specific  
       server  
 16    **PMIX\_SERVER\_URI** "pmix.srvr.uri" (*char\**)  
 17    URI of the PMIx server to be contacted.  
 18    **PMIX\_TCP\_URI** "pmix.tcp.uri" (*char\**)  
 19    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>.  
 21    **PMIX\_SERVER\_PIDINFO** "pmix.srvr.pidinfo" (*pid\_t*)  
 22    PID of the target PMIx server for a tool.  
 23    **PMIX\_SERVER\_NSPACE** "pmix.srv.nspace" (*char\**)  
 24    Name of the namespace to use for this PMIx server.  
 25    **PMIX\_CONNECT\_TO\_SYSTEM** "pmix.cnct.sys" (*bool*)  
 26    The requester requires that a connection be made only to a local, system-level PMIx server.  
 27    **PMIX\_CONNECT\_SYSTEM\_FIRST** "pmix.cnct.sys.first" (*bool*)  
 28    Preferentially, look for a system-level PMIx server first.  
 29    **PMIX\_PRIMARY\_SERVER** "pmix.pri.srvr" (*bool*)  
 30    The server to which the tool is connecting shall be designated the *primary* server once  
       connection has been accomplished.

1           **Description**

2     Establish a connection to a server. This function can be called at any time by a PMIx tool to create a  
3     new connection to a server. If a specific server is given and the tool is already attached to it, then  
4     the API shall return **PMIX\_SUCCESS!** (**PMIX\_SUCCESS!**) without taking any further action. In  
5     all other cases, the tool will attempt to discover a server using the method described in Section 14.1,  
6     ignoring all candidates to which it is already connected. The **PMIX\_ERR\_UNREACH** error shall be  
7     returned if no new connection is made.

8     The process identifier assigned to this tool is returned in the provided **pmix\_proc\_t** struct.

9     Passing a value of **NULL** for the *proc* parameter is allowed if the user wishes solely to connect to  
10    the PMIx server and does not require return of the identifier at that time.

11            **Advice to PMIx library implementers** 

12     When a tool connects to a server that is under a different namespace manager (e.g., host RM) from  
13     the prior server, the namespace in the identifier of the tool must remain unique in the new universe.  
14     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.

15            **Advice to users** 

16     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    **14.5.5 PMIx\_tool\_set\_server**

18           **Summary**

19     Designate a server as the tool's *primary* server.

20           **Format**

PMIx v4.0

C

21           **pmix\_status\_t**

22           **PMIx\_tool\_set\_server(pmix\_proc\_t \*server)**

C

23           **IN    server**

24            **pmix\_proc\_t** structure (handle)

25     Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

26           **Description**

27     Designate the specified server to be the tool's *primary* server for all subsequent API calls.

## 14.5.6 PMIx\_IOF\_pull

### Summary

Register to receive output forwarded from a set of remote processes.

### Format

PMIx v3.0

C

```
pmix_status_t  
PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,  
               const pmix_info_t directives[], size_t ndirs,  
               pmix_iof_channel_t channel, pmix_iof_cbfunc_t cbfunc,  
               pmix_hdlr_reg_cbfunc_t regcbfunc, void *regcbdata)
```

- 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.cszie" (uint32\_t)

The requested size of the PMIx server cache in bytes for each specified channel. By default, the server is allowed (but not required) to drop all bytes received beyond the max size.

**PMIX\_IOF\_DROP\_OLEDEST** "pmix.iof.old" (bool)

1 In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
2 cache.

3 **PMIX\_IOF\_DROP\_NEWEST "pmix.iof.new" (bool)**

4 In an overflow situation, the PMIx server is to drop any new bytes received until room  
5 becomes available in the cache (default).

▲-----▼ Optional Attributes ▼-----▲

## Optional Attributes

6 The following attributes are optional for PMIx libraries that support IO forwarding:

7 **PMIX\_IOF\_BUFFERING\_SIZE "pmix.iof.bsize" (uint32\_t)**

8 Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
9 specified number of bytes is collected to avoid being called every time a block of IO arrives.  
10 The PMIx tool library will execute the callback and reset the collection counter whenever the  
11 specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
12 the callback upon a call to deregister the respective channel.

13 **PMIX\_IOF\_BUFFERING\_TIME "pmix.iof.btime" (uint32\_t)**

14 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
15 size, this prevents IO from being held indefinitely while waiting for another payload to  
16 arrive.

17 **PMIX\_IOF\_TAG\_OUTPUT "pmix.iof.tag" (bool)**

18 Requests that output be prefixed with the nspace,rank of the source and a string identifying  
19 the channel (**stdout**, **stderr**, etc.)

20 **PMIX\_IOF\_TIMESTAMP\_OUTPUT "pmix.iof.ts" (bool)**

21 Requests that output be marked with the time at which the data was received by the tool -  
22 note that this will differ from the time at which the data was collected from the source

23 **PMIX\_IOF\_XML\_OUTPUT "pmix.iof.xml" (bool)**

24 Requests that output be formatted in XML

25 **Description**

26 Register to receive output forwarded from a set of remote processes.

▼----- Advice to users -----▼

27 Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated  
28 channels to be written to their corresponding **stdout/stderr** file descriptors. Use of  
29 **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but should  
30 be used carefully due to bandwidth and memory footprint considerations.

## 14.5.7 PMIx\_IOF\_deregister

### 2 Summary

3 Deregister from output forwarded from a set of remote processes.

### 4 Format

5 *PMIx v3.0*

C

```
6     pmix_status_t  
7     PMIx_IOF_deregister(size_t iofhdlr,  
8                           const pmix_info_t directives[], size_t ndirs,  
9                           pmix_op_cbfunc_t cbfunc, void *cbdata)
```

#### 9 IN iofhdlr

10 Registration number returned from the [pmix\\_hdlr\\_reg\\_cbfunc\\_t](#) callback from the  
11 call to [PMIx\\_IOF\\_pull \(size\\_t\)](#)

#### 12 IN directives

13 Array of [pmix\\_info\\_t](#) structures (array of handles)

#### 14 IN ndirs

15 Number of elements in the *directives* array (integer)

#### 16 IN cbfunc

17 Callback function to be called when deregistration has been completed. (function reference)

#### 18 IN cbdata

19 Data to be passed to the *cbfunc* callback function (memory reference)

20 Returns one of the following:

- 21 • [PMIX\\_SUCCESS](#), indicating that the request is being processed by the host environment - result  
22 will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback  
23 function prior to returning from the API.
- 24 • [PMIX\\_OPERATION\\_SUCCEEDED](#), indicating that the request was immediately processed and  
25 returned *success* - the *cbfunc* will *not* be called
- 26 • a PMIx error constant indicating either an error in the input or that the request was immediately  
27 processed and failed - the *cbfunc* will *not* be called

### 28 Description

29 Deregister from output forwarded from a set of remote processes.

### Advice to PMIx library implementers

30 Any currently buffered IO should be flushed upon receipt of a deregistration request. All received  
31 IO after receipt of the request shall be discarded.

## 14.5.8 PMIx\_IOC\_push

### Summary

Push data collected locally (typically from `stdin` or a file) to `stdin` of the target recipients.

### Format

PMIx v3.0

C

```
5     pmix_status_t  
6     PMIx_IOC_push(const pmix_proc_t targets[], size_t ntargs,  
7                     pmix_byte_object_t *bo,  
8                     const pmix_info_t directives[], size_t ndirs,  
9                     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

- 10 **IN** **targets**  
11 Array of proc structures identifying desired target processes (array of handles)  
12 **IN** **ntargs**  
13 Number of elements in the *targets* array (integer)  
14 **IN** **bo**  
15 Pointer to `pmix_byte_object_t` containing the payload to be delivered (handle)  
16 **IN** **directives**  
17 Array of `pmix_info_t` structures (array of handles)  
18 **IN** **ndirs**  
19 Number of elements in the *directives* array (integer)  
20 **IN** **directives**  
21 Array of `pmix_info_t` structures (array of handles)  
22 **IN** **cbfunc**  
23 Callback function to be called when operation has been completed. (`pmix_op_cbfunc_t`  
24 function reference)  
25 **IN** **cbdata**  
26 Data to be passed to the *cbfunc* callback function (memory reference)  
27 Returns one of the following:
  - `PMIX_SUCCESS`, indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning from the API.
  - `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
  - a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

## Required Attributes

1 The following attributes are required for PMIx libraries that support IO forwarding:

2   **PMIX\_IOF\_CACHE\_SIZE** "pmix.iof.csizes" (**uint32\_t**)

3         The requested size of the PMIx server cache in bytes for each specified channel. By default,  
4         the server is allowed (but not required) to drop all bytes received beyond the max size.

5   **PMIX\_IOF\_DROP\_OLDEST** "pmix.iof.old" (**bool**)

6         In an overflow situation, the PMIx server is to drop the oldest bytes to make room in the  
7         cache.

8   **PMIX\_IOF\_DROP\_NEWEST** "pmix.iof.new" (**bool**)

9         In an overflow situation, the PMIx server is to drop any new bytes received until room  
10        becomes available in the cache (default).

## Optional Attributes

11 The following attributes are optional for PMIx libraries that support IO forwarding:

12   **PMIX\_IOF\_BUFFERING\_SIZE** "pmix.iof.bsize" (**uint32\_t**)

13         Requests that IO on the specified channel(s) be aggregated in the PMIx tool library until the  
14         specified number of bytes is collected to avoid being called every time a block of IO arrives.  
15         The PMIx tool library will execute the callback and reset the collection counter whenever the  
16         specified number of bytes becomes available. Any remaining buffered data will be *flushed* to  
17         the callback upon a call to deregister the respective channel.

18   **PMIX\_IOF\_BUFFERING\_TIME** "pmix.iof.btime" (**uint32\_t**)

19         Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering  
20         size, this prevents IO from being held indefinitely while waiting for another payload to  
21         arrive.

22   **PMIX\_IOF\_PUSH\_STDIN** "pmix.iof.stdin" (**bool**)

23         Requests that the PMIx library collect the **stdin** of the requester and forward it to the  
24         processes specified in the **PMIx\_IOF\_push** call. All collected data is sent to the same  
25         targets until **stdin** is closed, or a subsequent call to **PMIx\_IOF\_push** is made that  
26         includes the **PMIX\_IOF\_COMPLETE** attribute indicating that forwarding of **stdin** is to  
27         be terminated.

1           **Description**  
2         Called either to:

- 3
  - 4           • push data collected by the caller themselves (typically from **stdin** or a file) to **stdin** of the  
5           target recipients;
  - 6           • request that the PMIx library automatically collect and push the **stdin** of the caller to the target  
7           recipients; or
  - 8           • indicate that automatic collection and transmittal of **stdin** is to stop

---

**Advice to users**

---

8         Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires  
9         the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets.  
10        Use of **PMIX\_RANK\_WILDCARD** to specify all processes in a given namespace is supported but  
11        should be used carefully due to bandwidth and memory footprint considerations.

---

## APPENDIX A

# Python Bindings

---

1 While the PMIx Standard is defined in terms of C-based APIs, there is no intent to limit the use of  
2 PMIx to that specific language. Support for other languages is captured in the Standard by  
3 describing their equivalent syntax for the PMIx APIs and native forms for the PMIx datatypes. This  
4 Appendix specifically deals with Python interfaces, beginning with a review of the PMIx datatypes.  
5 Support is restricted to Python 3 and above - i.e., the Python bindings do not support Python 2.

6 Note: the PMIx APIs have been loosely collected into three Python classes based on their PMIx  
7 “class” (i.e., client, server, and tool). All processes have access to a basic set of the APIs, and  
8 therefore those have been included in the “client” class. Servers can utilize any of those functions  
9 plus a set focused on operations not commonly executed by an application process. Finally, tools  
10 can also act as servers but have their own initialization function.

## A.1 Design Considerations

Several issues arose during design of the Python bindings:

### A.1.1 Error Codes vs Python Exceptions

The C programming language reports errors through the return of the corresponding integer status codes. PMIx has defined a range of negative values for this purpose. However, Python has the option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and return PMIx status codes in lieu of raising exceptions as this method was considered more consistent for those working in both domains.

### A.1.2 Representation of Structured Data

PMIx utilizes a number of C-language structures to efficiently bundle related information. For example, the PMIx process identifier is represented as a struct containing a character array for the namespace and a 32-bit unsigned integer for the process rank. There are several options for translating such objects to Python – e.g., the PMIx process identifier could be represented as a two-element tuple (nspc, rank) or as a dictionary ‘nspc’: name, ‘rank’: 0. Exploration found no discernible benefit to either representation, nor was any clearly identifiable rationale developed that would lead a user to expect one versus the other for a given PMIx data type. Consistency in the translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion. Hence, the decision was made to express all complex datatypes as Python dictionaries.

## 1 A.2 Datatype Definitions

2 PMIx defines a number of datatypes comprised of fixed-size character arrays, restricted range  
3 integers (e.g., `uint32_t`), and structures. Each datatype is represented by a named unsigned 16-bit  
4 integer (`uint16_t`) constant. Users are advised to use the named PMIx constants for indicating  
5 datatypes instead of integer values to ensure compatibility with future PMIx versions.

6 With only a few exceptions, the C-based PMIx datatypes defined in Chapter 3 on page 19 directly  
7 translate to Python. However, Python lacks the size-specific value definitions of C (e.g., `uint8_t`)  
8 and thus some care must be taken to protect against overflow/underflow situations when moving  
9 between the languages. Python bindings that accept values including PMIx datatypes shall  
10 therefore have the datatype and associated value checked for compatibility with their PMIx-defined  
11 equivalents, returning an error if:

- 12 • datatypes not defined by PMIx are encountered
- 13 • provided values fall outside the range of the C-equivalent definition - e.g., if a value identified as  
14     `PMIX_UINT8` lies outside the `uint8_t` range

15 Note that explicit labeling of PMIx datatype, even when Python itself doesn't care, is often required  
16 for the Python bindings to know how to properly interpret and label the provided value when  
17 passing it to the PMIx library.

18 Table A.1 lists the correspondence between datatypes in the two languages.

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                                       | PMIx Name                                                    | Python Definition                                              | Notes                                                                                                                                                |
|----------------------------------------------------|--------------------------------------------------------------|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bool</b>                                        | PMIX_BOOL                                                    | boolean                                                        |                                                                                                                                                      |
| <b>byte</b>                                        | PMIX_BYTE                                                    | A single element byte array (i.e., a byte array of length one) |                                                                                                                                                      |
| <b>char*</b>                                       | PMIX_STRING                                                  | string                                                         |                                                                                                                                                      |
| <b>size_t</b>                                      | PMIX_SIZE                                                    | integer                                                        |                                                                                                                                                      |
| <b>pid_t</b>                                       | PMIX_PID                                                     | integer                                                        | value shall be limited to the <b>uint32_t</b> range                                                                                                  |
| <b>int, int8_t, int16_t, int32_t, int64_t</b>      | PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64      | integer                                                        | value shall be limited to its corresponding range                                                                                                    |
| <b>uint, uint8_t, uint16_t, uint32_t, uint64_t</b> | PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64 | integer                                                        | value shall be limited to its corresponding range                                                                                                    |
| <b>float, double</b>                               | PMIX_FLOAT, PMIX_DOUBLE                                      | float                                                          | value shall be limited to its corresponding range                                                                                                    |
| <b>struct timeval</b>                              | PMIX_TIMEVAL                                                 | {'sec': sec, 'usec': microsec}                                 | each field is an integer value                                                                                                                       |
| <b>time_t</b>                                      | PMIX_TIME                                                    | integer                                                        | limited to positive values                                                                                                                           |
| <b>pmix_data_type_t</b>                            | PMIX_DATA_TYPE                                               | integer                                                        | value shall be limited to the <b>uint16_t</b> range                                                                                                  |
| <b>pmix_status_t</b>                               | PMIX_STATUS                                                  | integer                                                        |                                                                                                                                                      |
| <b>pmix_key_t</b>                                  | N/A                                                          | string                                                         | The string's length shall be limited to one less than the size of the <b>pmix_key_t</b> array (to reserve space for the terminating <b>NULL</b> )    |
| <b>pmix_nspace_t</b>                               | N/A                                                          | string                                                         | The string's length shall be limited to one less than the size of the <b>pmix_nspace_t</b> array (to reserve space for the terminating <b>NULL</b> ) |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                    | PMIx Name         | Python Definition                                                                                                                            | Notes                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>pmix_rank_t</code>        | PMIX_PROC_RANK    | integer                                                                                                                                      | value shall be limited to the <code>uint32_t</code> range excepting the reserved values near <code>UINT32_MAX</code>                                                                                                                                                                                                                                      |
| <code>pmix_proc_t</code>        | PMIX_PROC         | {'nspace': nspace, 'rank': rank}                                                                                                             | <i>nspace</i> is a Python string and <i>rank</i> is an integer value. The <i>nspace</i> string's length shall be limited to one less than the size of the <code>pmix_nspace_t</code> array (to reserve space for the terminating <code>NULL</code> ), and the <i>rank</i> value shall conform to the constraints associated with <code>pmix_rank_t</code> |
| <code>pmix_byte_object_t</code> | PMIX_BYT_E_OBJECT | {'bytes': bytes, 'size': size}                                                                                                               | <i>bytes</i> is a Python byte array and <i>size</i> is the integer number of bytes in that array.                                                                                                                                                                                                                                                         |
| <code>pmix_persistence_t</code> | PMIX_PERSISTENCE  | integer                                                                                                                                      | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                                                                                                                                  |
| <code>pmix_scope_t</code>       | PMIX_SCOPE        | integer                                                                                                                                      | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                                                                                                                                  |
| <code>pmix_data_range_t</code>  | PMIX_RANGE        | integer                                                                                                                                      | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                                                                                                                                  |
| <code>pmix_proc_state_t</code>  | PMIX_PROC_STATE   | integer                                                                                                                                      | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                                                                                                                                  |
| <code>pmix_proc_info_t</code>   | PMIX_PROC_INFO    | {'proc': {'nspace': nspace, 'rank': rank}, 'hostname': hostname, 'executable': executable, 'pid': pid, 'exitcode': exitcode, 'state': state} | <i>proc</i> is a Python <code>proc</code> dictionary; <i>hostname</i> and <i>executable</i> are Python strings; and <i>pid</i> , <i>exitcode</i> , and <i>state</i> are Python integers                                                                                                                                                                   |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                        | PMIx Name            | Python Definition                                                                      | Notes                                                                                                                                                                                                                                                                                                          |
|-------------------------------------|----------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>pmix_data_array_t</code>      | PMIX_DATA_ARRAY      | {'type': type, 'array': array}                                                         | <i>type</i> is the PMIx type of object in the array and <i>array</i> is a Python <i>list</i> containing the individual array elements. Note that <i>array</i> can consist of <i>any</i> PMIx types, including (for example) a Python <code>info</code> object that itself contains an <code>array</code> value |
| <code>pmix_info_directives_t</code> | PMIX_INFO_DIRECTIVES | integer                                                                                | value shall be limited to the <code>uint32_t</code> range                                                                                                                                                                                                                                                      |
| <code>pmix_alloc_directive_t</code> | PMIX_ALLOC_DIRECTIVE | integer                                                                                | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                                                                                       |
| <code>pmix_iof_channel_t</code>     | PMIX_IOF_CHANNEL     | integer                                                                                | value shall be limited to the <code>uint16_t</code> range                                                                                                                                                                                                                                                      |
| <code>pmix_envar_t</code>           | PMIX_ENVAR           | {'envar': envar, 'value': value, 'separator': separator}                               | <i>envar</i> and <i>value</i> are Python strings, and <i>separator</i> a single-character Python string                                                                                                                                                                                                        |
| <code>pmix_value_t</code>           | PMIX_VALUE           | {'value': value, 'val_type': type}                                                     | <i>type</i> is the PMIx datatype of <i>value</i> , and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype                                                                                                                                                |
| <code>pmix_info_t</code>            | PMIX_INFO            | {'key': key, 'flags': flags, 'value': value, 'val_type': type}                         | <i>key</i> is a Python string <code>key</code> , <i>flags</i> is a bitmask of <code>info directives</code> , <i>type</i> is the PMIx datatype of <i>value</i> , and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype                                   |
| <code>pmix_pdata_t</code>           | PMIX_PDATA           | {'proc': {'nspc': nspace, 'rank': rank}, 'key': key, 'value': value, 'val_type': type} | <i>proc</i> is a Python <code>proc</code> dictionary; <i>key</i> is a Python string <code>key</code> ; <i>type</i> is the PMIx datatype of <i>value</i> ; and <i>value</i> is the associated value expressed in the appropriate Python form for the specified datatype                                         |

Table A.1.: C-to-Python Datatype Correspondence

| C-Definition                   | PMIx Name       | Python Definition                                                                | Notes                                                                                                                                                                                                                                               |
|--------------------------------|-----------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>pmix_app_t</code>        | PMIX_APP        | {'cmd': cmd, 'argv': [argv], 'env': [env], 'maxprocs': maxprocs, 'info': [info]} | <i>cmd</i> is a Python string; <i>argv</i> and <i>env</i> are Python lists containing Python strings; <i>maxprocs</i> is an integer; and <i>info</i> is a Python list of <code>info</code> values                                                   |
| <code>pmix_query_t</code>      | PMIX_QUERY      | {'keys': [keys], 'qualifiers': [info]}                                           | <i>keys</i> is a Python list of Python strings, and <i>qualifiers</i> is a Python list of <code>info</code> values                                                                                                                                  |
| <code>pmix_regattr_t</code>    | PMIX_REGATTR    | {'name': name, 'key': key, 'type': type, 'info': [info], 'description': [desc]}  | <i>name</i> and <i>string</i> are Python strings; <i>type</i> is the PMIx datatype for the attribute's value; <i>info</i> is a Python list of <code>info</code> values; and <i>description</i> is a list of Python strings describing the attribute |
| <code>pmix_link_state_t</code> | PMIX_LINK_STATE | integer                                                                          | value shall be limited to the <code>uint8_t</code> range                                                                                                                                                                                            |

## 1 A.2.1 Example

2 Converting a C-based program to its Python equivalent requires translation of the relevant  
3 datatypes as well as use of the appropriate API form. An example small program may help  
4 illustrate the changes. Consider the following C-based program snippet:

```
5 #include <pmix.h>
6 ...
7
8 pmix_info_t info[2];
9
10 PMIX_INFO_LOAD(&info[0], PMIX_PROGRAMMING_MODEL, "TEST", PMIX_STRING)
11 PMIX_INFO_LOAD(&info[1], PMIX_MODEL_LIBRARY_NAME, "PMIX", PMIX_STRING)
12
13 rc = PMIx_Init(&myproc, info, 2);
14
15 PMIX_INFO_DESTRUCT(&info[0]); // free the copied string
16 PMIX_INFO_DESTRUCT(&info[1]); // free the copied string
17
```

18 Moving to the Python version requires that the `pmix_info_t` be translated to the Python `info`  
19 equivalent, and that the returned information be captured in the return parameters as opposed to a  
20 pointer parameter in the function call, as shown below:

```
21 import pmix
22 ...
23
24 myclient = PMIxClient()
25 info = [{'key':PMIX_PROGRAMMING_MODEL,
26           'value':'TEST', 'val_type'key':PMIX_MODEL_LIBRARY_NAME,
28           'value':'PMIX', 'val_type
```

31 Note the use of the `PMIX_STRING` identifier to ensure the Python bindings interpret the provided  
32 string value as a PMIx "string" and not an array of bytes.

## 33 A.3 Callback Function Definitions

### 34 A.3.1 IOF Delivery Function

#### 35 Summary

36 Callback function for delivering forwarded IO to a process

1 Format  
2 def iofcbfunc(iofhdlr:integer, channel:integer,  
3 source:dict, payload:dict, info:list)  
4 IN iofhdlr  
5 Registration number of the handler being invoked (integer)  
6 IN channel  
7 Python channel bitmask identifying the channel the data arrived on (integer)  
8 IN source  
9 Python proc identifying the namespace/rank of the process that generated the data (dict)  
10 IN payload  
11 Python byteobject containing the data (dict)  
12 IN info  
13 List of Python info provided by the source containing metadata about the payload. This  
14 could include PMIX\_IOF\_COMPLETE (list)  
15 Returns: nothing  
16 See pmix\_iof\_cbfunc\_t for details

## 17 A.3.2 Event Handler

18           **Summary**  
19           Callback function for event handlers  
  
20           **Format**

---

21           `def evhandler(evhdlr:integer, status:integer,`  
22                           `source:dict, info:list, results:list)`

---

23           **IN    iofhdlr**  
24           Registration number of the handler being invoked (integer)  
25           **IN    status**  
26           Status associated with the operation (integer)  
27           **IN    source**  
28           Python `proc` identifying the namespace/rank of the process that generated the event (dict)  
29           **IN    info**  
30           List of Python `info` provided by the source containing metadata about the event (list)  
31           **IN    results**  
32           List of Python `info` containing the aggregated results of all prior evhandlers (list)  
  
33           Returns:

- *rc* - Status returned by the event handler's operation (integer)
  - *results* - List of Python `info` containing results from this event handler's operation on the event (list)

See `pmix_notification_fn_t` for details

### A.3.3 Server Module Functions

The following definitions represent functions that may be provided to the PMIx server library at time of initialization for servicing of client requests. Module functions that are not provided default to returning "not supported" to the caller.

### A.3.3.1 Client Connected

## Summary

Notify the host server that a client connected to this server.

## Format

PMIx v4.0

## Python

```
def clientconnected(proc:dict is not None)
```

## Python

IN proc

Python `proc` identifying the namespace/rank of the process that connected (dict)

### Returns:

- `rc` - **PMIX\_SUCCESS** or a PMIx error code indicating the connection should be rejected (integer)

See `pmix server client connected fn t` for details

### A.3.3.2 Client Finalized

## Summary

Notify the host environment that a client called **PMIx\_Finalize**.

## Format

PMIx v4.0

# Python

```
def clientfinalized(proc:dict is not None):
```

## Python

IN proc

Python `proc` identifying the namespace/rank of the process that finalized (dict)

Returns: nothing

See `pmix_server_client_finalized_fn_t` for details.

1 **A.3.3.3 Client Aborted**

2 **Summary**

3 Notify the host environment that a local client called **PMIx\_Abort**.

4 **Format**

5 *PMIx v4.0*

Python

6 `def clientaborted(args:dict is not None)`

Python

7 **IN args**

8 Python dictionary containing:

- 'caller': Python **proc** 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 **proc** identifying the namespace/rank of the processes to be aborted (list)

9 Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

10 See [pmix\\_server\\_abort\\_fn\\_t](#) for details

11 **A.3.3.4 Fence**

12 **Summary**

13 At least one client called either **PMIx\_Fence** or **PMIx\_Fence\_nb**

14 **Format**

15 *PMIx v4.0*

Python

16 `def fence(args:dict is not None)`

Python

17 **IN args**

18 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)
- 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)

19 Returns:

- 1     • *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)  
2  
3     • *data* - Python bytearray containing the aggregated data from all participants (bytearray)

4     See [pmix\\_server\\_fencenb\\_fn\\_t](#) for details

### A.3.3.5 Direct Modex

#### Summary

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return a direct modex blob for that proc.

#### Format

PMIx v4.0

def dmodex(args:dict is not None)

Python

Python

#### 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)

### A.3.3.6 Publish

#### Summary

Publish data per the PMIx API specification.

#### Format

PMIx v4.0

def publish(args:dict is not None)

Python

Python

#### 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

1 **A.3.3.7 Lookup**

2      **Summary**

3        Lookup published data.

4      **Format**

PMIx v4.0

Python

def lookup(args:dict is not None)

Python

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.

18     **Format**

PMIx v4.0

Python

def unpublish(args:dict is not None)

Python

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

### 1 A.3.3.9 Spawn

#### 2 Summary

3 Spawn a set of applications/processes as per the [PMIx\\_Spawn](#) API.

#### 4 Format

5 *PMIx v4.0*

```
def spawn(args:dict is not None)
```

Python

Python

#### 6 IN args

7 Python dictionary containing:

- 8 • 'proc': Python [proc](#) of process making the request (dict)
- 9 • 'jobinfo': Optional list of Python [info](#) job-level directives and information (list)
- 10 • 'apps': List of Python [app](#) describing applications to be spawned (list)

11 Returns:

- 12 • *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- 13 • *nspace* - Python string containing namespace of the spawned job (str)

14 See [pmix\\_server\\_spawn\\_fn\\_t](#) for details

### 15 A.3.3.10 Connect

#### 16 Summary

17 Record the specified processes as *connected*.

#### 18 Format

19 *PMIx v4.0*

```
def connect(args:dict is not None)
```

Python

Python

#### 20 IN args

21 Python dictionary containing:

- 22 • 'procs': List of Python [proc](#) identifying the namespace/rank of the participating  
23 processes (list)
- 24 • 'directives': Optional list of Python [info](#) containing directives controlling the operation  
25 (list)

26 Returns:

- 27 • *rc* - [PMIX\\_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

28 See [pmix\\_server\\_connect\\_fn\\_t](#) for details

1 **A.3.3.11 Disconnect**

2       **Summary**

3       Disconnect a previously connected set of processes.

4       **Format**

5       *PMIx v4.0*

6       `def disconnect(args:dict is not None)`

7                    *Python*

8                    *Python*

9       **IN args**

10      Python dictionary containing:

- 11
- '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)

12      Returns:

- 13
- `rc - PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

14      See `pmix_server_disconnect_fn_t` for details

15 **A.3.3.12 Register Events**

16       **Summary**

17       Register to receive notifications for the specified events.

18       **Format**

19       *PMIx v4.0*

20       `def register_events(args:dict is not None)`

21                    *Python*

22                    *Python*

23       **IN args**

24      Python dictionary containing:

- 25
- 'codes': List of Python integers (list)
  - 'directives': Optional list of Python `info` containing directives controlling the operation (list)

26      Returns:

- 27
- `rc - PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

See `pmix_server_register_events_fn_t` for details

### 1 A.3.3.13 Deregister Events

#### 2      Summary

3      Deregister to receive notifications for the specified events.

#### 4      Format

5      PMIx v4.0

```
6      def deregister_events(args:dict is not None)
7                    Python
```

8                    Python

#### 9      IN args

10     Python dictionary containing:

- 'codes': List of Python integers (list)

11     Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

12     See [pmix\\_server\\_deregister\\_events\\_fn\\_t](#) for details

### 12 A.3.3.14 Notify Event

#### 13      Summary

14      Notify the specified range of processes of an event.

#### 15      Format

16      PMIx v4.0

```
17      def notify_event(args:dict is not None)
18                    Python
```

19                    Python

#### 20      IN args

21     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)

22     Returns:

- *rc* - **PMIX\_SUCCESS** or a PMIx error code indicating the operation failed (integer)

23     See [pmix\\_server\\_notify\\_event\\_fn\\_t](#) for details

### 26 A.3.3.15 Query

#### 27      Summary

28      Query information from the resource manager.

```
1      Format  
2      PMIx v4.0  
3      def query(args:dict is not None)  
4      Python  
5      IN  args  
6          Python dictionary containing:  
7              • 'source': Python proc of requesting process (dict)  
8              • 'queries': List of Python query directives (list)  
9  
10     Returns:  
11        • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
12        • info - List of Python info containing the returned results (list)  
13  
14     See pmix\_server\_query\_fn\_t for details
```

### A.3.3.16 Tool Connected

#### Summary

Register that a tool has connected to the server.

```
14      Format  
15      PMIx v4.0  
16      def tool_connected(args:dict is not None)  
17      Python  
18      IN  args  
19          Python dictionary containing:  
20              • 'directives': Optional list of Python info info on the connecting tool (list)  
21  
22     Returns:  
23        • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
24        • proc - Python proc containing the assigned namespace:rank for the tool (dict)  
25  
26     See pmix\_server\_tool\_connection\_fn\_t for details
```

### A.3.3.17 Log

#### Summary

Log data on behalf of a client.

```
1      Format  
2      PMIx v4.0  
3      def log(args:dict is not None)  
4      Python  
5      IN  args  
6          Python dictionary containing:  
7              • 'source': Python proc of requesting process (dict)  
8              • 'data': Optional list of Python info containing data to be logged (list)  
9              • 'directives': Optional list of Python info containing directives (list)  
10         Returns:  
11             • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
12             See pmix\_server\_log\_fn\_t for details
```

### A.3.3.18 Allocate Resources

```
12     Summary  
13     Request allocation operations on behalf of a client.  
14     Format  
15     PMIx v4.0  
16     def allocate(args:dict is not None)  
17     Python  
18     IN  args  
19         Python dictionary containing:  
20             • 'source': Python proc of requesting process (dict)  
21             • 'action': Python allocdir specifying requested action (integer)  
22             • 'directives': Optional list of Python info containing directives (list)  
23         Returns:  
24             • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
25             • refarginfo - List of Python info containing results of requested operation (list)  
26             See pmix\_server\_alloc\_fn\_t for details
```

### A.3.3.19 Job Control

```
26     Summary  
27     Execute a job control action on behalf of a client.
```

```
1      Format  
2      PMIx v4.0          Python  
3      def job_control(args:dict is not None)  
4      Python  
5      IN   args  
6      Python dictionary containing:  
7          • 'source': Python proc of requesting process (dict)  
8          • 'targets': List of Python proc specifying target processes (list)  
9          • 'directives': Optional list of Python info containing directives (list)  
10     Returns:  
11         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
12     See pmix\_server\_job\_control\_fn\_t for details
```

### A.3.3.20 Monitor

#### Summary

Request that a client be monitored for activity.

```
14     Format  
15     PMIx v4.0          Python  
16     def monitor(args:dict is not None)  
17     Python  
18     IN   args  
19     Python dictionary containing:  
20         • 'source': Python proc of requesting process (dict)  
21         • 'monitor': Python info attribute indicating the type of monitor being requested (dict)  
22         • 'error': Status code to be used when generating an event notification (integer) alerting that  
23             the monitor has been triggered.  
24         • 'directives': Optional list of Python info containing directives (list)  
25     Returns:  
26         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
27     See pmix\_server\_monitor\_fn\_t for details
```

### A.3.3.21 Get Credential

#### Summary

Request a credential from the host environment

```
1      Format  
2      PMIx v4.0          Python  
3      def get_credential(args:dict is not None)  
4      Python  
5      IN  args  
6          Python dictionary containing:  
7              • 'source': Python proc of requesting process (dict)  
8              • 'directives': Optional list of Python info containing directives (list)  
9      Returns:  
10         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
11         • cred - Python byteobject containing returned credential (dict)  
12         • info - List of Python info containing any additional info about the credential (list)  
13     See pmix\_server\_get\_cred\_fn\_t for details
```

### A.3.3.22 Validate Credential

```
13    Summary  
14    Request validation of a credential
```

```
15    Format  
16    PMIx v4.0          Python  
17    def validate_credential(args:dict is not None)  
18    Python  
19    IN  args  
20        Python dictionary containing:  
21            • 'source': Python proc of requesting process (dict)  
22            • 'credential': Python byteobject containing credential (dict)  
23            • 'directives': Optional list of Python info containing directives (list)  
24        Returns:  
25            • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
26            • info - List of Python info containing any additional info from the credential (list)  
27     See pmix\_server\_validate\_cred\_fn\_t for details
```

### A.3.3.23 IO Forward

```
27    Summary  
28    Request the specified IO channels be forwarded from the given array of processes.
```

1      **Format**

2      `def iof_pull(args:dict is not None)`

Python

3      **IN args**

4      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)

8      Returns:

- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

10     See `pmix_server_iof_fn_t` for details

11    **A.3.3.24 IO Push**

12    **Summary**

13    Pass standard input data to the host environment for transmission to specified recipients.

14    **Format**

15    `def iof_push(args:dict is not None)`

Python

16    **IN args**

17    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)

22    Returns:

- `rc` - `PMIX_SUCCESS` or a PMIx error code indicating the operation failed (integer)

24    See `pmix_server_stdin_fn_t` for details

25    **A.3.3.25 Group Operations**

26    **Summary**

27    Request group operations (construct, destruct, etc.) on behalf of a set of processes.

```
1      Format  
1  PMIx v4.0  Python  
2      def group(args:dict is not None)  
2  Python  
3  IN  args  
4      Python dictionary containing:  
5          • 'op': Operation host is to perform on the specified group (integer)  
6          • 'group': String identifier of target group (str)  
7          • 'procs': List of Python proc of participating processes (dict)  
8          • 'directives': Optional list of Python info containing directives (list)  
9      Returns:  
10         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
11         • refarginfo - List of Python info containing results of requested operation (list)  
12      See pmix\_server\_grp\_fn\_t for details
```

### A.3.3.26 Fabric Operations

```
14     Summary  
15     Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.  
16     Format  
16  PMIx v4.0  Python  
17     def fabric(args:dict is not None)  
17  Python  
18  IN  args  
19      Python dictionary containing:  
20          • 'source': Python proc of requesting process (dict)  
21          • 'op': Operation host is to perform on the specified fabric (integer)  
22          • 'directives': Optional list of Python info containing directives (list)  
23      Returns:  
24         • rc - PMIX_SUCCESS or a PMIx error code indicating the operation failed (integer)  
25         • refarginfo - List of Python info containing results of requested operation (list)  
26      See pmix\_server\_fabric\_fn\_t for details
```

## 1 A.4 PMIxClient

2 The client Python class is by far the richest in terms of APIs as it houses all the APIs that an  
3 application might utilize. Due to the datatype translation requirements of the C-Python interface,  
4 only the blocking form of each API is supported – providing a Python callback function directly to  
5 the C interface underlying the bindings was not a supportable option.

### 6 A.4.1 Client.init

#### 7 Summary

8 Initialize the PMIx client library after obtaining a new PMIxClient object

#### 9 Format

10 *PMIx v4.0*  *Python*   
*rc, proc = myclient.init(info:list)*  *Python* 

#### 11 IN info

12 List of Python *info* dictionaries (list)

13 Returns:

- 14 • *rc* - *PMIX\_SUCCESS* or a negative value corresponding to a PMIx error constant (integer)  
15 • *proc* - a Python *proc* dictionary (dict)

16 See [PMIx\\_Init](#) for description of all relevant attributes and behaviors

## 17 A.4.2 Client.initialized

#### 18 Format

19 *PMIx v4.0*  *Python*   
*rc = myclient.initialized()*  *Python* 

20 Returns:

- 21 • *rc* - a value of **1** (true) will be returned if the PMIx library has been initialized, and **0** (false)  
22 otherwise (integer)

23 See [PMIx\\_Initialized](#) for description of all relevant attributes and behaviors

1 **A.4.3 Client.get\_version**

2       **Format**

3     *PMIx v4.0*

4     *vers = myclient.get\_version()*

5              Python

6              Python

7     Returns:

- 8       • *vers* - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)

9     See [PMIx\\_Get\\_version](#) for description of all relevant attributes and behaviors

10      **A.4.4 Client.finalize**

11       **Summary**

12     Finalize the PMIx client library.

13       **Format**

14     *PMIx v4.0*

15     *rc = myclient.finalize(info:list)*

16              Python

17              Python

18       **IN info**

19       List of Python **info** dictionaries (list)

20     Returns:

- 21       • *rc* - [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

22     See [PMIx\\_Finalize](#) for description of all relevant attributes and behaviors

23      **A.4.5 Client.abort**

24       **Summary**

25     Request that the provided list of procs be aborted

1      **Format**

Python

2      `rc = myclient.abort(status:integer, msg:str, targets:list)`

Python

3      **IN status**

4            PMIx status to be returned on exit (integer)

5      **IN msg**

6            String message to be printed (string)

7      **IN targets**

8            List of Python `proc` dictionaries (list)

9      Returns:

- 10     • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

11     See `PMIx_Abort` for description of all relevant attributes and behaviors

12    **A.4.6 Client.store\_internal**

13    **Summary**

14    Store some data locally for retrieval by other areas of the process

15    **Format**

Python

16    `rc = myclient.store_internal(proc:dict, key:str, value:dict)`

Python

17    **IN proc**

18            Python `proc` dictionary of the process being referenced (dict)

19    **IN key**

20            String key of the data (string)

21    **IN value**

22            Python `value` dictionary (dict)

23    Returns:

- 24     • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

25     See `PMIx_Store_Internal` for details

26    **A.4.7 Client.put**

27    **Summary**

28    Push a key/value pair into the client's namespace.

1           **Format**  
2        *PMIx v4.0*      Python  
3        rc = myclient.put(scope:integer, key:str, value:dict)  
4            Python  
5        IN scope  
6            Scope of the data being posted (integer)  
7        IN key  
8            String key of the data (string)  
9        IN value  
10          Python **value** dictionary (dict)  
11        Returns:  
12          • rc - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
13        See **PMIx\_Put** for description of all relevant attributes and behaviors

## 12 A.4.8 Client.commit

13           **Summary**  
14        Push all previously **PMIxClient.put** values to the local PMIx server.  
15           **Format**  
16        *PMIx v4.0*      Python  
17        rc = myclient.commit()  
18            Python  
19        Returns:  
20          • rc - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
21        See **PMIx\_Commit** for description of all relevant attributes and behaviors

## 20 A.4.9 Client.fence

21           **Summary**  
22        Execute a blocking barrier across the processes identified in the specified list

```
1 Format
2 PMIx v4.0
3 rc = myclient.fence(peers:list, directives:list)
4
5 IN peers
6 List of Python proc dictionaries (list)
7 IN directives
8 List of Python info dictionaries (list)
9
10 Returns:
11     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
12
13 See PMIx_Fence for description of all relevant attributes and behaviors
```

## 10 A.4.10 Client.get

11 **Summary**  
12 Retrieve a key/value pair

13 **Format**

14 *PMIx v4.0*  Python   
`rc, val = myclient.get(proc:dict, key:str, directives:list)`

15 **IN proc**  
16 Python **proc** whose data is being requested (dict)  
17 **IN key**  
18 Python string key of the data to be returned (str)  
19 **IN directives**  
20 List of Python **info** dictionaries (list)

21 Returns:  
22 • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
23 • *val* - Python **value** containing the returned data (dict)

24 See **PMIx Get** for description of all relevant attributes and behaviors

## 25 A.4.11 Client.publish

26                   **Summary**  
27                   Publish data for later access via **PMIx\_Lookup**

```
1      Format  
2      rc = myclient.publish(directives:list) Python  
3      IN  directives  
4          List of Python info dictionaries containing data to be published and directives (list)  
5      Returns:  
6          • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
7      See PMIx\_Publish for description of all relevant attributes and behaviors
```

## 8 A.4.12 Client.lookup

```
9      Summary  
10     Lookup information published by this or another process with PMIx\_Publish.  
11      Format  
12      rc,info = myclient.lookup(pdata:list, directives:list) Python  
13      IN  pdata  
14          List of Python pdata dictionaries identifying data to be retrieved (list)  
15      IN  directives  
16          List of Python info dictionaries (list)  
17      Returns:  
18          • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
19          • info - Python list of info containing the returned data (list)  
20      See PMIx\_Lookup for description of all relevant attributes and behaviors
```

## 21 A.4.13 Client.unpublish

```
22      Summary  
23      Delete data published by this process with PMIx\_Publish.
```

1           **Format**  
2        *PMIx v4.0*      Python  
3        *rc = myclient.unpublish(keys:list, directives:list)*  
4            Python  
5           **IN keys**  
6           List of Python string keys identifying data to be deleted (list)  
7           **IN directives**  
8           List of Python `info` dictionaries (list)  
9           Returns:  
10           

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

  
11           See [PMIx\\_Unpublish](#) for description of all relevant attributes and behaviors

## 10 A.4.14 Client.spawn

11           **Summary**  
12        Spawn a new job.  
13           **Format**  
14        *PMIx v4.0*      Python  
15        *rc, nspace = myclient.spawn(jobinfo:list, apps:list)*  
16            Python  
17           **IN jobinfo**  
18           List of Python `info` dictionaries (list)  
19           **IN apps**  
20           List of Python `app` dictionaries (list)  
21           Returns:  
22           

- *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- *nspace* - Python `nspace` of the new job (dict)

  
23           See [PMIx\\_Spawn](#) for description of all relevant attributes and behaviors

## 23 A.4.15 Client.connect

24           **Summary**  
25        Connect namespaces.

1           **Format**

2       `rc = myclient.connect(peers:list, directives:list)`

3           **IN peers**

4           List of Python `proc` dictionaries (list)

5           **IN directives**

6           List of Python `info` dictionaries (list)

7           Returns:

- 8
  - `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

9           See [PMIx\\_Connect](#) for description of all relevant attributes and behaviors

10          **A.4.16 Client.disconnect**

11          **Summary**

12          Disconnect namespaces.

13          **Format**

14        `rc = myclient.disconnect(peers:list, directives:list)`

15           **IN peers**

16           List of Python `proc` dictionaries (list)

17           **IN directives**

18           List of Python `info` dictionaries (list)

19           Returns:

- 20
  - `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

21           See [PMIx\\_Disconnect](#) for description of all relevant attributes and behaviors

22          **A.4.17 Client.resolve\_peers**

23          **Summary**

24          Return list of processes within the specified `nspcace` on the given node.

```
1 Format
PMIx v4.0
2 rc,procs = myclient.resolve_peers(node:str, nspace:str)
3 IN node
4 Name of node whose processes are being requested (str)
5 IN nspace
6 Python nspace whose processes are to be returned (str)
7 Returns:
8 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9 • procs - List of Python proc dictionaries (list)
10 See PMIx_Resolve_peers for description of all relevant attributes and behaviors
```

## 11 A.4.18 Client.resolve\_nodes

**Summary**  
Return list of nodes hosting processes within the specified `namespace`.

```
14          Format
15          PMIx v4.0
16          rc, nodes = myclient.resolve_nodes(nspace:str)
17
18          IN  nspace
19          Python nspace (str)
20
21          Returns:
22
23          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
24
25          • nodes - List of Python string node names (list)
26
27          See PMIx Resolve nodes for description of all relevant attributes and behaviors
```

## **22 A.4.19 Client.query**

**Summary**  
Query information about the system in general

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myclient.query(queries:list)  
4      Python          Python  
5      IN   queries  
6          List of Python query dictionaries (list)  
7      Returns:  
8          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9          • info - List of Python info containing results of the query (list)  
10         See PMIx_Query_info_nb for description of all relevant attributes and behaviors
```

## 9 A.4.20 Client.log

```
10     Summary  
11     Log data to a central data service/store  
12     Format  
13     PMIx v4.0          Python  
14     rc = myclient.log(data:list, directives:list)  
15     Python          Python  
16     IN   data  
17         List of Python info (list)  
18     IN   directives  
19         Optional list of Python info (list)  
20     Returns:  
21         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22         See PMIx_Log for description of all relevant attributes and behaviors
```

## 21 A.4.21 Client.allocate

```
22     Summary  
23     Request an allocation operation from the host resource manager.
```

1           **Format**  
2        *rc, info = myclient.allocate(request:integer, directives:list)*  
3        IN **request**  
4           Python **allocadir** specifying requested operation (integer)  
5        IN **directives**  
6           List of Python **info** describing request (list)  
7        Returns:  
8           • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
9           • *info* - List of Python **info** containing results of the request (list)  
10      See **PMIx\_Allocation\_request\_nb** for description of all relevant attributes and behaviors

## 11     **A.4.22 Client.job\_ctrl**

12       **Summary**  
13      Request a job control action  
  
14       **Format**  
15       *rc, info = myclient.job\_ctrl(targets:list, directives:list)*  
16       IN **targets**  
17           List of Python **proc** specifying targets of requested operation (integer)  
18       IN **directives**  
19           List of Python **info** describing operation to be performed (list)  
20       Returns:  
21           • *rc* - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
22           • *info* - List of Python **info** containing results of the request (list)  
23      See **PMIx\_Job\_control\_nb** for description of all relevant attributes and behaviors

## 24     **A.4.23 Client.monitor**

25       **Summary**  
26      Request that something be monitored

```
1 Format
PMIx v4.0 Python
2 rc,info = myclient.monitor(monitor:dict, error_code:integer, directives:list)
Python
3 IN monitor
4 Python info specifying specifying the type of monitor being requested (dict)
5 IN error_code
6 Status code to be used when generating an event notification alerting that the monitor has
7 been triggered (integer)
8 IN directives
9 List of Python info describing request (list)

10 Returns:
11 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
12 • info - List of Python info containing results of the request (list)
13 See PMIx Process monitor nb for description of all relevant attributes and behaviors
```

## 14 A.4.24 Client.get credential

**15                   Summary**  
16                   Request a credential from the PMIx server/SMS

```
17 Format  
PMIx v4.0 Python  
18 rc,cred = myclient.get_credential(directives:list)  
Python  
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
```

#### **25 A.4.25 Client.validate credential**

## 26           **Summary**

## 27           Request validation of a credential by the PMIx server/SMS

```
1 Format
PMIx v4.0 Python
2 rc,info = myclient.validate_credential(cred:dict, directives:list)
Python
3 IN cred
4 Python byteobject containing credential (dict)
5 IN directives
6 Optional list of Python info describing request (list)
7 Returns:
8 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
9 • info - List of Python info containing additional results of the request (list)
10 See PMIx.Validate_credential for description of all relevant attributes and behaviors
```

## 11 A.4.26 Client.group\_construct

## Summary

13 Construct a new group composed of the specified processes and identified with the provided group  
14 identifier

15                   **Format**

16                    rc,info = myclient.construct\_group(grp:string, members:list, directives:list)

17                    IN grp

18                    Python string identifier for the group (str)

19                    IN members

20                    List of Python proc dictionaries identifying group members (list)

21                    IN directives

22                    Optional list of Python info describing request (list)

23                    Returns:

24                    • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)

25                    • info - List of Python info containing results of the request (list)

26                    See PMIx\_Group\_construct for description of all relevant attributes and behaviors

## **27 A.4.27 Client.group\_invite**

## Summary

Explicitly invite specified processes to join a group

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myclient.group_invite(grp:string, members:list, directives:list)  
4      Python  
5      IN  grp  
6          Python string identifier for the group (str)  
7      IN  members  
8          List of Python proc dictionaries identifying processes to be invited (list)  
9      IN  directives  
10         Optional list of Python info describing request (list)  
11  
12     Returns:  
13         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
14         • info - List of Python info containing results of the request (list)  
15  
16     See PMIx_Group_invite for description of all relevant attributes and behaviors
```

## 13 A.4.28 Client.group\_join

```
14    Summary  
15    Respond to an invitation to join a group that is being asynchronously constructed
```

```
16    Format  
17    PMIx v4.0          Python  
18    rc,info = myclient.group_join(grp:string, leader:dict, opt:integer, directives:list)  
19    Python  
20    IN  grp  
21        Python string identifier for the group (str)  
22    IN  leader  
23        Python proc dictionary identifying process leading the group (dict)  
24    IN  opt  
25        One of the pmix_group_opt_t values indicating decline/accept (integer)  
26    IN  directives  
27        Optional list of Python info describing request (list)  
28  
29    Returns:  
30        • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
31        • info - List of Python info containing results of the request (list)  
32  
33    See PMIx_Group_join for description of all relevant attributes and behaviors
```

## 1 A.4.29 Client.group\_leave

### 2 Summary

3 Leave a PMIx Group

### 4 Format

5 *PMIx v4.0*

Python

```
6   rc = myclient.group_leave(grp:string, directives:list)
```

Python

#### IN grp

Python string identifier for the group (str)

#### IN directives

Optional list of Python `info` describing request (list)

10 Returns:

- 11 • *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

12 See [PMIx\\_Group\\_leave](#) for description of all relevant attributes and behaviors

## 13 A.4.30 Client.group\_destruct

### 14 Summary

15 Destruct a PMIx Group

### 16 Format

17 *PMIx v4.0*

Python

```
18   rc = myclient.group_destruct(grp:string, directives:list)
```

Python

#### IN grp

Python string identifier for the group (str)

#### IN directives

Optional list of Python `info` describing request (list)

22 Returns:

- 23 • *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

24 See [PMIx\\_Group\\_destruct](#) for description of all relevant attributes and behaviors

## 25 A.4.31 Client.register\_event\_handler

### 26 Summary

27 Register an event handler to report events.

```
1 Format
PMIx v4.0 Python
2 rc,id = myclient.register_event_handler(codes:list, directives:list, cbfunc)
Python
3 IN codes
4 List of Python integer status codes that should be reported to this handler (list)
5 IN directives
6 Optional list of Python info describing request (list)
7 IN cbfunc
8 Python evhandler to be called when event is received (func)

9 Returns:
10 • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
11 • id - PMIx reference identifier for handler (integer)
12 See PMIx_Register_event_handler for description of all relevant attributes and behaviors
```

### **13 A.4.32 Client.deregister event handler**

## 14           **Summary**

## 15           Deregister an event handler

```
16 Format
PMIx v4.0 Python
17 myclient.deregister_event_handler(id:integer)
Python
18 IN id
19 PMIx reference identifier for handler (integer)
20 Returns: None
21 See PMIx\_Deregister\_event\_handler for description of all relevant attributes and
22 behaviors
```

## 23 A.4.33 Client.notify event

**Summary**  
Report an event for notification via any registered handler.

1 Format  
2 `rc = myclient.notify_event(status:integer, source:dict,`  
3 `range:integer, directives:list)`

4 IN **status**  
5 PMIx status code indicating the event being reported (integer)  
6 IN **source**  
7 Python **proc** of the process that generated the event (dict)  
8 IN **range**  
9 Python **range** in which the event is to be reported (integer)  
10 IN **directives**  
11 Optional list of Python **info** dictionaries describing the event (list)

12 Returns:

13 • `rc` - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

14 See **PMIx\_Notify\_event** for description of all relevant attributes and behaviors

## 15 A.4.34 Client.fabric\_register

**Summary**  
Register for access to fabric-related information, including communication cost matrix.

```
18      Format  
19      PMIx v4.0  Python  
20      rc,fabricinfo = myserver.fabric_register(directives:list)  
21          Python  
22      IN  directives  
23          Optional list of Python info containing directives (list)  
24      Returns:  
25          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
26          • fabricinfo - List of Python info containing fabric info (list)  
27      See PMIx Fabric register for details
```

## **26 A.4.35 Client.fabric update**

**Summary**  
Update fabric-related information, including communication cost matrix.

```
1 Format  
2 PMIx v4.0  
3 rc, fabricinfo = myserver.fabric_update()  
4 Returns:  
5 • rc - PMIx_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
6 • fabricinfo - List of Python info containing updated fabric info (list)  
7 See PMIx Fabric update for details
```

## **7 A.4.36 Client.fabric\_deregister**

```
8      Summary
9      Deregister fabric
10     Format
11      PMIx v4.0
12      rc = myserver.fabric_deregister()
13      Returns:
14      • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
15      See PMIx Fabric deregister for details
```

### 15 A.4.37 Client.fabric\_get\_vertex\_info

**Summary**  
Given a communication cost matrix index for a specified fabric, return an array of information describing the corresponding NIC.

19       **Format**  
20        rc, nicinfo = myserver.fabric\_get\_vertex\_info(index:integer)  
21        Returns:  
22        • rc - **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
23        • nicinfo - List of Python **info** describing the referenced NIC (list)  
24        See **PMIx Fabric get vertex info** for details

1 **A.4.38 Client.fabric\_get\_device\_index**

2      **Summary**

3        Given info describing a given vertex, return the corresponding communication cost matrix index

4      **Format**

5     *PMIx v4.0*

Python

6     *rc, index = myserver.fabric\_get\_device\_index(info:list)*

Python

7      **IN    info**

8        List of Python `info` containing vertex description (list)

9      Returns:

- 10
  - *rc* - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
  - *index* - Index of corresponding NIC (integer)

11     See `PMIx_Fabric_get_device_index` for details

12 **A.4.39 Client.error\_string**

13      **Summary**

14        Pretty-print string representation of `pmix_status_t`.

15      **Format**

16     *PMIx v4.0*

Python

17     *rep = myclient.error\_string(status:integer)*

Python

18      **IN    status**

19        PMIx status code (integer)

20      Returns:

- 21
  - *rep* - String representation of the provided status code (str)

22     See `PMIx_Error_string` for further details

23 **A.4.40 Client.proc\_state\_string**

24      **Summary**

25        Pretty-print string representation of `pmix_proc_state_t`.

```
1      Format  
2      PMIx v4.0   Python  
3      rep = myclient.proc_state_string(state:integer)  
4      Python  
5      IN state  
6      PMIx process state code (integer)
```

5 Returns:  
6 • *rep* - String representation of the provided process state (str)  
7 See [PMIx\\_Proc\\_state\\_string](#) for further details

## 8 A.4.41 Client.scope\_string

```
9      Summary  
10     Pretty-print string representation of pmix\_scope\_t .
```

```
11     Format  
12     PMIx v4.0   Python  
13     rep = myclient.scope_string(scope:integer)  
14     Python  
15     IN scope  
16     PMIx scope value (integer)
```

15 Returns:  
16 • *rep* - String representation of the provided scope (str)  
17 See [PMIx\\_Scope\\_string](#) for further details

## 18 A.4.42 Client.persistence\_string

```
19      Summary  
20     Pretty-print string representation of pmix\_persistence\_t .
```

```
21      Format  
22     PMIx v4.0   Python  
23     rep = myclient.persistence_string(persistence:integer)  
24     Python  
25     IN persistence  
26     PMIx persistence value (integer)
```

25 Returns:  
26 • *rep* - String representation of the provided persistence (str)  
27 See [PMIx\\_Persistence\\_string](#) for further details

1 **A.4.43 Client.data\_range\_string**

2      **Summary**

3      Pretty-print string representation of [pmix\\_data\\_range\\_t](#) .

4      **Format**

PMIx v4.0

Python

5      `rep = myclient.data_range_string(range:integer)`

Python

6      **IN range**

7      PMIx data range value (integer)

8      Returns:

- 9      • *rep* - String representation of the provided data range (str)

10     See [PMIx\\_Data\\_range\\_string](#) for further details

11 **A.4.44 Client.info\_directives\_string**

12      **Summary**

13      Pretty-print string representation of [pmix\\_info\\_directives\\_t](#) .

14      **Format**

PMIx v4.0

Python

15      `rep = myclient.info_directives_string(directives:integer)`

Python

16      **IN directives**

17      PMIx info directives value (integer)

18      Returns:

- 19      • *rep* - String representation of the provided info directives (str)

20     See [PMIx\\_Info\\_directives\\_string](#) for further details

21 **A.4.45 Client.data\_type\_string**

22      **Summary**

23      Pretty-print string representation of [pmix\\_data\\_type\\_t](#) .

```
1 Format  
2 PMIx v4.0 Python  
3 rep = myclient.data_type_string(dtype:integer)  
4 IN dtype  
5 PMIx datatype value (integer)
```

5            Returns:  
6            • *rep* - String representation of the provided datatype (str)  
7            See [PMIx\\_Data\\_type\\_string](#) for further details

## **8 A.4.46 Client.alloc directive string**

## Summary

Pretty-print string representation of `pmix_alloc_directive_t`.

## Format

```
rep = myclient.alloc_directive_string(adir:integer)
```

**IN adir**  
PMIx allocation directive value (integer)

Returns:

- *rep* - String representation of the provided allocation directive (str)

#### **8 A.4.47 Client.iof channel string**

## Summary

Pretty-print string representation of `pmix_iof_channel_t`.

## Format

```
rep = myclient.iof_channel_string(channel:integer)
```

**IN channel**  
PMIx IOF channel value (integer)

Returns:

- *rep* - String representation of the provided IOF channel (str)

See [DMIOF\\_IOC\\_channel\\_string](#) for further details.

1 **A.4.48 Client.job\_state\_string**

2 **Summary**

3 Pretty-print string representation of [pmix\\_job\\_state\\_t](#).

4 **Format**

5 *PMIx v4.0*

Python

6 `rep = myclient.job_state_string(state:integer)`

Python

7 **IN state**

8 PMIx job state value (integer)

9 Returns:

- 10 • *rep* - String representation of the provided job state (str)

See [PMIx\\_Job\\_state\\_string](#) for further details

11 **A.4.49 Client.get\_attribute\_string**

12 **Summary**

13 Pretty-print string representation of a PMIx attribute.

14 **Format**

15 *PMIx v4.0*

Python

16 `rep = myclient.get_attribute_string(attribute:str)`

Python

17 **IN attribute**

18 PMIx attribute name (string)

19 Returns:

- 20 • *rep* - String representation of the provided attribute (str)

See [PMIx\\_Get\\_attribute\\_string](#) for further details

21 **A.4.50 Client.get\_attribute\_name**

22 **Summary**

23 Pretty-print name of a PMIx attribute corresponding to the provided string

```
1      Format  
2      PMIx v4.0   Python  
3      rep = myclient.get_attribute_name(attribute:str)  
4      Python  
5      IN  attributestring  
6          Attribute string (string)  
7      Returns:  
8          • rep - Attribute name corresponding to the provided string (str)  
9      See PMIx\_Get\_attribute\_name for further details
```

## 8 A.4.51 Client.link\_state\_string

```
9      Summary  
10     Pretty-print string representation of pmix\_link\_state\_t.  
11      Format  
12      PMIx v4.0   Python  
13      rep = myclient.link_state_string(state:integer)  
14      Python  
15      IN  state  
16          PMIx link state value (integer)  
17      Returns:  
18          • rep - String representation of the provided link state (str)  
19      See PMIx\_Link\_state\_string for further details
```

# 18 A.5 PMIxServer

19 The server Python class inherits the Python "client" class as its parent. Thus, it includes all client  
20 functions in addition to the ones defined in this section.

## 21 A.5.1 Server.init

```
22      Summary  
23      Initialize the PMIx server library after obtaining a new PMIxServer object
```

```
1 Format
2 rc = myserver.init(directives:list, map:dict)
3 IN directives
4 List of Python info dictionaries (list)
5 IN map
6 Python dictionary key-function pairs that map server module callback functions to
7 provided implementations (dict)
8 Returns:
9 • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
10 See PMIx\_server\_init for description of all relevant attributes and behaviors
```

## 11 A.5.2 Server.finalize

## Summary

## Finalize the PMIx server library

```
14      Format  
15          rc = myserver.finalize()  
16      Returns:  
17          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
18          See PMIx server finalize for details
```

### 19 A.5.3 Server.generate\_regex

## Summary

Generate a regular expression representation of the input strings.

```
1      Format  
2      PMIx v4.0          Python  
3      rc, regex = myserver.generate_regex(input:list)  
4      Python  
5      IN   input  
6      List of Python strings (e.g., node names) (list)  
7      Returns:  
8      • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9      • regex - Python bytearray containing regular expression representation of the input list  
10     See PMIx_generate_regex for details
```

## 10 A.5.4 Server.generate\_ppn

```
11     Summary  
12     Generate a regular expression representation of the input strings.
```

```
13     Format  
14     PMIx v4.0          Python  
15     rc, regex = myserver.generate_ppn(input:list)  
16     Python  
17     IN   input  
18     List of Python strings, each string consisting of a comma-delimited list of ranks on each node,  
19     with the strings being in the same order as the node names provided to "generate_regex" (list)  
20     Returns:  
21     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22     • regex - Python bytearray containing regular expression representation of the input list  
23     See PMIx_generate_ppn for details
```

## 23 A.5.5 Server.register\_nspace

```
24     Summary  
25     Setup the data about a particular namespace.
```

```
1      Format  
2      PMIx v4.0   Python  
3      rc = myserver.register_nspace(nspace:str,  
4                                       nlocalprocs:integer,  
5                                       directives:list)  
6      Python  
7      IN  nspace  
8          Python string containing the namespace (str)  
9      IN  nlocalprocs  
10         Number of local processes (integer)  
11      IN  directives  
12         List of Python info dictionaries (list)  
13  
14     Returns:  
15       • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
16  
17     See PMIx\_server\_register\_nspace for description of all relevant attributes and behaviors
```

## 14 A.5.6 Server.deregister\_nspace

```
15    Summary  
16    Deregister a namespace.  
17    Format  
18    PMIx v4.0   Python  
19    myserver.deregister_nspace(nspace:str)  
20    Python  
21    IN  nspace  
22        Python string containing the namespace (str)  
23    Returns: None  
24  
25    See PMIx\_server\_deregister\_nspace for details
```

## 23 A.5.7 Server.register\_client

```
24    Summary  
25    Register a client process with the PMIx server library.
```

1           **Format**

2       `rc = myserver.register_client(proc:dict, uid:integer, gid:integer)`

3       **IN proc**

4           Python `proc` dictionary identifying the client process (dict)

5       **IN uid**

6           Linux uid value for user executing client process (integer)

7       **IN gid**

8           Linux gid value for user executing client process (integer)

9           Returns:

- 10
  - `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

11           See `PMIx_server_register_client` for details

12      **A.5.8 Server.deregister\_client**

13           **Summary**

14           Deregister a client process and purge all data relating to it

15           **Format**

16       `myserver.deregister_client(proc:dict)`

17       **IN proc**

18           Python `proc` dictionary identifying the client process (dict)

19           Returns: None

20           See `PMIx_server_deregister_client` for details

21      **A.5.9 Server.setup\_fork**

22           **Summary**

23           Setup the environment of a child process that is to be forked by the host

```
1      Format  
2      PMIx v4.0          Python  
3      rc = myserver.setup_fork(proc:dict, envin:dict)  
4      Python  
5      IN proc  
6          Python proc dictionary identifying the client process (dict)  
7      INOUT envin  
8          Python dictionary containing the environment to be passed to the client (dict)  
9      Returns:  
10     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
11     See PMIx\_server\_setup\_fork for details
```

## 10 A.5.10 Server.dmodex\_request

```
11    Summary  
12    Function by which the host server can request modex data from the local PMIx server.  
13    Format  
14    PMIx v4.0          Python  
15    rc,data = myserver.dmodex_request(proc:dict)  
16    Python  
17    IN proc  
18        Python proc dictionary identifying the process whose data is requested (dict)  
19    Returns:  
20    • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
21    • data - Python byteobject containing the returned data (dict)  
22    See PMIx\_server\_dmodex\_request for details
```

## 21 A.5.11 Server.setup\_application

```
22    Summary  
23    Function by which the resource manager can request application-specific setup data prior to launch  
24    of a job .
```

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myserver.setup_application(nspace:str, directives:list)  
4      Python  
5      IN  nspace  
6      Namespace whose setup information is being requested (str)  
7      IN  directives  
8      Python list of info directives  
9      Returns:  
10     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
11     • info - Python list of info dictionaries containing the returned data (list)  
12     See PMIx_server_setup_application for details
```

## A.5.12 Server.register\_attributes

```
12    Summary  
13    Register host environment attribute support for a function.  
14    Format  
15    PMIx v4.0          Python  
16    rc = myserver.register_attributes(function:str, attrs:list)  
17    Python  
18    IN  function  
19    Name of the function (str)  
20    IN  attrs  
21    Python list of regattr describing the supported attributes  
22    Returns:  
23    • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
24    See PMIx_Register_attributes for details
```

## A.5.13 Server.setup\_local\_support

```
24    Summary  
25    Function by which the local PMIx server can perform any application-specific operations prior to  
26    spawning local clients of a given application
```

1 Format  
2 `rc = myserver.setup_local_support(nspace:str, info:list)`  
3 IN **nspace**  
4 Namespace whose setup information is being requested (str)  
5 IN **info**  
6 Python list of **info** containing the setup data (list)  
7 Returns:  
8 • `rc` - **PMIx\_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)  
9 See **PMIx\_server\_setup\_local\_support** for details

#### **10 A.5.14 Server.iof deliver**

## Summary

Function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

```
14 Format
PMIx v4.0 Python
15 rc = myserver.iof_deliver(source:dict, channel:integer,
                           data:dict, directives:list)
Python
16
17 IN source
18     Python proc dictionary identifying the process who generated the data (dict)
19 IN channel
20     Python channel bitmask identifying IO channel of the provided data (integer)
21 IN data
22     Python byteobject containing the data (dict)
23 IN directives
24     Python list of info containing directives (list)
25 Returns:
26     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)
27 See PMIx_server_IOF_deliver for details
```

## 28 A.5.15 Server.collect inventory

## Summary

Collect inventory of resources on a node

```
1      Format  
2      PMIx v4.0          Python  
3      rc,info = myserver.collect_inventory(directives:list)  
4      Python  
5      IN  directives  
6      Optional Python list of info containing directives (list)  
7      Returns:  
8      • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9      • info - Python list of info containing the returned data (list)  
10     See PMIx\_server\_collect\_inventory for details
```

## 9 A.5.16 Server.deliver\_inventory

```
10     Summary  
11     Pass collected inventory to the PMIx server library for storage  
12     Format  
13     PMIx v4.0          Python  
14     rc = myserver.deliver_inventory(info:list, directives:list)  
15     Python  
16     IN  info  
17     - Python list of info dictionaries containing the inventory data (list)  
18     IN  directives  
19     Python list of info dictionaries containing directives (list)  
20     Returns:  
21     • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
22     See PMIx\_server\_deliver\_inventory for details
```

## 21 A.6 PMIxTool

```
22     The tool Python class inherits the Python "server" class as its parent. Thus, it includes all client and  
23     server functions in addition to the ones defined in this section.
```

### 24 A.6.1 Tool.init

```
25     Summary  
26     Initialize the PMIx tool library after obtaining a new PMIxTool object
```

```
1      Format  
2      PMIx v4.0   Python  
3      rc,proc = mytool.init(info:list)  
4      Python  
5      IN  info  
6          List of Python info directives (list)  
7      Returns:  
8          • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9          • proc - a Python proc (dict)  
10         See PMIx\_tool\_init for description of all relevant attributes and behaviors
```

## 9 A.6.2 Tool.finalize

```
10     Summary  
11     Finalize the PMIx tool library, closing the connection to the server.  
12     Format  
13     PMIx v4.0   Python  
14     rc = mytool.finalize()  
15     Python  
16     Returns:  
17         • rc - PMIX\_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
18         See PMIx\_tool\_finalize for description of all relevant attributes and behaviors
```

## 17 A.6.3 Tool.connect\_to\_server

```
18     Summary  
19     Switch connection from the current PMIx server to another one, or initialize a connection to a  
20     specified server.
```

1           **Format**

2       `rc, proc = mytool.connect_to_server(info:list)`

3           **IN info**

4           List of Python `info` dictionaries (list)

5           Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `proc` - a Python `proc` (dict)

8           See `PMIx_tool_connect_to_server` for description of all relevant attributes and behaviors

9   **A.6.4 Tool.iof\_pull**

10          **Summary**

11          Register to receive output forwarded from a remote process.

12          **Format**

13       `rc,id = mytool.iof_pull(sources:list, channel:integer, directives:list, cbfunc:func)`

14           **IN sources**

15           List of Python `proc` dictionaries of processes whose IO is being requested (list)

16           **IN channel**

17           Python `channel` bitmask identifying IO channels to be forwarded (integer)

18           **IN directives**

19           List of Python `info` dictionaries describing request (list)

20           **IN cbfunc**

21           Python `iofcbfunc` to receive IO payloads (func)

22           Returns:

- `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)
- `id` - PMIx reference identifier for request (integer)

25           See `PMIx_IOF_pull` for description of all relevant attributes and behaviors

26   **A.6.5 Tool.iof\_deregister**

27          **Summary**

28          Deregister from output forwarded from a remote process.

```
1      Format  
2      rc = mytool.iof_deregister(id:integer, directives:list)  
3      IN  id  
4          PMIx reference identifier returned by pull request (list)  
5      IN  directives  
6          List of Python info dictionaries describing request (list)  
7      Returns:  
8          • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
9          See PMIx_IOF_deregister for description of all relevant attributes and behaviors
```

## 10 A.6.6 Tool.iof\_push

```
11     Summary  
12     Push data collected locally (typically from stdin) to stdin of target recipients
```

```
13     Format  
14     rc = mytool.iof_push(targets:list, data:dict, directives:list)  
15     IN  sources  
16         List of Python proc of target processes (list)  
17     IN  data  
18         Python byteobject containing data to be delivered (dict)  
19     IN  directives  
20         Optional list of Python info describing request (list)  
21     Returns:  
22         • rc - PMIX_SUCCESS or a negative value corresponding to a PMIx error constant (integer)  
23         See PMIx_IOF_push for description of all relevant attributes and behaviors
```

## 24 A.7 Example Usage

```
25     The following examples are provided to illustrate the use of the Python bindings.
```

## 1 A.7.1 Python Client

2 The following example contains a client program that illustrates a fairly common usage pattern.  
3 The program instantiates and initializes the PMIxClient class, posts some data that is to be shared  
4 across all processes in the job, executes a “fence” that circulates the data, and then retrieves a value  
5 posted by one of its peers. Note that the example has been formatted to fit the document layout.

### Python

```
6 from pmix import *
7
8 def main():
9     # Instantiate a client object
10    myclient = PMIxClient()
11    print("Testing PMIx ", myclient.get_version())
12
13    # Initialize the PMIx client library, declaring the programming model
14    # as "TEST" and the library name as "PMIX", just for the example
15    info = [ {'key':PMIX_PROGRAMMING_MODEL,
16              'value':'TEST', 'val_type':PMIX_STRING,
17              'key':PMIX_MODEL_LIBRARY_NAME,
18              'value':'PMIX', 'val_type':PMIX_STRING}]
19    rc,myname = myclient.init(info)
20    if PMIX_SUCCESS != rc:
21        print("FAILED TO INIT WITH ERROR", myclient.error_string(rc))
22        exit(1)
23
24    # try posting a value
25    rc = myclient.put(PMIX_GLOBAL, "mykey",
26                      'value':1, 'val_type':PMIX_INT32)
27    if PMIX_SUCCESS != rc:
28        print("PMIx_Put FAILED WITH ERROR", myclient.error_string(rc))
29        # cleanly finalize
30        myclient.finalize()
31        exit(1)
32
33    # commit it
34    rc = myclient.commit()
35    if PMIX_SUCCESS != rc:
36        print("PMIx_Coмmit FAILED WITH ERROR",
37              myclient.error_string(rc))
38    # cleanly finalize
39    myclient.finalize()
40    exit(1)
41
```

```

1      # execute fence across all processes in my job
2      procs = []
3      info = []
4      rc = myclient.fence(procs, info)
5      if PMIX_SUCCESS != rc:
6          print("PMIx_Fence FAILED WITH ERROR", myclient.error_string(rc))
7          # cleanly finalize
8          myclient.finalize()
9          exit(1)
10
11      # Get a value from a peer
12      if 0 != myname['rank']:
13          info = []
14          rc, get_val = myclient.get('nspacename': "testnspace", 'rank': 0,
15                                      "mykey", info)
16          if PMIX_SUCCESS != rc:
17              print("PMIx_Commit FAILED WITH ERROR",
18                  myclient.error_string(rc))
19              # cleanly finalize
20              myclient.finalize()
21              exit(1)
22          print("Get value returned: ", get_val)
23
24      # test a fence that should return not_supported because
25      # we pass a required attribute that the server is known
26      # not to support
27      procs = []
28      info = ['key': 'ARBIT', 'flags': PMIX_INFO_REQD,
29              'value': 10, 'val_type': PMIX_INT]
30      rc = myclient.fence(procs, info)
31      if PMIX_SUCCESS == rc:
32          print("PMIx_Fence SUCCEEDED BUT SHOULD HAVE FAILED")
33          # cleanly finalize
34          myclient.finalize()
35          exit(1)
36
37      # Publish something
38      info = ['key': 'ARBITRARY', 'value': 10, 'val_type': PMIX_INT]
39      rc = myclient.publish(info)
40      if PMIX_SUCCESS != rc:
41          print("PMIx_Publish FAILED WITH ERROR",
42                  myclient.error_string(rc))
43      # cleanly finalize

```

```

1     myclient.finalize()
2     exit(1)
3
4     # finalize
5     info = []
6     myclient.finalize(info)
7     print("Client finalize complete")
8
9 # Python main program entry point
10 if __name__ == '__main__':
11     main()

```

Python

## 12 A.7.2 Python Server

13 The following example contains a minimum-level server host program that instantiates and  
14 initializes the PMIxServer class. The program illustrates passing several server module functions to  
15 the bindings and includes code to setup and spawn a simple client application, waiting until the  
16 spawned client terminates before finalizing and exiting itself. Note that the example has been  
17 reformatted to fit the document layout.

Python

```

18 from pmix import *
19 import signal, time
20 import os
21 import select
22 import subprocess
23
24 def clientconnected(proc:tuple is not None):
25     print("CLIENT CONNECTED", proc)
26     return PMIX_OPERATION_SUCCEEDED
27
28 def clientfinalized(proc:tuple is not None):
29     print("CLIENT FINALIZED", proc)
30     return PMIX_OPERATION_SUCCEEDED
31
32 def clientfence(procs:list, directives:list, data:bytearray):
33     # check directives
34     if directives is not None:
35         for d in directives:
36             # these are each an info dict
37             if "pmix" not in d['key']:
38                 # we do not support such directives - see if

```

```

1          # it is required
2      try:
3          if d['flags'] & PMIX_INFO_REQD:
4              # return an error
5              return PMIX_ERR_NOT_SUPPORTED
6      except:
7          #it can be ignored
8          pass
9      return PMIX_OPERATION_SUCCEEDED
10
11 def main():
12     try:
13         myserver = PMIxServer()
14     except:
15         print("FAILED TO CREATE SERVER")
16         exit(1)
17     print("Testing server version ", myserver.get_version())
18
19     args = ['key':PMIX_SERVER_SCHEDULER,
20             'value':'T', 'val_type':PMIX_BOOL]
21     map = 'clientconnected': clientconnected,
22           'clientfinalized': clientfinalized,
23           'fencenb': clientfence
24     my_result = myserver.init(args, map)
25
26     # get our environment as a base
27     env = os.environ.copy()
28
29     # register an nspace for the client app
30     (rc, regex) = myserver.generate_regex("test000,test001,test002")
31     (rc, ppn) = myserver.generate_ppn("0")
32     kvals = ['key':PMIX_NODE_MAP,
33               'value':regex, 'val_type':PMIX_STRING,
34               'key':PMIX_PROC_MAP,
35               'value':ppn, 'val_type':PMIX_STRING,
36               'key':PMIX_UNIV_SIZE,
37               'value':1, 'val_type':PMIX_UINT32,
38               'key':PMIX_JOB_SIZE,
39               'value':1, 'val_type':PMIX_UINT32]
40     rc = foo.register_nspace("testnspace", 1, kvals)
41     print("RegNspace ", rc)
42
43     # register a client

```

```

1   uid = os.getuid()
2   gid = os.getgid()
3   rc = myserver.register_client('nspacename' :"testnspace", 'rank':0,
4                                 uid, gid)
5   print("RegClient ", rc)
6   # setup the fork
7   rc = myserver.setup_fork('nspacename' :"testnspace", 'rank':0, env)
8   print("SetupFrk", rc)
9
10  # setup the client argv
11  args = ["../client.py"]
12  # open a subprocess with stdout and stderr
13  # as distinct pipes so we can capture their
14  # output as the process runs
15  p = subprocess.Popen(args, env=env,
16                      stdout=subprocess.PIPE, stderr=subprocess.PIPE)
17  # define storage to catch the output
18  stdout = []
19  stderr = []
20  # loop until the pipes close
21  while True:
22      reads = [p.stdout.fileno(), p.stderr.fileno()]
23      ret = select.select(reads, [], [])
24
25      stdout_done = True
26      stderr_done = True
27
28      for fd in ret[0]:
29          # if the data
30          if fd == p.stdout.fileno():
31              read = p.stdout.readline()
32              if read:
33                  read = read.decode('utf-8').rstrip()
34                  print('stdout: ' + read)
35                  stdout_done = False
36          elif fd == p.stderr.fileno():
37              read = p.stderr.readline()
38              if read:
39                  read = read.decode('utf-8').rstrip()
40                  print('stderr: ' + read)
41                  stderr_done = False
42
43      if stdout_done and stderr_done:

```

```
1         break
2     print("FINALIZING")
3     myserver.finalize()
4
5
6 if __name__ == '__main__':
7     main()
```

Python

## APPENDIX B

# Acknowledgements

---

This document represents the work of many people who have contributed to the PMIx community. Without the hard work and dedication of these people this document would not have been possible. The sections below list some of the active participants and organizations in the various PMIx standard iterations.

## B.1 Version 3.0

The following list includes some of the active participants in the PMIx v3 standardization process.

- Ralph H. Castain, Andrew Friedley, Brandon Yates
- Joshua Hursey
- Aurelien Bouteiller and George Bosilca
- Dirk Schubert
- Kevin Harms

The following institutions supported this effort through time and travel support for the people listed above.

- Intel Corporation
- IBM, Inc.
- University of Tennessee, Knoxville
- The Exascale Computing Project, an initiative of the US Department of Energy
- National Science Foundation
- Argonne National Laboratory
- Allinea (ARM)

## 1    **B.2 Version 2.0**

2        The following list includes some of the active participants in the PMIx v2 standardization process.

- 3           • Ralph H. Castain, Annapurna Dasari, Christopher A. Holguin, Andrew Friedley, Michael Klemm  
4           and Terry Wilmarth
- 5           • Joshua Hursey, David Solt, Alexander Eichenberger, Geoff Paulsen, and Sameh Sharkawi
- 6           • Aurelien Bouteiller and George Bosilca
- 7           • Artem Polyakov, Igor Ivanov and Boris Karasev
- 8           • Gilles Gouaillardet
- 9           • Michael A Raymond and Jim Stoffel
- 10          • Dirk Schubert
- 11          • Moe Jette
- 12          • Takahiro Kawashima and Shinji Sumimoto
- 13          • Howard Pritchard
- 14          • David Beer
- 15          • Brice Goglin
- 16          • Geoffroy Vallee, Swen Boehm, Thomas Naughton and David Bernholdt
- 17          • Adam Moody and Martin Schulz
- 18          • Ryan Grant and Stephen Olivier
- 19          • Michael Karo

20        The following institutions supported this effort through time and travel support for the people listed  
21        above.

- 22          • Intel Corporation
- 23          • IBM, Inc.
- 24          • University of Tennessee, Knoxville
- 25          • The Exascale Computing Project, an initiative of the US Department of Energy
- 26          • National Science Foundation
- 27          • Mellanox, Inc.
- 28          • Research Organization for Information Science and Technology
- 29          • HPE Co.

- 1 • Allinea (ARM)
- 2 • SchedMD, Inc.
- 3 • Fujitsu Limited
- 4 • Los Alamos National Laboratory
- 5 • Adaptive Solutions, Inc.
- 6 • INRIA
- 7 • Oak Ridge National Laboratory
- 8 • Lawrence Livermore National Laboratory
- 9 • Sandia National Laboratory
- 10 • Altair

## 11 **B.3 Version 1.0**

12 The following list includes some of the active participants in the PMIx v1 standardization process.

- 13 • Ralph H. Castain, Annapurna Dasari and Christopher A. Holguin
- 14 • Joshua Hursey and David Solt
- 15 • Aurelien Bouteiller and George Bosilca
- 16 • Artem Polyakov, Elena Shipunova, Igor Ivanov, and Joshua Ladd
- 17 • Gilles Gouaillardet
- 18 • Gary Brown
- 19 • Moe Jette

20 The following institutions supported this effort through time and travel support for the people listed  
21 above.

- 22 • Intel Corporation
- 23 • IBM, Inc.
- 24 • University of Tennessee, Knoxville
- 25 • Mellanox, Inc.
- 26 • Research Organization for Information Science and Technology
- 27 • Adaptive Solutions, Inc.
- 28 • SchedMD, Inc.

# Bibliography

---

- [1] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. PMIx: Process management for exascale environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.
- [2] Balaji P. et al. PMI: A scalable parallel process-management interface for extreme-scale systems. In *Recent Advances in the Message Passing Interface*, EuroMPI '10, pages 31–41, Berlin, Heidelberg, 2010. Springer.

# Index

---

General terms and other items not induced in the other indices.

application, 7, 10, [14](#), 80, 81, 134, 190, 260, 262

fabric, [15](#)

fabric plane, [15](#), 352

fabric planes, [15](#), 341

fabrics, [15](#)

host environment, [15](#)

job, 7, 8, 10, [14](#), 80–82, 134, 136, 190, 253, 259, 260, 262, 272, 274, 476

namespace, [14](#)

rank, [14](#), 136, 263

resource manager, [15](#)

scheduler, [15](#), 350

session, 7, 8, 10, [14](#), 80, 81, 134, 190, 259

slot, [14](#)

slots, [14](#)

workflow, [14](#)

workflows, [14](#), 113

# Index of APIs

---

PMIx\_Abort, 6, 23, 35, [154](#), 155, 286, 287, [436](#), 450  
    PMIxClient.abort (Python), [449](#)

PMIx\_Alloc\_directive\_string, 7, [118](#), 469  
    PMIxClient.alloc\_directive\_string (Python), [469](#)

PMIx\_Allocation\_request, 9–11, 94, 191, [192](#), 198

PMIx\_Allocation\_request\_nb, 7, 94, 179, [195](#), 198, 458  
    PMIxClient.allocate (Python), [457](#)

PMIx\_Commit, 6, 111, 125, 127, [138](#), 138, 271, 292, 451  
    PMIxClient.commit (Python), [451](#)

PMIx\_Connect, 6, 7, 23, 160, [167](#), 169, 171, 173, 362–364, 455  
    PMIxClient.connect (Python), [454](#)

PMIx\_Connect\_nb, 6, [170](#), 170

pmix\_connection\_cbfunc\_t, [111](#), 313

pmix\_credential\_cbfunc\_t, [112](#), 239, 328

PMIx\_Data\_copy, 7, [233](#)

PMIx\_Data\_copy\_payload, 7, [234](#)

PMIx\_Data\_pack, 7, [229](#), 230, 249, 250

PMIx\_Data\_print, 7, [233](#)

PMIx\_Data\_range\_string, 7, [117](#), 468  
    PMIxClient.data\_range\_string (Python), [468](#)

PMIx\_Data\_type\_string, 7, [117](#), 469  
    PMIxClient.data\_type\_string (Python), [468](#)

PMIx\_Data\_unpack, 7, [231](#)

PMIx\_Deregister\_event\_handler, 7, 11, [221](#), 463  
    PMIxClient.deregister\_event\_handler (Python), [463](#)

PMIx\_Disconnect, 6, 7, 23, 169, [171](#), 173, [175](#), 363, 364, 455  
    PMIxClient.disconnect (Python), [455](#)

PMIx\_Disconnect\_nb, 6, [173](#), 175, 363

pmix\_dmodex\_response\_fn\_t, [110](#), 271

PMIx\_Error\_string, 6, [116](#), 466  
    PMIxClient.error\_string (Python), [466](#)

pmix\_event\_notification\_cbfunc\_fn\_t, [106](#), 106, 106, 108

pmix\_evhdlr\_reg\_cbfunc\_t, [106](#), 106, 219

PMIx\_Fabric\_deregister, 12, [354](#), 355, 465  
    PMIxClient.fabric\_deregister (Python), [465](#)

PMIx\_Fabric\_deregister\_nb, 12, [355](#)

PMIx\_Fabric\_get\_device\_index, 12, [358](#), 360, 466  
    PMIxClient.fabric\_get\_device\_index (Python), [466](#)

PMIx\_Fabric\_get\_device\_index\_nb, 12, 359  
PMIx\_Fabric\_get\_vertex\_info, 12, 355, 358, 465  
    PMIxClient.fabric\_get\_vertex\_info (Python), 465  
PMIx\_Fabric\_get\_vertex\_info\_nb, 12, 358  
PMIx\_Fabric\_register, 11, 344, 351, 353, 464  
    PMIxClient.fabric\_register (Python), 464  
PMIx\_Fabric\_register\_nb, 11, 352  
PMIx\_Fabric\_update, 11, 352, 353, 354, 465  
    PMIxClient.fabric\_update (Python), 464  
PMIx\_Fabric\_update\_nb, 11, 353  
PMIx\_Fence, 4, 6, 13, 125, 139, 140, 142, 169, 173, 248, 271, 287, 290, 361, 368, 373, 436, 452  
    PMIxClient.fence (Python), 451  
PMIx\_Fence\_nb, 6, 10, 104, 140, 142, 287, 290, 436  
PMIx\_Finalize, 6, 23, 26, 35, 85, 123, 124, 124, 167, 285, 286, 435, 449  
    PMIxClient.finalize (Python), 449  
PMIx\_generate\_locality\_string, 12, 175, 177, 281  
PMIx\_generate\_ppn, 6, 250, 473  
    PMIxServer.generate\_ppn (Python), 473  
PMIx\_generate\_regex, 6, 249, 250, 259, 473  
    PMIxServer.generate\_regex (Python), 472  
PMIx\_Get, 3, 6–8, 25, 41, 74–78, 82, 84–92, 94, 96, 98, 123, 127, 128, 130, 131, 133–135, 138, 157–159, 162–164, 178, 185, 190, 191, 251, 253, 254, 257, 280, 300–303, 341, 348, 349, 361, 368, 395, 413, 452  
    PMIxClient.get (Python), 452  
PMIx\_Get\_attribute\_name, 119, 471  
    PMIxClient.get\_attribute\_name (Python), 470  
PMIx\_Get\_attribute\_string, 118, 470  
    PMIxClient.get\_attribute\_string (Python), 470  
PMIx\_Get\_credential, 9, 11, 98, 237, 329, 459  
    PMIxClient.get\_credential (Python), 459  
PMIx\_Get\_credential\_nb, 238  
PMIx\_Get\_nb, 6, 17, 104, 105, 130  
PMIx\_Get\_relative\_locality, 11, 176, 178, 257, 281  
PMIx\_Get\_version, 6, 17, 121, 449  
    PMIxClient.get\_version (Python), 449  
PMIx\_Group\_construct, 11, 362, 364, 367, 368, 371, 460  
    PMIxClient.group\_construct (Python), 460  
PMIx\_Group\_construct\_nb, 11, 368, 371  
PMIx\_Group\_destruct, 11, 364, 371, 373, 375, 462  
    PMIxClient.group\_destruct (Python), 462  
PMIx\_Group\_destruct\_nb, 11, 373, 375  
PMIx\_Group\_invite, 11, 363, 375, 378, 379, 381, 461  
    PMIxClient.group\_invite (Python), 460

PMIx\_Group\_invite\_nb, 11, [379](#)  
PMIx\_Group\_join, 11, 64, 363, 378, [381](#), 381, 383, 384, 386, 461  
    PMIxClient.group\_join (Python), [461](#)  
PMIx\_Group\_join\_nb, 11, 381, [384](#), 386  
PMIx\_Group\_leave, 11, 364, [386](#), 387, 388, 462  
    PMIxClient.group\_leave (Python), [462](#)  
PMIx\_Group\_leave\_nb, 11, [387](#)  
pmix\_hdlr\_reg\_cbfnc\_t, 106, [115](#), [421](#), [423](#)  
pmix\_info\_cbfnc\_t, 101, [105](#), 105, 186, 196, 201, 202, 204, 207, 209, 279, 314, 320, 322, 325,  
    336, 338, 358, 360, 369, 379, 385  
PMIx\_Info\_directives\_string, 7, [117](#), [468](#)  
    PMIxClient.info\_directives\_string (Python), [468](#)  
PMIx\_Init, 7, 89, [121](#), [121](#), 123, [124](#), 158, [163](#), 284, 301, 399, 409, 410, 413, 448  
    PMIxClient.init (Python), [448](#)  
PMIx\_Initialized, 6, [120](#), [448](#)  
    PMIxClient.initialized (Python), [448](#)  
pmix\_iof\_cbfnc\_t, [114](#), [421](#), [434](#)  
PMIx\_IOF\_channel\_string, 9, [118](#), [469](#)  
    PMIxClient.iof\_channel\_string (Python), [469](#)  
PMIx\_IOF\_deregister, 9, 11, [423](#), [482](#)  
    PMIxTool.iof\_deregister (Python), [481](#)  
PMIx\_IOF\_pull, 9, 11, 404, [421](#), [423](#), [481](#)  
    PMIxTool.iof\_pull (Python), [481](#)  
PMIx\_IOF\_push, 9, 11, 406, 408, [424](#), [425](#), [482](#)  
    PMIxTool.iof\_push (Python), [482](#)  
PMIx\_Job\_control, 8, 11, 96, [199](#), 201, 204, 324, 411  
PMIx\_Job\_control\_nb, 7, 96, 179, 198, [202](#), 258, 458  
    PMIxClient.job\_ctrl (Python), [458](#)  
PMIx\_Job\_state\_string, [118](#), [470](#)  
    PMIxClient.job\_state\_string (Python), [470](#)  
PMIx\_Link\_state\_string, [119](#), [471](#)  
    PMIxClient.link\_state\_string (Python), [471](#)  
PMIx\_Load\_topology, 11, 83, [175](#)  
PMIx\_Log, 8, 91, 92, [209](#), 212, 398, 457  
    PMIxClient.log (Python), [457](#)  
PMIx\_Log\_nb, 7, [212](#), [215](#)  
PMIx\_Lookup, 6, 54, 143, [146](#), 149, 150, 452, 453  
    PMIxClient.lookup (Python), [453](#)  
pmix\_lookup\_cbfnc\_t, [104](#), [104](#), [295](#)  
PMIx\_Lookup\_nb, 6, [104](#), [149](#)  
pmix\_modex\_cbfnc\_t, 101, [102](#), 102, 288, 291  
pmix\_notification\_fn\_t, [107](#), [107](#), 219, 435  
PMIx\_Notify\_event, 7, 11, [222](#), 312, 464

PMIxClient.notify\_event (Python), 463  
pmix\_op\_cfunc\_t, 103, 103, 107, 110, 145, 152, 170, 174, 212, 222, 223, 251, 267–269, 276, 278,  
    280, 284, 285, 287, 292, 297, 304, 306, 308, 310, 311, 318, 323, 326, 332, 334, 353–355,  
    374, 387, 424  
PMIx\_Persistence\_string, 7, 117, 467  
    PMIxClient.persistence\_string (Python), 467  
PMIx\_Proc\_state\_string, 7, 116, 467  
    PMIxClient.proc\_state\_string (Python), 466  
PMIx\_Process\_monitor, 9, 11, 205, 209  
PMIx\_Process\_monitor\_nb, 7, 98, 179, 207, 209, 459  
    PMIxClient.monitor (Python), 458  
PMIx\_Publish, 6, 39, 85, 143, 144–146, 293, 294, 453  
    PMIxClient.publish (Python), 452  
PMIx\_Publish\_nb, 6, 145, 146  
PMIx\_Put, 6, 38, 39, 41, 111, 125, 126, 127, 130, 138–140, 167, 190, 271, 292, 368, 378, 451  
    PMIxClient.put (Python), 450  
PMIx\_Query\_info, 181, 185, 190, 191, 348, 389, 410  
PMIx\_Query\_info\_nb, 7, 8, 59, 82, 91, 138, 167, 179, 185, 186, 191, 275, 341, 361, 362, 457  
    PMIxClient.query (Python), 456  
PMIx\_Register\_attributes, 10, 12, 274, 477  
    PMIxServer.register\_attributes (Python), 477  
PMIx\_Register\_event\_handler, 7, 11, 90, 106, 179, 218, 398, 463  
    PMIxClient.register\_event\_handler (Python), 462  
pmix\_release\_cfunc\_t, 101, 101  
PMIx\_Resolve\_nodes, 6, 180, 456  
    PMIxClient.resolve\_nodes (Python), 456  
PMIx\_Resolve\_peers, 6, 179, 456  
    PMIxClient.resolve\_peers (Python), 455  
PMIx\_Scope\_string, 7, 117, 467  
    PMIxClient.scope\_string (Python), 467  
pmix\_server\_abort\_fn\_t, 286, 436  
pmix\_server\_alloc\_fn\_t, 319, 443  
pmix\_server\_client\_connected\_fn\_t, 104, 236, 269, 283, 284, 435  
pmix\_server\_client\_finalized\_fn\_t, 285, 286, 435  
PMIx\_server\_collect\_inventory, 9, 278, 479  
    PMIxServer.collect\_inventory (Python), 478  
pmix\_server\_connect\_fn\_t, 167, 303, 305, 307, 439  
PMIx\_server\_deliver\_inventory, 9, 279, 479  
    PMIxServer.deliver\_inventory (Python), 479  
PMIx\_server\_deregister\_client, 6, 269, 475  
    PMIxServer.deregister\_client (Python), 475  
pmix\_server\_deregister\_events\_fn\_t, 309, 441  
PMIx\_server\_deregister\_nspace, 6, 266, 269, 474

PMIxServer.deregister\_nspace (Python), 474  
    pmix\_server\_disconnect\_fn\_t, 305, 307, 440  
    pmix\_server\_dmodex\_req\_fn\_t, 8, 9, 102, 291, 437  
    PMIx\_server\_dmodex\_request, 6, 110, 111, 270, 271, 476  
        PMIxServer.dmodex\_request (Python), 476  
        pmix\_server\_fabric\_fn\_t, 12, 337, 344, 350, 447  
        pmix\_server\_fencenb\_fn\_t, 10, 102, 287, 290, 437  
    PMIx\_server\_finalize, 6, 248, 472  
        PMIxServer.finalize (Python), 472  
    pmix\_server\_get\_cred\_fn\_t, 328, 331, 445  
    pmix\_server\_grp\_fn\_t, 12, 335, 447  
    PMIx\_server\_init, 6, 121, 245, 275, 282, 348, 390, 391, 395, 472  
        PMIxServer.init (Python), 471  
    PMIx\_server\_IOF\_deliver, 9, 277, 404, 478  
        PMIxServer.iof\_deliver (Python), 478  
    pmix\_server\_iof\_fn\_t, 331, 446  
    pmix\_server\_job\_control\_fn\_t, 322, 444  
    pmix\_server\_listener\_fn\_t, 312  
    pmix\_server\_log\_fn\_t, 317, 443  
    pmix\_server\_lookup\_fn\_t, 294, 438  
    pmix\_server\_module\_t, 246, 248, 275, 276, 282, 282  
    pmix\_server\_monitor\_fn\_t, 325, 444  
    pmix\_server\_notify\_event\_fn\_t, 109, 224, 311, 312, 441  
    pmix\_server\_publish\_fn\_t, 292, 437  
    pmix\_server\_query\_fn\_t, 313, 442  
    PMIx\_server\_register\_client, 6, 236, 268, 269, 284, 286, 475  
        PMIxServer.register\_client (Python), 474  
    pmix\_server\_register\_events\_fn\_t, 307, 440  
    PMIx\_server\_register\_nspace, 6, 8, 17, 81, 104, 130, 249, 250, 259, 262, 281, 474  
        PMIxServer.register\_nspace (Python), 473  
    PMIx\_server\_setup\_application, 7, 10, 109, 110, 272, 277, 280, 477  
        PMIxServer.setup\_application (Python), 476  
    PMIx\_server\_setup\_fork, 6, 270, 476  
        PMIxServer.setup\_fork (Python), 475  
    PMIx\_server\_setup\_local\_support, 7, 276, 478  
        PMIxServer.setup\_local\_support (Python), 477  
    pmix\_server\_spawn\_fn\_t, 103, 298, 399, 439  
    pmix\_server\_stdin\_fn\_t, 334, 446  
    pmix\_server\_tool\_connection\_fn\_t, 236, 316, 390, 442  
    pmix\_server\_unpublish\_fn\_t, 296, 438  
    pmix\_server\_validate\_cred\_fn\_t, 329, 445  
    pmix\_setup\_application\_cbfunc\_t, 109, 272  
    PMIx\_Spawn, 6, 10, 57, 78, 88, 94, 155, 155, 156, 161, 162, 166, 195, 257, 258, 270, 298, 302,

303, 321, 396, 399, 400, 402, 404, 408–410, 412–414, 439, 454  
PMIxClient.spawn (Python), 454  
pmix\_spawn\_cbfunc\_t, 103, 103, 161, 299, 402  
PMIx\_Spawn\_nb, 6, 57, 103, 161  
PMIx\_Store\_internal, 6, 133, 450  
    PMIxClient.store\_internal (Python), 450  
PMIx\_tool\_connect\_to\_server, 9, 392, 395, 417, 418, 481  
    PMIxTool.connect\_to\_server (Python), 480  
pmix\_tool\_connection\_cbfunc\_t, 111, 316  
PMIx\_tool\_disconnect, 12, 418  
PMIx\_tool\_finalize, 7, 417, 480  
    PMIxTool.finalize (Python), 480  
PMIx\_tool\_init, 7, 121, 389, 392, 394–396, 404, 414, 417, 480  
    PMIxTool.init (Python), 479  
PMIx\_tool\_set\_server, 12, 391, 420  
PMIx\_Unpublish, 6, 150, 152, 153, 454  
    PMIxClient.unpublish (Python), 453  
PMIx\_Unpublish\_nb, 6, 152  
PMIx\_Validate\_credential, 9, 11, 240, 460  
    PMIxClient.validate\_credential (Python), 459  
PMIx\_Validate\_credential\_nb, 242  
pmix\_validation\_cbfunc\_t, 113, 243, 330  
pmix\_value\_cbfunc\_t, 17, 104, 104

# Index of Support Macros

---

PMIX\_APP\_CONSTRUCT, [58](#)  
PMIX\_APP\_CREATE, [58](#)  
PMIX\_APP\_DESTRUCT, [58](#)  
PMIX\_APP\_FREE, [59](#)  
PMIX\_APP\_INFO\_CREATE, [8](#), [9](#), [59](#)  
PMIX\_APP\_RELEASE, [58](#)  
PMIX\_ARGV\_APPEND, [68](#)  
PMIX\_ARGV\_APPEND\_UNIQUE, [69](#)  
PMIX\_ARGV\_COPY, [72](#)  
PMIX\_ARGV\_COUNT, [71](#)  
PMIX\_ARGV\_FREE, [70](#)  
PMIX\_ARGV\_JOIN, [71](#)  
PMIX\_ARGV\_PREPEND, [69](#)  
PMIX\_ARGV\_SPLIT, [70](#)  
PMIX\_BYTE\_OBJECT\_CONSTRUCT, [65](#)  
PMIX\_BYTE\_OBJECT\_CREATE, [65](#)  
PMIX\_BYTE\_OBJECT\_DESTRUCT, [65](#)  
PMIX\_BYTE\_OBJECT\_FREE, [66](#)  
PMIX\_BYTE\_OBJECT\_LOAD, [66](#)  
PMIX\_CHECK\_KEY, [28](#)  
PMIX\_CHECK\_NSPACE, [29](#)  
PMIX\_CHECK\_PROCID, [33](#)  
PMIX\_COORD\_CONSTRUCT, [342](#)  
PMIX\_COORD\_CREATE, [342](#)  
PMIX\_COORD\_DESTRUCT, [342](#)  
PMIX\_COORD\_FREE, [342](#)  
PMIX\_CPUSET\_CONSTRUCT, [282](#)  
PMIX\_DATA\_ARRAY\_CONSTRUCT, [40](#), [67](#)  
PMIX\_DATA\_ARRAY\_CREATE, [41](#), [67](#)  
PMIX\_DATA\_ARRAY\_DESTRUCT, [40](#), [67](#)  
PMIX\_DATA\_ARRAY\_FREE, [41](#), [68](#)  
PMIX\_DATA\_BUFFER\_CONSTRUCT, [227](#), [230](#), [232](#)  
PMIX\_DATA\_BUFFER\_CREATE, [227](#), [230](#), [232](#)  
PMIX\_DATA\_BUFFER\_DESTRUCT, [228](#)  
PMIX\_DATA\_BUFFER\_LOAD, [228](#)  
PMIX\_DATA\_BUFFER\_RELEASE, [227](#)  
PMIX\_DATA\_BUFFER\_UNLOAD, [229](#), [249](#), [250](#)  
PMIX\_ENVAR\_CONSTRUCT, [53](#)

PMIX\_ENVAR\_CREATE, [53](#)  
PMIX\_ENVAR\_DESTRUCT, [53](#)  
PMIX\_ENVAR\_FREE, [54](#)  
PMIX\_ENVAR\_LOAD, [54](#)  
PMIX\_FABRIC\_CONSTRUCT, [347](#)  
PMIx\_Heartbeat, [7](#), [209](#)  
PMIX\_INFO\_CONSTRUCT, [47](#)  
PMIX\_INFO\_CREATE, [47](#), [50](#), [51](#)  
PMIX\_INFO\_DESTRUCT, [47](#)  
PMIX\_INFO\_FREE, [47](#)  
PMIX\_INFO\_IS\_END, [8](#), [10](#), [51](#)  
PMIX\_INFO\_IS\_OPTIONAL, [51](#)  
PMIX\_INFO\_IS\_REQUIRED, [49](#), [50](#), [51](#)  
PMIX\_INFO\_LOAD, [48](#)  
PMIX\_INFO\_OPTIONAL, [50](#)  
PMIX\_INFO\_REQUIRED, [49](#), [50](#)  
PMIX\_INFO\_TRUE, [49](#)  
PMIX\_INFO\_XFER, [48](#), [259](#)  
PMIX\_LOAD\_KEY, [28](#)  
PMIX\_LOAD\_NSPACE, [30](#)  
PMIX\_LOAD\_PROCID, [32](#), [33](#)  
PMIX\_MULTICLUSTER\_NSPACE\_CONSTRUCT, [33](#)  
PMIX\_MULTICLUSTER\_NSPACE\_PARSE, [34](#)  
PMIX\_PDATA\_CONSTRUCT, [55](#)  
PMIX\_PDATA\_CREATE, [55](#)  
PMIX\_PDATA\_DESTRUCT, [55](#)  
PMIX\_PDATA\_FREE, [56](#)  
PMIX\_PDATA\_LOAD, [56](#)  
PMIX\_PDATA\_RELEASE, [55](#)  
PMIX\_PDATA\_XFER, [57](#)  
PMIX\_PROC\_CONSTRUCT, [31](#)  
PMIX\_PROC\_CREATE, [32](#)  
PMIX\_PROC\_DESTRUCT, [31](#)  
PMIX\_PROC\_FREE, [32](#), [180](#)  
PMIX\_PROC\_INFO\_CONSTRUCT, [36](#)  
PMIX\_PROC\_INFO\_CREATE, [37](#)  
PMIX\_PROC\_INFO\_DESTRUCT, [36](#)  
PMIX\_PROC\_INFO\_FREE, [37](#)  
PMIX\_PROC\_INFO\_RELEASE, [37](#)  
PMIX\_PROC\_LOAD, [32](#)  
PMIX\_PROC\_RELEASE, [32](#)  
PMIX\_QUERY\_CONSTRUCT, [60](#)  
PMIX\_QUERY\_CREATE, [60](#)

PMIX\_QUERY\_DESTRUCT, [60](#)  
PMIX\_QUERY\_FREE, [61](#)  
PMIX\_QUERY\_QUALIFIERS\_CREATE, [8](#), [9](#), [61](#)  
PMIX\_QUERY\_RELEASE, [60](#)  
PMIX\_REGATTR\_CONSTRUCT, [63](#)  
PMIX\_REGATTR\_CREATE, [63](#)  
PMIX\_REGATTR\_DESTRUCT, [63](#)  
PMIX\_REGATTR\_FREE, [63](#)  
PMIX\_REGATTR\_LOAD, [64](#)  
PMIX\_REGATTR\_XFER, [64](#)  
PMIX\_SETENV, [72](#)  
PMIX\_SYSTEM\_EVENT, [27](#)  
PMIX\_TOPO\_CONSTRUCT, [177](#)  
PMIX\_VALUE\_CONSTRUCT, [43](#)  
PMIX\_VALUE\_CREATE, [43](#)  
PMIX\_VALUE\_DESTRUCT, [43](#)  
PMIX\_VALUE\_FREE, [44](#)  
PMIX\_VALUE\_GET\_NUMBER, [46](#)  
PMIX\_VALUE\_LOAD, [44](#)  
PMIX\_VALUE\_RELEASE, [43](#)  
PMIX\_VALUE\_UNLOAD, [45](#)  
PMIX\_VALUE\_XFER, [45](#)

# Index of Data Structures

---

pmix\_alloc\_directive\_t, 52, 52, 74, 118, 320, 431, 469  
pmix\_app\_t, 8, 9, 57, 57–59, 68–70, 72, 156, 161, 299, 302, 396, 398–400, 409, 411–414, 432  
pmix\_byte\_object\_t, 65, 65, 66, 73, 113, 237, 241, 243, 278, 330, 334, 424, 430  
pmix\_coord\_t, 74, 341, 341, 342  
pmix\_coord\_view\_t, 343, 348  
pmix\_cpuset\_t, 281, 281, 282  
pmix\_data\_array\_t, 8, 9, 40, 40, 41, 66, 66–68, 74, 91, 95, 184, 189, 191, 194, 197, 252, 254–256, 261–264, 273, 315, 322, 338, 341, 346, 348, 349, 351, 410, 431  
pmix\_data\_buffer\_t, 226, 226–231, 235  
pmix\_data\_range\_t, 39, 39, 74, 117, 223, 311, 430, 468  
pmix\_data\_type\_t, 40, 41, 44, 46, 48, 56, 64, 67, 73, 73, 74, 117, 230, 232–234, 429, 468  
pmix\_envar\_t, 52, 53, 54, 74, 431  
pmix\_fabric\_operation\_t, 338, 344, 344  
pmix\_fabric\_t, 339, 340, 344, 344, 347, 348, 351–356, 358–360  
pmix\_group\_operation\_t, 335, 364, 364, 382, 385  
pmix\_group\_opt\_t, 64, 64, 461  
pmix\_info\_directives\_t, 49, 49, 50, 74, 117, 431, 468  
pmix\_info\_t, 4, 7, 8, 10, 13, 28, 39, 46, 46–52, 59, 61, 62, 64, 74, 81, 82, 92, 93, 95, 97, 105, 107, 108, 114, 115, 122–124, 144, 148, 181, 185, 191–194, 196, 197, 199, 201, 204, 205, 209, 211, 214, 223, 237, 239, 241, 243, 246, 248, 252, 254, 259, 261–264, 273, 278–280, 311, 316, 317, 319, 321, 322, 325, 326, 332, 339, 345, 346, 349, 356, 359, 360, 365, 367, 369, 372, 374, 376, 379, 382, 385–387, 396, 400, 415, 419, 421, 423, 424, 431, 433  
pmix\_iof\_channel\_t, 52, 52, 74, 115, 118, 278, 332, 421, 431, 469  
pmix\_job\_state\_t, 35, 35, 38, 38, 74, 118, 470  
pmix\_key\_t, 27, 27, 28, 64, 126, 128, 429  
pmix\_link\_state\_t, 74, 119, 343, 343, 347, 350, 357, 432, 471  
pmix\_locality\_t, 177, 178, 178  
pmix\_nspace\_t, 29, 29, 30, 32–34, 103, 429, 430  
pmix\_pdata\_t, 54, 54–57, 104, 148, 431  
pmix\_persistence\_t, 39, 39, 74, 117, 430, 467  
pmix\_proc\_info\_t, 36, 36, 37, 74, 91, 184, 189, 315, 410, 430  
pmix\_proc\_state\_t, 34, 34, 74, 116, 430, 466  
pmix\_proc\_t, 30, 31, 31–33, 56, 64, 73, 80, 87, 108, 112, 115, 123, 130, 139–141, 154, 220, 223, 224, 230, 231, 256, 268–271, 278, 284, 285, 287, 288, 291, 292, 295, 297, 299, 304, 306, 311, 313, 318, 320, 323, 326, 328, 330, 332, 334, 335, 338, 365, 369, 376, 379, 383, 415, 417–420, 430  
pmix\_query\_t, 8, 9, 59, 59–61, 74, 183, 184, 188, 189, 191, 313, 316, 432  
pmix\_rank\_t, 30, 30–33, 74, 430

pmix\_regattr\_t, 10, [61](#), 61–64, 74, 99, 191, 275, 432  
pmix\_scope\_t, [38](#), 38, 74, 117, 127, 430, 467  
pmix\_status\_t, [21](#), 21, 27, 45, 46, 68, 69, 72, 73, 105–108, 110–114, 116, 218, 223, 308, 310, 311,  
        [429](#), [441](#), 466  
pmix\_topology\_t, [176](#), [177](#), 177  
pmix\_value\_t, [41](#), 41–46, 73, 105, 126, 127, 431

# Index of Constants

---

PMIX\_ALLOC\_DIRECTIVE, [74](#)  
PMIX\_ALLOC\_EXTEND, [52](#)  
PMIX\_ALLOC\_EXTERNAL, [52](#)  
PMIX\_ALLOC\_NEW, [52](#)  
PMIX\_ALLOC\_REAQUIRE, [52](#)  
PMIX\_ALLOC\_RELEASE, [52](#)  
PMIX\_APP, [73](#)  
PMIX\_APP\_WILDCARD, [20](#)  
PMIX\_BOOL, [73](#)  
PMIX\_BUFFER, [73](#)  
PMIX\_BYTE, [73](#)  
PMIX\_BYTE\_OBJECT, [73](#)  
PMIX\_COMMAND, [74](#)  
PMIX\_COMPRESSED\_STRING, [74](#)  
PMIX\_CONNECT\_REQUESTED, [24](#)  
PMIX\_COORD, [74](#)  
PMIX\_COORD\_LOGICAL\_VIEW, [343](#)  
PMIX\_COORD\_PHYSICAL\_VIEW, [343](#)  
PMIX\_COORD\_VIEW\_UNDEF, [343](#)  
PMIX\_DATA\_ARRAY, [74](#)  
PMIX\_DATA\_RANGE, [74](#)  
PMIX\_DATA\_TYPE, [74](#)  
PMIX\_DATA\_TYPE\_MAX, [74](#)  
PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY, [413](#)  
PMIX\_DEBUGGER\_RELEASE, [413](#)  
PMIX\_DOUBLE, [73](#)  
PMIX\_ENVAR, [74](#)  
PMIX\_ERR\_BAD\_PARAM, [22](#)  
PMIX\_ERR\_COMM\_FAILURE, [22](#)  
PMIX\_ERR\_CONFLICTING\_CLEANUP\_DIRECTIVES, [22](#)  
PMIX\_ERR\_DATA\_VALUE\_NOT\_FOUND, [22](#)  
PMIX\_ERR\_DEBUGGER\_RELEASE, [413](#)  
PMIX\_ERR\_DUPLICATE\_KEY, [24](#)  
PMIX\_ERR\_EVENT\_REGISTRATION, [23](#)  
PMIX\_ERR\_GET\_MALLOC\_REQD, [25](#)  
PMIX\_ERR\_HANDSHAKE\_FAILED, [21](#)  
PMIX\_ERR\_IN\_ERRNO, [22](#)  
PMIX\_ERR\_INIT, [22](#)

PMIX\_ERR\_INVALID\_ARG, [22](#)  
PMIX\_ERR\_INVALID\_ARGS, [22](#)  
PMIX\_ERR\_INVALID\_CRED, [21](#)  
PMIX\_ERR\_INVALID\_KEY, [22](#)  
PMIX\_ERR\_INVALID\_KEY\_LENGTH, [22](#)  
PMIX\_ERR\_INVALID\_KEYVALP, [22](#)  
PMIX\_ERR\_INVALID\_LENGTH, [22](#)  
PMIX\_ERR\_INVALID\_NAMESPACE, [22](#)  
PMIX\_ERR\_INVALID\_NUM\_ARGS, [22](#)  
PMIX\_ERR\_INVALID\_NUM\_PARSED, [22](#)  
PMIX\_ERR\_INVALID\_OPERATION, [24](#)  
PMIX\_ERR\_INVALID\_SIZE, [22](#)  
PMIX\_ERR\_INVALID\_TERMINATION, [23](#)  
PMIX\_ERR\_INVALID\_VAL, [22](#)  
PMIX\_ERR\_INVALID\_VAL\_LENGTH, [22](#)  
PMIX\_ERR\_IOF\_COMPLETE, [25](#)  
PMIX\_ERR\_IOF\_FAILURE, [25](#)  
PMIX\_ERR\_JOB\_ABORTED, [23](#), [26](#)  
PMIX\_ERR\_JOB\_ABORTED\_BY\_SIG, [23](#), [26](#)  
PMIX\_ERR\_JOB\_ABORTED\_BY\_SYS\_EVENT, [26](#)  
PMIX\_ERR\_JOB\_ALLOC\_FAILED, [23](#), [26](#)  
PMIX\_ERR\_JOB\_APP\_NOT\_EXECUTABLE, [23](#), [26](#)  
PMIX\_ERR\_JOB\_CANCELED, [26](#)  
PMIX\_ERR\_JOB\_CANCELLED, [23](#)  
PMIX\_ERR\_JOB\_CANNOT\_LAUNCH, [23](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_LAUNCH, [23](#), [26](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_MAP, [23](#), [26](#)  
PMIX\_ERR\_JOB\_FAILED\_TO\_START, [23](#)  
PMIX\_ERR\_JOB\_KILLED\_BY\_CMD, [23](#), [26](#)  
PMIX\_ERR\_JOB\_NEVER\_LAUNCHED, [23](#)  
PMIX\_ERR\_JOB\_NO\_EXE\_SPECIFIED, [23](#), [26](#)  
PMIX\_ERR\_JOB\_NON\_ZERO\_TERM, [23](#), [26](#)  
PMIX\_ERR\_JOB\_SENSOR\_BOUND\_EXCEEDED, [23](#), [26](#)  
PMIX\_ERR\_JOB\_TERM\_WO\_SYNC, [23](#), [26](#)  
PMIX\_ERR\_JOB\_TERMINATED, [23](#)  
PMIX\_ERR\_LOST\_CONNECTION\_TO\_CLIENT, [22](#)  
PMIX\_ERR\_LOST\_CONNECTION\_TO\_SERVER, [22](#)  
PMIX\_ERR\_LOST\_PEER\_CONNECTION, [22](#)  
PMIX\_ERR\_NO\_PERMISSIONS, [22](#)  
PMIX\_ERR\_NODE\_DOWN, [26](#)  
PMIX\_ERR\_NODE\_OFFLINE, [26](#)  
PMIX\_ERR\_NOMEM, [22](#)  
PMIX\_ERR\_NOT\_FOUND, [22](#)

PMIX\_ERR\_NOT\_IMPLEMENTED, [22](#)  
PMIX\_ERR\_NOT\_SUPPORTED, [22](#)  
PMIX\_ERR\_OUT\_OF\_RESOURCE, [22](#)  
PMIX\_ERR\_PACK\_FAILURE, [22](#)  
PMIX\_ERR\_PACK\_MISMATCH, [22](#)  
PMIX\_ERR\_PARTIAL\_SUCCESS, [24](#)  
PMIX\_ERR\_PROC\_ABORTED, [21](#)  
PMIX\_ERR\_PROC\_ABORTING, [21](#)  
PMIX\_ERR\_PROC\_CHECKPOINT, [21](#)  
PMIX\_ERR\_PROC\_ENTRY\_NOT\_FOUND, [21](#)  
PMIX\_ERR\_PROC\_MIGRATE, [21](#)  
PMIX\_ERR\_PROC\_REQUESTED\_ABORT, [21](#)  
PMIX\_ERR\_PROC\_RESTART, [21](#)  
PMIX\_ERR\_READY\_FOR\_HANDSHAKE, [21](#)  
PMIX\_ERR\_REPEAT\_ATTR\_REGISTRATION, [25](#)  
PMIX\_ERR\_RESOURCE\_BUSY, [22](#)  
PMIX\_ERR\_SERVER\_FAILED\_REQUEST, [21](#)  
PMIX\_ERR\_SERVER\_NOT\_AVAIL, [22](#)  
PMIX\_ERR\_SILENT, [21](#)  
PMIX\_ERR\_SYS\_BASE, [26](#)  
PMIX\_ERR\_SYS\_OTHER, [26](#)  
PMIX\_ERR\_TIMEOUT, [22](#)  
PMIX\_ERR\_TYPE\_MISMATCH, [21](#)  
PMIX\_ERR\_UNKNOWN\_DATA\_TYPE, [21](#)  
PMIX\_ERR\_UNPACK\_FAILURE, [21](#)  
PMIX\_ERR\_UNPACK\_INADEQUATE\_SPACE, [21](#)  
PMIX\_ERR\_UNPACK\_READ\_PAST\_END\_OF\_BUFFER, [22](#)  
PMIX\_ERR\_UNREACH, [22](#)  
PMIX\_ERR\_UPDATE\_ENDPOINTS, [23](#)  
PMIX\_ERR\_WOULD\_BLOCK, [21](#)  
PMIX\_ERROR, [21](#)  
PMIX\_EVENT\_ACTION\_COMPLETE, [26](#)  
PMIX\_EVENT\_ACTION\_DEFERRED, [26](#)  
PMIX\_EVENT\_JOB\_END, [25](#)  
PMIX\_EVENT\_JOB\_START, [25](#)  
PMIX\_EVENT\_NO\_ACTION\_TAKEN, [26](#)  
PMIX\_EVENT\_PARTIAL\_ACTION\_TAKEN, [26](#)  
PMIX\_EVENT\_SESSION\_END, [25](#)  
PMIX\_EVENT\_SESSION\_START, [25](#)  
PMIX\_EXISTS, [21](#)  
PMIX\_EXTERNAL\_ERR\_BASE, [27](#)  
PMIX\_FABRIC\_GET\_DEVICE\_INDEX, [344](#)  
PMIX\_FABRIC\_GET\_VERTEX\_INFO, [344](#)

PMIX\_FABRIC\_REQUEST\_INFO, [344](#)  
PMIX\_FABRIC\_UPDATE\_INFO, [344](#)  
PMIX\_FABRIC\_UPDATE\_PENDING, [340](#)  
PMIX\_FABRIC\_UPDATED, [340](#)  
PMIX\_FLOAT, [73](#)  
PMIX\_FWD\_ALL\_CHANNELS, [52](#)  
PMIX\_FWD\_NO\_CHANNELS, [52](#)  
PMIX\_FWD\_STDDIAG\_CHANNEL, [52](#)  
PMIX\_FWD\_STDERR\_CHANNEL, [52](#)  
PMIX\_FWD\_STDIN\_CHANNEL, [52](#)  
PMIX\_FWD\_STDOUT\_CHANNEL, [52](#)  
PMIX\_GDS\_ACTION\_COMPLETE, [23](#)  
PMIX\_GLOBAL, [38](#)  
PMIX\_GROUP\_ACCEPT, [64](#), [364](#)  
PMIX\_GROUP\_CONSTRUCT, [364](#)  
PMIX\_GROUP\_CONSTRUCT\_ABORT, [24](#)  
PMIX\_GROUP\_CONSTRUCT\_COMPLETE, [25](#)  
PMIX\_GROUP\_CONTEXT\_ID\_ASSIGNED, [25](#)  
PMIX\_GROUP\_DECLINE, [64](#), [364](#)  
PMIX\_GROUP\_DESTRUCT, [364](#)  
PMIX\_GROUP\_INVITE\_ACCEPTED, [24](#)  
PMIX\_GROUP\_INVITE\_DECLINED, [24](#)  
PMIX\_GROUP\_INVITE\_FAILED, [24](#)  
PMIX\_GROUP\_INVITED, [24](#)  
PMIX\_GROUP\_LEADER\_FAILED, [25](#)  
PMIX\_GROUP\_LEADER\_SELECTED, [25](#)  
PMIX\_GROUP\_LEFT, [24](#)  
PMIX\_GROUP\_MEMBER\_FAILED, [24](#)  
PMIX\_GROUP\_MEMBERSHIP\_UPDATE, [24](#)  
PMIX\_INFO, [73](#)  
PMIX\_INFO\_ARRAY\_END, [50](#)  
PMIX\_INFO\_DIR\_RESERVED, [50](#)  
PMIX\_INFO\_DIRECTIVES, [74](#)  
PMIX\_INFO REQD, [50](#)  
PMIX\_INT, [73](#)  
PMIX\_INT16, [73](#)  
PMIX\_INT32, [73](#)  
PMIX\_INT64, [73](#)  
PMIX\_INT8, [73](#)  
PMIX\_INTERNAL, [38](#)  
PMIX\_IOF\_CHANNEL, [74](#)  
PMIX\_JCTRL\_CHECKPOINT, [22](#)  
PMIX\_JCTRL\_CHECKPOINT\_COMPLETE, [22](#)

PMIX\_JCTRL\_PREEMPT\_ALERT, [23](#)  
PMIX\_JOB\_STATE, [74](#)  
PMIX\_JOB\_STATE\_CONNECTED, [35](#), [38](#)  
PMIX\_JOB\_STATE\_LAUNCH\_UNDERWAY, [35](#), [38](#)  
PMIX\_JOB\_STATE\_PREPPED, [35](#), [38](#)  
PMIX\_JOB\_STATE\_RUNNING, [35](#), [38](#)  
PMIX\_JOB\_STATE\_SUSPENDED, [35](#), [38](#)  
PMIX\_JOB\_STATE\_TERMINATED, [35](#), [39](#)  
PMIX\_JOB\_STATE\_TERMINATED\_WITH\_ERROR, [35](#), [39](#)  
PMIX\_JOB\_STATE\_UNDEF, [35](#), [38](#)  
PMIX\_JOB\_STATE\_UNTERMINATED, [38](#)  
PMIX\_KVAL, [73](#)  
PMIX\_LAUNCH\_COMPLETE, [25](#)  
PMIX\_LAUNCH\_DIRECTIVE, [403](#)  
PMIX\_LAUNCHER\_READY, [403](#)  
PMIX\_LINK\_DOWN, [343](#)  
PMIX\_LINK\_STATE, [74](#)  
PMIX\_LINK\_STATE\_UNKNOWN, [343](#)  
PMIX\_LINK\_UP, [343](#)  
PMIX\_LOCAL, [38](#)  
PMIX\_LOCALITY\_NONLOCAL, [178](#)  
PMIX\_LOCALITY\_SHARE\_CORE, [178](#)  
PMIX\_LOCALITY\_SHARE\_HWTHREAD, [178](#)  
PMIX\_LOCALITY\_SHARE\_L1CACHE, [178](#)  
PMIX\_LOCALITY\_SHARE\_L2CACHE, [178](#)  
PMIX\_LOCALITY\_SHARE\_L3CACHE, [178](#)  
PMIX\_LOCALITY\_SHARE\_NODE, [178](#)  
PMIX\_LOCALITY\_SHARE\_PACKAGE, [178](#)  
PMIX\_LOCALITY\_UNKNOWN, [178](#)  
PMIX\_MAX\_KEYLEN, [20](#)  
PMIX\_MAX\_NSLEN, [20](#)  
PMIX\_MODEL\_DECLARED, [23](#)  
PMIX\_MODEL\_RESOURCES, [24](#)  
PMIX\_MONITOR\_FILE\_ALERT, [23](#)  
PMIX\_MONITOR\_HEARTBEAT\_ALERT, [23](#)  
PMIX\_NOTIFY\_ALLOC\_COMPLETE, [22](#)  
PMIX\_OPENMP\_PARALLEL\_ENTERED, [24](#)  
PMIX\_OPENMP\_PARALLEL\_EXITED, [24](#)  
PMIX\_OPERATION\_IN\_PROGRESS, [24](#)  
PMIX\_OPERATION\_SUCCEEDED, [24](#)  
PMIX\_PDATA, [73](#)  
PMIX\_PERSIST, [74](#)  
PMIX\_PERSIST\_APP, [39](#)

PMIX\_PERSIST\_FIRST\_READ, [39](#)  
PMIX\_PERSIST\_INDEF, [39](#)  
PMIX\_PERSIST\_INVALID, [39](#)  
PMIX\_PERSIST\_PROC, [39](#)  
PMIX\_PERSIST\_SESSION, [39](#)  
PMIX\_PID, [73](#)  
PMIX\_POINTER, [74](#)  
PMIX\_PROC, [73](#)  
PMIX\_PROC\_HAS\_CONNECTED, [24](#)  
PMIX\_PROC\_INFO, [74](#)  
PMIX\_PROC\_RANK, [74](#)  
PMIX\_PROC\_STATE, [74](#)  
PMIX\_PROC\_STATE\_ABORTED, [35](#)  
PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG, [35](#)  
PMIX\_PROC\_STATE\_CALLED\_ABORT, [35](#)  
PMIX\_PROC\_STATE\_CANNOT\_RESTART, [35](#)  
PMIX\_PROC\_STATE\_COMM\_FAILED, [35](#)  
PMIX\_PROC\_STATE\_CONNECTED, [34](#)  
PMIX\_PROC\_STATE\_ERROR, [34](#)  
PMIX\_PROC\_STATE FAILED\_TO\_LAUNCH, [35](#)  
PMIX\_PROC\_STATE FAILED\_TO\_START, [35](#)  
PMIX\_PROC\_STATE\_HEARTBEAT FAILED, [35](#)  
PMIX\_PROC\_STATE\_KILLED\_BY\_CMD, [34](#)  
PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY, [34](#)  
PMIX\_PROC\_STATE\_MIGRATING, [35](#)  
PMIX\_PROC\_STATE\_PREPPED, [34](#)  
PMIX\_PROC\_STATE\_RESTART, [34](#)  
PMIX\_PROC\_STATE\_RUNNING, [34](#)  
PMIX\_PROC\_STATE\_SENSOR\_BOUND\_EXCEEDED, [35](#)  
PMIX\_PROC\_STATE\_TERM\_NON\_ZERO, [35](#)  
PMIX\_PROC\_STATE\_TERM\_WO\_SYNC, [35](#)  
PMIX\_PROC\_STATE\_TERMINATE, [34](#)  
PMIX\_PROC\_STATE\_TERMINATED, [34](#)  
PMIX\_PROC\_STATE\_UNDEF, [34](#)  
PMIX\_PROC\_STATE\_UNTERMINATED, [34](#)  
PMIX\_PROC\_TERMINATED, [23](#)  
PMIX\_QUERY, [74](#)  
PMIX\_QUERY\_PARTIAL\_SUCCESS, [22](#)  
PMIX\_RANGE\_CUSTOM, [39](#)  
PMIX\_RANGE\_GLOBAL, [39](#)  
PMIX\_RANGE\_INVALID, [39](#)  
PMIX\_RANGE\_LOCAL, [39](#)  
PMIX\_RANGE\_NAMESPACE, [39](#)

PMIX\_RANGE\_PROC\_LOCAL, [39](#)  
PMIX\_RANGE\_RM, [39](#)  
PMIX\_RANGE\_SESSION, [39](#)  
PMIX\_RANGE\_UNDEF, [39](#)  
PMIX\_RANK\_INVALID, [30](#)  
PMIX\_RANK\_LOCAL\_NODE, [30](#)  
PMIX\_RANK\_LOCAL\_PEERS, [30](#)  
PMIX\_RANK\_UNDEF, [30](#)  
PMIX\_RANK\_VALID, [30](#)  
PMIX\_RANK\_WILDCARD, [30](#)  
PMIX\_REGATTR, [74](#)  
PMIX\_REGEX, [74](#)  
PMIX\_REMOTE, [38](#)  
PMIX\_SCOPE, [74](#)  
PMIX\_SCOPE\_UNDEF, [38](#)  
PMIX\_SIZE, [73](#)  
PMIX\_STATUS, [73](#)  
PMIX\_STRING, [73](#)  
PMIX\_SUCCESS, [21](#)  
PMIX\_TIME, [73](#)  
PMIX\_TIMEVAL, [73](#)  
PMIX\_UINT, [73](#)  
PMIX\_UINT16, [73](#)  
PMIX\_UINT32, [73](#)  
PMIX\_UINT64, [73](#)  
PMIX\_UINT8, [73](#)  
PMIX\_UNDEF, [73](#)  
PMIX\_VALUE, [73](#)

# Index of Environmental Variables

---

PMIX\_LAUNCHER\_PAUSE\_FOR\_TOOL, [394](#)

PMIX\_LAUNCHER\_RENDEZVOUS\_FILE, [395](#)

# Index of Attributes

---

PMIX\_ADD\_ENVAR, [94](#), [159](#), [165](#)  
PMIX\_ADD\_HOST, [88](#), [157](#), [163](#), [300](#)  
PMIX\_ADD\_HOSTFILE, [88](#), [157](#), [163](#), [300](#)  
PMIX\_ALLOC\_BANDWIDTH, [95](#), [95](#), [160](#), [165](#), [194](#), [197](#), [273](#), [322](#)  
PMIX\_ALLOC\_CPU\_LIST, [95](#), [160](#), [165](#), [194](#), [197](#), [321](#)  
PMIX\_ALLOC\_FABRIC, [95](#), [95](#), [194](#), [197](#), [273](#), [321](#)  
PMIX\_ALLOC\_FABRIC\_ENDPTS, [95](#), [96](#), [96](#), [160](#), [166](#), [194](#), [195](#), [197](#), [198](#), [273](#), [321](#)  
PMIX\_ALLOC\_FABRIC\_ENDPTS\_NODE, [96](#), [96](#), [160](#), [166](#), [195](#), [198](#), [273](#)  
PMIX\_ALLOC\_FABRIC\_ID, [95](#), [95](#), [194](#), [197](#), [273](#), [321](#)  
PMIX\_ALLOC\_FABRIC\_PLANE, [95](#), [96](#), [96](#), [160](#), [166](#), [194](#), [197](#), [198](#), [273](#), [322](#)  
PMIX\_ALLOC\_FABRIC\_QOS, [95](#), [96](#), [160](#), [165](#), [194](#), [197](#), [198](#), [273](#), [322](#)  
PMIX\_ALLOC\_FABRIC\_SEC\_KEY, [95](#), [96](#), [96](#), [194](#), [195](#), [197](#), [198](#), [273](#), [322](#)  
PMIX\_ALLOC\_FABRIC\_TYPE, [95](#), [96](#), [96](#), [160](#), [165](#), [194](#), [197](#), [198](#), [273](#), [321](#), [322](#)  
PMIX\_ALLOC\_ID, [10](#), [94](#), [195](#), [321](#)  
PMIX\_ALLOC\_MEM\_SIZE, [95](#), [160](#), [165](#), [194](#), [197](#), [321](#)  
PMIX\_ALLOC\_NETWORK (**Deprecated**), [95](#)  
PMIX\_ALLOC\_NETWORK\_ENDPTS (**Deprecated**), [96](#)  
PMIX\_ALLOC\_NETWORK\_ENDPTS\_NODE (**Deprecated**), [96](#)  
PMIX\_ALLOC\_NETWORK\_ID (**Deprecated**), [95](#)  
PMIX\_ALLOC\_NETWORK\_PLANE (**Deprecated**), [96](#)  
PMIX\_ALLOC\_NETWORK\_QOS (**Deprecated**), [95](#)  
PMIX\_ALLOC\_NETWORK\_SEC\_KEY (**Deprecated**), [96](#)  
PMIX\_ALLOC\_NETWORK\_TYPE (**Deprecated**), [96](#)  
PMIX\_ALLOC\_NODE\_LIST, [95](#), [159](#), [165](#), [194](#), [197](#), [321](#)  
PMIX\_ALLOC\_NUM\_CPU\_LIST, [95](#), [160](#), [165](#), [194](#), [197](#), [321](#)  
PMIX\_ALLOC\_NUM\_CPUS, [95](#), [159](#), [165](#), [193](#), [197](#), [321](#)  
PMIX\_ALLOC\_NUM\_NODES, [95](#), [159](#), [165](#), [193](#), [196](#), [321](#)  
PMIX\_ALLOC\_QUEUE, [91](#), [94](#), [159](#), [165](#), [184](#), [189](#), [315](#)  
PMIX\_ALLOC\_REQ\_ID, [10](#), [94](#), [193](#), [196](#)  
PMIX\_ALLOC\_TIME, [96](#), [159](#), [165](#), [193](#), [197](#), [274](#), [321](#)  
PMIX\_ALLOCATED\_NODELIST, [79](#), [252](#)  
PMIX\_ANL\_MAP, [86](#), [254](#)  
PMIX\_APP\_ARGV, [90](#), [254](#)  
PMIX\_APP\_INFO, [81](#), [129](#), [132](#), [136](#), [182](#), [187](#), [254](#)  
PMIX\_APP\_INFO\_ARRAY, [81](#), [82](#), [254](#), [263](#)  
PMIX\_APP\_MAP\_REGEX, [86](#)  
PMIX\_APP\_MAP\_TYPE, [86](#)  
PMIX\_APP\_RANK, [79](#), [257](#)

PMIX\_APP\_SIZE, [82](#), 136, 254, 262  
PMIX\_APPEND\_ENVAR, [94](#), 159, 165  
PMIX\_APPLDR, [79](#), 254, 262  
PMIX\_APPNUM, [79](#), 81, 129, 132, 136, 182, 187, 254, 256, 263  
PMIX\_ARCH, [78](#)  
PMIX\_ATTR\_UNDEF, [75](#)  
PMIX\_AVAIL\_PHYS\_MEMORY, [83](#), 256  
PMIX\_BINDTO, [89](#), 158, 163, 253, 301  
PMIX\_CLEANUP\_EMPTY, [97](#), 200, 203  
PMIX\_CLEANUP\_IGNORE, [97](#), 200, 203  
PMIX\_CLEANUP\_LEAVE\_TOPDIR, [97](#), 200, 203  
PMIX\_CLEANUP\_RECURSIVE, [97](#), 200, 203  
PMIX\_CLIENT\_ATTRIBUTES, [11](#), [99](#), 183, 188, 410  
PMIX\_CLIENT\_AVG\_MEMORY, [83](#)  
PMIX\_CLIENT\_FUNCTIONS, [92](#), [99](#), 182, 187  
PMIX\_CLUSTER\_ID, [79](#), 252  
PMIX\_CMD\_LINE, [90](#)  
PMIX\_COLLECT\_DATA, [84](#), 139, 141, 289  
PMIX\_COLLECTIVE\_ALGO, [7](#), [84](#), 139, 142, 168, 171, 289, 304  
PMIX\_COLLECTIVE\_ALGO\_REQD, 139, 142, 168, 171, 289, 305  
PMIX\_COLLECTIVE\_ALGO\_REQD (Deprecated), [85](#)  
PMIX\_CONNECT\_MAX\_RETRIES, [395](#), 416  
PMIX\_CONNECT\_RETRY\_DELAY, [395](#), 416  
PMIX\_CONNECT\_SYSTEM\_FIRST, [393](#), [395](#), 415, 419  
PMIX\_CONNECT\_TO\_SYSTEM, [392](#), [395](#), 415, 419  
PMIX\_COSPAWN\_APP, [414](#)  
PMIX\_CPU\_LIST, [90](#), 158, 164, 302  
PMIX\_CPUS\_PER\_PROC, [89](#), 158, 164, 302  
PMIX\_CPUSET, [78](#)  
PMIX\_CRED\_TYPE, [98](#), 329  
PMIX\_CREDENTIAL, [78](#)  
PMIX\_CRYPTO\_KEY, [98](#)  
PMIX\_DAEMON\_MEMORY, [83](#)  
PMIX\_DATA\_SCOPE, [85](#), 128, 131  
PMIX\_DEBUG\_DAEMONS\_PER\_NODE, 302, 411, [414](#), 414  
PMIX\_DEBUG\_DAEMONS\_PER\_PROC, 302, 411, [414](#), 414  
PMIX\_DEBUG\_JOB (Deprecated), [413](#)  
PMIX\_DEBUG\_STOP\_IN\_INIT, [399](#), 409, 410, [413](#)  
PMIX\_DEBUG\_STOP\_ON\_EXEC, [398](#), 409, 410, [413](#)  
PMIX\_DEBUG\_TARGET, 302, 410, 411, [413](#), 413, 414  
PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY, [399](#), 409, [413](#)  
PMIX\_DEBUGGER\_DAEMONS, 302, 411, [414](#)  
PMIX\_DISPLAY\_MAP, [88](#), 157, 163, 301

PMIX\_DSTPATH, [76](#)  
PMIX\_EMBED\_BARRIER, [85](#), [124](#)  
PMIX\_ENUM\_VALUE, [11](#), [62](#), [99](#)  
PMIX\_EVENT\_ACTION\_TIMEOUT, [88](#), [220](#)  
PMIX\_EVENT\_AFFECTED\_PROC, [87](#), [220](#), [224](#)  
PMIX\_EVENT\_AFFECTED\_PROCS, [87](#), [220](#), [224](#)  
PMIX\_EVENT\_BASE, [75](#), [123](#), [247](#), [416](#)  
PMIX\_EVENT\_CUSTOM\_RANGE, [87](#), [220](#), [224](#)  
PMIX\_EVENT\_DO\_NOT\_CACHE, [87](#)  
PMIX\_EVENT\_HDLR\_AFTER, [87](#), [219](#)  
PMIX\_EVENT\_HDLR\_APPEND, [87](#), [220](#)  
PMIX\_EVENT\_HDLR\_BEFORE, [87](#), [219](#)  
PMIX\_EVENT\_HDLR\_FIRST, [86](#), [219](#)  
PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY, [87](#), [219](#)  
PMIX\_EVENT\_HDLR\_LAST, [86](#), [219](#)  
PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY, [87](#), [219](#)  
PMIX\_EVENT\_HDLR\_NAME, [86](#), [219](#)  
PMIX\_EVENT\_HDLR\_PREPEND, [87](#), [220](#)  
PMIX\_EVENT\_NO\_TERMINATION, [88](#)  
PMIX\_EVENT\_NON\_DEFAULT, [87](#), [224](#)  
PMIX\_EVENT\_PROXY, [87](#)  
PMIX\_EVENT\_RETURN\_OBJECT, [87](#), [220](#)  
PMIX\_EVENT\_SILENT\_TERMINATION, [87](#), [220](#)  
PMIX\_EVENT\_TERMINATE\_JOB, [87](#), [220](#)  
PMIX\_EVENT\_TERMINATE\_NODE, [87](#), [220](#)  
PMIX\_EVENT\_TERMINATE\_PROC, [88](#), [220](#)  
PMIX\_EVENT\_TERMINATE\_SESSION, [87](#), [220](#)  
PMIX\_EVENT\_TEXT\_MESSAGE, [87](#)  
PMIX\_EVENT\_TIMESTAMP, [25](#), [87](#), [90](#), [398](#)  
PMIX\_EVENT\_WANT\_TERMINATION, [88](#)  
PMIX\_EXEC\_AGENT, [402](#), [403](#), [412](#)  
PMIX\_EXIT\_CODE, [80](#)  
PMIX\_FABRIC\_COORDINATE, [348](#)  
PMIX\_FABRIC\_COST\_MATRIX, [345](#), [348](#)  
PMIX\_FABRIC\_DEVICE, [346](#), [349](#)  
PMIX\_FABRIC\_DEVICE\_ADDRESS, [346](#), [350](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_BUS\_TYPE, [347](#), [349](#), [356](#)  
PMIX\_FABRIC\_DEVICE\_DRIVER, [346](#), [349](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_FIRMWARE, [346](#), [350](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_ID, [346](#), [349](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_INDEX, [339](#), [344](#), [349](#), [349](#)  
PMIX\_FABRIC\_DEVICE\_MTU, [347](#), [350](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_NAME, [346](#), [349](#), [356](#)

PMIX\_FABRIC\_DEVICE\_PCI\_DEVID, 347, [350](#), [350](#), [356](#), [357](#), [359](#)  
PMIX\_FABRIC\_DEVICE\_SPEED, 347, [350](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_STATE, 347, [350](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_TYPE, 347, [350](#), [357](#)  
PMIX\_FABRIC\_DEVICE\_VENDOR, 346, [349](#), [356](#)  
PMIX\_FABRIC\_DIMS, [345](#), [348](#)  
PMIX\_FABRIC\_ENDPT, [349](#)  
PMIX\_FABRIC\_GROUPS, [345](#), [348](#)  
PMIX\_FABRIC\_IDENTIFIER, 338, 345, [348](#), [351](#)  
PMIX\_FABRIC\_INDEX, [344](#), [348](#), [360](#)  
PMIX\_FABRIC\_NUM\_VERTICES, [345](#), [348](#)  
PMIX\_FABRIC\_PLANE, 338, 341, 346, [348](#), [351](#), [352](#)  
PMIX\_FABRIC\_SHAPE, [346](#), [349](#)  
PMIX\_FABRIC\_SHAPE\_STRING, [346](#), [349](#)  
PMIX\_FABRIC\_SWITCH, [349](#)  
PMIX\_FABRIC\_VENDOR, 338, 345, [348](#), [351](#)  
PMIX\_FABRIC\_VIEW, [348](#)  
PMIX\_FIRST\_ENVAR, [94](#), [159](#), [165](#)  
PMIX\_FORK\_EXEC\_AGENT, [90](#)  
PMIX\_FORKEXEC\_AGENT, [402](#), [403](#), [412](#)  
PMIX\_FWD\_STDDIAG, 9, [396](#), [401](#), [403](#)  
PMIX\_FWD\_STDERR, 301, [317](#), [396](#), [401](#), [403](#), [404](#)  
PMIX\_FWD\_STDIN, 301, [317](#), [396](#), [400](#), [402](#)  
PMIX\_FWD\_STDOUT, 301, [317](#), [396](#), [401](#), [403](#), [404](#)  
PMIX\_GDS\_MODULE, [78](#), [123](#)  
PMIX\_GET\_REFRESH\_CACHE, [85](#)  
PMIX\_GET\_STATIC\_VALUES, [85](#), [129](#), [132](#)  
PMIX\_GLOBAL\_RANK, [79](#), [257](#)  
PMIX\_GROUP\_ASSIGN\_CONTEXT\_ID, [100](#), [336](#), [337](#), [366](#), [370](#), [377](#), [380](#)  
PMIX\_GROUP\_CONTEXT\_ID, [100](#), [337](#)  
PMIX\_GROUP\_ENDPT\_DATA, [100](#), [336](#), [337](#)  
PMIX\_GROUP\_FT\_COLLECTIVE, [100](#)  
PMIX\_GROUP\_ID, [100](#), [337](#)  
PMIX\_GROUP\_INVITE\_DECLINE, [100](#)  
PMIX\_GROUP\_LEADER, [100](#), [365](#), [367](#), [370](#), [379](#), [384](#)  
PMIX\_GROUP\_LOCAL\_ONLY, [100](#), [336](#), [366](#), [370](#)  
PMIX\_GROUP\_MEMBERSHIP, [100](#), [337](#), [367](#)  
PMIX\_GROUP\_NOTIFY\_TERMINATION, [100](#), [366](#), [367](#), [370](#), [373](#), [377](#), [380](#)  
PMIX\_GROUP\_OPTIONAL, [100](#), [336](#), [365](#), [367](#), [370](#), [376](#), [380](#)  
PMIX\_GRPID, [76](#), [114](#), [143](#), [145](#), [147](#), [149](#), [151](#), [153](#), [183](#), [188](#), [193](#), [196](#), [200](#), [202](#), [206](#), [208](#), [210](#),  
    [213](#), [238](#), [239](#), [241](#), [243](#), [293–296](#), [298](#), [300](#), [309](#), [314](#), [316](#), [318](#), [320](#), [324](#), [326](#), [328](#), [331](#),  
    [332](#), [335](#)  
PMIX\_HOST, [88](#), [157](#), [162](#), [300](#)

PMIX\_HOST\_ATTRIBUTES, 11, [99](#), 183, 188, 191, 410  
PMIX\_HOST\_FUNCTIONS, [92](#), [99](#), 182, 187  
PMIX\_HOSTFILE, [88](#), [157](#), 162, 300  
PMIX\_HOSTNAME, [80](#), 80–82, 129, 132, 137, 182, 187, 255, 257, 346, 347, 350, 356, 357, 359  
PMIX\_HOSTNAME\_ALIASES, [80](#), 255  
PMIX\_HOSTNAME\_KEEP\_FQDN, [80](#), 254  
PMIX\_HWLOC\_HOLE\_KIND (**Deprecated**), [84](#)  
PMIX\_HWLOC\_SHARE\_TOPO (**Deprecated**), [84](#)  
PMIX\_HWLOC\_SHMEM\_ADDR, [256](#)  
PMIX\_HWLOC\_SHMEM\_ADDR (**Deprecated**), [84](#)  
PMIX\_HWLOC\_SHMEM\_FILE, [256](#)  
PMIX\_HWLOC\_SHMEM\_FILE (**Deprecated**), [84](#)  
PMIX\_HWLOC\_SHMEM\_SIZE, [256](#)  
PMIX\_HWLOC\_SHMEM\_SIZE (**Deprecated**), [84](#)  
PMIX\_HWLOC\_XML\_V1, [256](#)  
PMIX\_HWLOC\_XML\_V1 (**Deprecated**), [84](#)  
PMIX\_HWLOC\_XML\_V2, [256](#)  
PMIX\_HWLOC\_XML\_V2 (**Deprecated**), [84](#)  
PMIX\_IMMEDIATE, [84](#), 128, 130, 131  
PMIX\_INDEX\_ARGV, [89](#), 158, 164, 302  
PMIX\_IOF\_BUFFERING\_SIZE, 333, 397, 401, [407](#), 422, 425  
PMIX\_IOF\_BUFFERING\_TIME, 333, 397, 401, [407](#), 422, 425  
PMIX\_IOF\_CACHE\_SIZE, 333, 397, 401, [407](#), 421, 425  
PMIX\_IOF\_COMPLETE, [115](#), 406, [407](#), 408, 425, 434  
PMIX\_IOF\_COPY, [405](#), [408](#)  
PMIX\_IOF\_DROP\_NEWEST, 333, 397, 401, [407](#), 422, 425  
PMIX\_IOF\_DROP\_OLDEST, 333, 397, 401, [407](#), 421, 425  
PMIX\_IOF\_PUSH\_STDIN, [406](#), [408](#), 425  
PMIX\_IOF\_REDIRECT, [405](#), [408](#)  
PMIX\_IOF\_TAG\_OUTPUT, 398, 401, 405, [408](#), 422  
PMIX\_IOF\_TIMESTAMP\_OUTPUT, 398, 401, 405, [408](#), 422  
PMIX\_IOF\_XML\_OUTPUT, 398, 401, 405, [408](#), 422  
PMIX\_JOB\_CONTINUOUS, [90](#), 159, 164, 303  
PMIX\_JOB\_CTRL\_CANCEL, [96](#), 201, 204, 324  
PMIX\_JOB\_CTRL\_CHECKPOINT, [97](#), 201, 204, 324  
PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT, [97](#), 201, 204, 324  
PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD, [97](#), 201, 204, 325  
PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL, [97](#), 201, 204, 324  
PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT, [97](#), 201, 204, 324  
PMIX\_JOB\_CTRL\_ID, [96](#), [96](#), 200, 201, 203, 204, 324  
PMIX\_JOB\_CTRL\_KILL, [96](#), 200, 203, 324  
PMIX\_JOB\_CTRL\_PAUSE, [96](#), 200, 203, 324  
PMIX\_JOB\_CTRL\_PREEMPTIBLE, [97](#), 201, 204, 325

PMIX\_JOB\_CTRL\_PROVISION, [97](#), [201](#), [204](#), [325](#)  
PMIX\_JOB\_CTRL\_PROVISION\_IMAGE, [97](#), [201](#), [204](#), [325](#)  
PMIX\_JOB\_CTRL\_RESTART, [97](#), [201](#), [204](#), [324](#)  
PMIX\_JOB\_CTRL\_RESUME, [96](#), [200](#), [203](#), [324](#)  
PMIX\_JOB\_CTRL\_SIGNAL, [97](#), [200](#), [203](#), [324](#)  
PMIX\_JOB\_CTRL\_TERMINATE, [97](#), [200](#), [203](#), [324](#)  
PMIX\_JOB\_INFO, [80](#), [128](#), [132](#), [135](#), [182](#), [187](#)  
PMIX\_JOB\_INFO\_ARRAY, [8](#), [81](#), [82](#), [252](#), [262](#)  
PMIX\_JOB\_NUM\_APPS, [82](#), [135](#), [253](#), [262](#)  
PMIX\_JOB\_RECOVERABLE, [90](#), [159](#), [164](#), [302](#)  
PMIX\_JOB\_SIZE, [8](#), [10](#), [82](#), [135](#), [253](#), [262](#)  
PMIX\_JOB\_TERM\_STATUS, [85](#)  
PMIX\_JOBID, [25](#), [79](#), [80](#), [81](#), [129](#), [132](#), [136](#), [182](#), [187](#), [253](#), [262](#)  
PMIX\_LAUNCHER, [390](#), [394](#), [396](#)  
PMIX\_LAUNCHER\_DAEMON, [402](#), [403](#)  
PMIX\_LAUNCHER\_RENDEZVOUS\_FILE, [391](#), [395](#)  
PMIX\_LOCAL\_CPUSETS, [80](#), [255](#), [265](#)  
PMIX\_LOCAL\_PEERS, [80](#), [255](#), [265](#)  
PMIX\_LOCAL\_PROCS, [80](#), [256](#)  
PMIX\_LOCAL\_RANK, [79](#), [182](#), [183](#), [187](#), [188](#), [257](#), [412](#)  
PMIX\_LOCAL\_SIZE, [82](#), [255](#)  
PMIX\_LOCAL\_TOPO (Deprecated), [84](#)  
PMIX\_LOCALITY (Deprecated), [178](#)  
PMIX\_LOCALITY\_STRING, [178](#), [257](#), [281](#)  
PMIX\_LOCALLDR, [79](#), [255](#)  
PMIX\_LOG\_EMAIL, [93](#), [211](#), [214](#), [319](#)  
PMIX\_LOG\_EMAIL\_ADDR, [93](#), [211](#), [214](#), [319](#)  
PMIX\_LOG\_EMAIL\_MSG, [93](#), [211](#), [214](#), [319](#)  
PMIX\_LOG\_EMAIL\_SENDER\_ADDR, [93](#)  
PMIX\_LOG\_EMAIL\_SERVER, [93](#)  
PMIX\_LOG\_EMAIL\_SRVR\_PORT, [93](#)  
PMIX\_LOG\_EMAIL\_SUBJECT, [93](#), [211](#), [214](#), [319](#)  
PMIX\_LOG\_GENERATE\_TIMESTAMP, [93](#), [211](#), [214](#)  
PMIX\_LOG\_GLOBAL\_DATASTORE, [93](#), [212](#), [214](#)  
PMIX\_LOG\_GLOBAL\_SYSLOG, [93](#), [211](#), [213](#)  
PMIX\_LOG\_JOB\_EVENTS, [90](#), [398](#)  
PMIX\_LOG\_JOB\_RECORD, [93](#), [211](#), [214](#)  
PMIX\_LOG\_LOCAL\_SYSLOG, [93](#), [211](#), [213](#)  
PMIX\_LOG\_MSG, [93](#), [319](#)  
PMIX\_LOG\_ONCE, [93](#), [211](#), [213](#)  
PMIX\_LOG\_SOURCE, [92](#), [211](#), [214](#)  
PMIX\_LOG\_STDERR, [92](#), [210](#), [213](#), [318](#)  
PMIX\_LOG\_STDOUT, [92](#), [210](#), [213](#), [318](#)

PMIX\_LOG\_SYSLOG, [93](#), [210](#), [213](#), [318](#)  
PMIX\_LOG\_SYSLOG\_PRI, [93](#), [211](#), [213](#)  
PMIX\_LOG\_TAG\_OUTPUT, [93](#), [211](#), [214](#)  
PMIX\_LOG\_TIMESTAMP, [93](#), [211](#), [214](#)  
PMIX\_LOG\_TIMESTAMP\_OUTPUT, [93](#), [211](#), [214](#)  
PMIX\_LOG\_XML\_OUTPUT, [93](#), [211](#), [214](#)  
PMIX\_MAP\_BLOB, [86](#)  
PMIX\_MAPBY, [88](#), [157](#), [163](#), [253](#), [301](#)  
PMIX\_MAPPER, [88](#), [88](#), [157](#), [163](#), [301](#)  
PMIX\_MAX\_PROCS, [10](#), [62](#), [82](#), [83](#), [83](#), [137](#), [252–254](#), [256](#)  
PMIX\_MAX\_RESTARTS, [90](#), [159](#), [164](#), [303](#)  
PMIX\_MAX\_VALUE, [11](#), [62](#), [99](#)  
PMIX\_MERGE\_STDERR\_STDOUT, [89](#), [158](#), [164](#), [302](#)  
PMIX\_MIN\_VALUE, [11](#), [62](#), [99](#)  
PMIX\_MODEL\_AFFINITY\_POLICY, [77](#)  
PMIX\_MODEL\_CPU\_TYPE, [76](#)  
PMIX\_MODEL\_LIBRARY\_NAME, [76](#), [255](#), [274](#)  
PMIX\_MODEL\_LIBRARY\_VERSION, [76](#), [255](#), [274](#)  
PMIX\_MODEL\_NUM\_CPUS, [76](#)  
PMIX\_MODEL\_NUM\_THREADS, [76](#)  
PMIX\_MODEL\_PHASE\_NAME, [76](#)  
PMIX\_MODEL\_PHASE\_TYPE, [76](#)  
PMIX\_MONITOR\_APP\_CONTROL, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_CANCEL, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_FILE, [98](#), [206–208](#), [327](#)  
PMIX\_MONITOR\_FILE\_ACCESS, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_FILE\_CHECK\_TIME, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_FILE\_DROPS, [98](#), [206](#), [209](#), [327](#)  
PMIX\_MONITOR\_FILE MODIFY, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_FILE\_SIZE, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_HEARTBEAT, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_HEARTBEAT\_DROPS, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_HEARTBEAT\_TIME, [98](#), [206](#), [208](#), [327](#)  
PMIX\_MONITOR\_ID, [98](#), [206](#), [208](#), [327](#)  
PMIX\_NO\_OVERSUBSCRIBE, [89](#), [158](#), [164](#), [302](#)  
PMIX\_NO\_PROCS\_ON\_HEAD, [89](#), [158](#), [164](#), [302](#)  
PMIX\_NODE\_INFO, [81](#), [129](#), [132](#), [137](#), [182](#), [187](#), [256](#)  
PMIX\_NODE\_INFO\_ARRAY, [81](#), [82](#), [255](#), [262](#), [264](#)  
PMIX\_NODE\_LIST, [79](#)  
PMIX\_NODE\_MAP, [10](#), [86](#), [253](#), [261–263](#), [274](#)  
PMIX\_NODE\_RANK, [79](#), [257](#)  
PMIX\_NODE\_SIZE, [83](#), [137](#), [255](#)  
PMIX\_NODEID, [80](#)–[82](#), [129](#), [132](#), [137](#), [182](#), [187](#), [255](#), [257](#), [347](#), [350](#), [356](#), [357](#), [359](#)

PMIX\_NOHUP, [398](#), [401](#), [403](#)  
PMIX\_NON\_PMI, [89](#), [158](#), [163](#), [301](#)  
PMIX\_NOTIFY\_COMPLETION, [90](#), [398](#)  
PMIX\_NOTIFY\_JOB\_EVENTS, [90](#), [398](#)  
PMIX\_NOTIFY\_LAUNCH, [85](#)  
PMIX\_NPROC\_OFFSET, [79](#), [253](#)  
PMIX\_NSDIR, [78](#), [78](#), [256](#), [257](#)  
PMIX\_NSPACE, [25](#), [79](#), [80–82](#), [91](#), [92](#), [129](#), [132](#), [136](#), [182–185](#), [187–190](#), [253](#), [262](#), [314](#), [315](#), [410](#)  
PMIX\_NUM\_NODES, [83](#), [134–136](#), [261](#), [262](#)  
PMIX\_NUM\_SLOTS, [83](#)  
PMIX\_OPTIONAL, [85](#), [128](#), [130](#), [131](#)  
PMIX\_OUTPUT\_TO\_DIRECTORY, [89](#)  
PMIX\_OUTPUT\_TO\_FILE, [89](#), [158](#), [164](#), [302](#)  
PMIX\_PACKAGE\_RANK, [79](#)  
PMIX\_PARENT\_ID, [80](#), [156](#), [162](#), [300](#)  
PMIX\_PERSISTENCE, [85](#), [144](#), [146](#), [293](#)  
PMIX\_PERSONALITY, [88](#), [157](#), [163](#), [301](#)  
PMIX\_PPR, [88](#), [157](#), [163](#), [301](#)  
PMIX\_PREFIX, [88](#), [157](#), [162](#), [300](#)  
PMIX\_PRELOAD\_BIN, [89](#), [157](#), [163](#), [300](#)  
PMIX\_PRELOAD\_FILES, [89](#), [157](#), [163](#), [300](#)  
PMIX\_PREPEND\_ENVAR, [94](#), [159](#), [165](#)  
PMIX\_PRIMARY\_SERVER, [395](#), [419](#)  
PMIX\_PROC\_BLOB, [86](#)  
PMIX\_PROC\_DATA, [86](#)  
PMIX\_PROC\_INFO\_ARRAY, [81](#), [86](#), [256](#), [263](#)  
PMIX\_PROC\_MAP, [10](#), [86](#), [253](#), [261](#), [262](#), [274](#)  
PMIX\_PROC\_PID, [79](#)  
PMIX\_PROC\_STATE\_STATUS, [85](#)  
PMIX\_PROC\_TERM\_STATUS, [85](#)  
PMIX\_PROC\_URI, [80](#), [185](#), [190](#)  
PMIX\_PROCDIR, [78](#), [257](#)  
PMIX\_PROCID, [79](#), [81](#), [92](#), [182–184](#), [187–189](#), [315](#)  
PMIX\_PROGRAMMING\_MODEL, [76](#), [255](#), [274](#)  
PMIX\_PSET\_NAME, [76](#), [254](#), [361](#)  
PMIX\_QUERY\_ALLOC\_STATUS, [92](#), [185](#), [190](#), [315](#)  
PMIX\_QUERY\_ATTRIBUTE\_SUPPORT, [92](#), [182](#), [187](#), [191](#), [410](#)  
PMIX\_QUERY\_AUTHORIZATIONS, [91](#)  
PMIX\_QUERY\_AVAIL\_SERVERS, [92](#)  
PMIX\_QUERY\_DEBUG\_SUPPORT, [92](#), [184](#), [189](#), [315](#)  
PMIX\_QUERY\_JOB\_STATUS, [91](#), [184](#), [189](#), [314](#)  
PMIX\_QUERY\_LOCAL\_ONLY, [92](#), [315](#)  
PMIX\_QUERY\_LOCAL\_PROC\_TABLE, [91](#), [184](#), [189](#), [315](#), [410](#)

PMIX\_QUERY\_MEMORY\_USAGE, [92](#), [184](#), [189](#), [315](#)  
PMIX\_QUERY\_NAMESPACE\_INFO, [91](#)  
PMIX\_QUERY\_NAMESPACES, [91](#), [184](#), [189](#), [314](#)  
PMIX\_QUERY\_NUM\_PSETS, [92](#)  
PMIX\_QUERY\_PROC\_TABLE, [91](#), [184](#), [189](#), [315](#), [410](#)  
PMIX\_QUERY\_PSET\_NAMES, [92](#)  
PMIX\_QUERY\_QUEUE\_LIST, [91](#), [184](#), [189](#), [314](#)  
PMIX\_QUERY\_QUEUE\_STATUS, [91](#), [184](#), [189](#), [314](#)  
PMIX\_QUERY\_REFRESH\_CACHE, [91](#), [181](#), [185](#), [186](#), [190](#)  
PMIX\_QUERY\_REPORT\_AVG, [92](#), [184](#), [189](#), [315](#)  
PMIX\_QUERY\_REPORT\_MINMAX, [92](#), [185](#), [189](#), [315](#)  
PMIX\_QUERY\_SPAWN\_SUPPORT, [91](#), [184](#), [189](#), [315](#)  
PMIX\_QUERY\_STORAGE\_LIST, [101](#)  
PMIX\_QUERY\_SUPPORTED\_KEYS, [91](#)  
PMIX\_QUERY\_SUPPORTED\_QUALIFIERS, [91](#)  
PMIX\_RANGE, [85](#), [144](#), [146](#), [147](#), [150](#), [151](#), [153](#), [207](#), [220](#), [293](#), [295](#), [298](#), [312](#), [337](#)  
PMIX\_RANK, [79](#), [81](#), [92](#), [182](#)–[184](#), [187](#)–[189](#), [256](#), [315](#), [412](#), [414](#)  
PMIX\_RANKBY, [89](#), [158](#), [163](#), [253](#), [301](#)  
PMIX\_RECONNECT\_SERVER (**Deprecated**), [395](#)  
PMIX\_REGISTER\_CLEANUP, [97](#), [200](#), [203](#)  
PMIX\_REGISTER\_CLEANUP\_DIR, [97](#), [200](#), [203](#)  
PMIX\_REGISTER\_NODATA, [86](#), [251](#)  
PMIX\_REINCARNATION, [76](#), [257](#)  
PMIX\_REPORT\_BINDINGS, [90](#), [158](#), [164](#), [302](#)  
PMIX\_REQUESTOR\_IS\_CLIENT, [76](#), [156](#), [162](#)  
PMIX\_REQUESTOR\_IS\_TOOL, [76](#), [156](#), [162](#)  
PMIX\_REQUIRED\_KEY, [86](#)  
PMIX\_RM\_NAME, [94](#)  
PMIX\_RM\_VERSION, [94](#)  
PMIX\_SEND\_HEARTBEAT, [98](#)  
PMIX\_SERVER\_ATTRIBUTES, [11](#), [99](#), [183](#), [188](#)  
PMIX\_SERVER\_ENABLE\_MONITORING, [75](#)  
PMIX\_SERVER\_FUNCTIONS, [92](#), [99](#), [182](#), [187](#)  
PMIX\_SERVER\_GATEWAY, [75](#)  
PMIX\_SERVER\_HOSTNAME, [395](#)  
PMIX\_SERVER\_INFO\_ARRAY, [82](#)  
PMIX\_SERVER\_NSPACE, [75](#), [246](#), [252](#), [392](#), [415](#), [419](#)  
PMIX\_SERVER\_PIDINFO, [392](#), [395](#), [415](#), [419](#)  
PMIX\_SERVER\_RANK, [75](#), [246](#), [252](#)  
PMIX\_SERVER\_REMOTE\_CONNECTIONS, [75](#), [247](#)  
PMIX\_SERVER\_SCHEDULER, [348](#), [350](#)  
PMIX\_SERVER\_SESSION\_SUPPORT, [75](#)  
PMIX\_SERVER\_START\_TIME, [75](#)

PMIX\_SERVER\_SYSTEM\_SUPPORT, [75](#), [246](#), [390](#)  
PMIX\_SERVER\_TMPDIR, [75](#), [246](#), [390–392](#)  
PMIX\_SERVER\_TOOL\_SUPPORT, [75](#), [236](#), [246](#), [248](#)  
PMIX\_SERVER\_URI, [185](#), [190](#), [392](#), [393](#), [395](#), [415](#), [419](#)  
PMIX\_SESSION\_ID, [25](#), [79](#), [80](#), [81](#), [128](#), [132](#), [134](#), [182](#), [187](#), [252](#), [261](#), [262](#)  
PMIX\_SESSION\_INFO, [80](#), [128](#), [132](#), [134](#), [181](#), [186](#), [252](#), [253](#)  
PMIX\_SESSION\_INFO\_ARRAY, [8](#), [81](#), [82](#), [252](#), [261](#)  
PMIX\_SET\_ENVAR, [94](#), [159](#), [164](#)  
PMIX\_SET\_SESSION\_CWD, [89](#), [157](#), [162](#), [300](#)  
PMIX\_SETUP\_APP\_ALL, [99](#), [272](#)  
PMIX\_SETUP\_APP\_ENVARS, [99](#), [272](#)  
PMIX\_SETUP\_APP\_NONENVARS, [99](#), [272](#)  
PMIX\_SINGLE\_LISTENER, [77](#), [122](#)  
PMIX\_SOCKET\_MODE, [77](#), [122](#), [247](#), [416](#)  
PMIX\_SPAWN\_TOOL, [90](#)  
PMIX\_SPAWNED, [78](#), [156](#), [162](#), [257](#), [300](#)  
PMIX\_STDIN\_TGT, [89](#), [158](#), [163](#), [301](#)  
PMIX\_STORAGE\_AVAIL\_BW, [101](#)  
PMIX\_STORAGE\_BW, [101](#)  
PMIX\_STORAGE\_CAPACITY\_AVAIL, [101](#)  
PMIX\_STORAGE\_CAPACITY\_FREE, [101](#)  
PMIX\_STORAGE\_CAPACITY\_LIMIT, [101](#)  
PMIX\_STORAGE\_ID, [100](#)  
PMIX\_STORAGE\_OBJECT\_LIMIT, [101](#)  
PMIX\_STORAGE\_OBJECTS\_AVAIL, [101](#)  
PMIX\_STORAGE\_OBJECTS\_FREE, [101](#)  
PMIX\_STORAGE\_PATH, [101](#)  
PMIX\_STORAGE\_TYPE, [101](#)  
PMIX\_SWITCH\_PEERS, [349](#)  
PMIX\_SYSTEM\_TMPDIR, [75](#), [246](#), [390](#), [392](#)  
PMIX\_TAG\_OUTPUT, [89](#), [158](#), [164](#), [302](#)  
PMIX\_TCP\_DISABLE\_IPV4, [77](#), [123](#), [247](#), [416](#)  
PMIX\_TCP\_DISABLE\_IPV6, [77](#), [123](#), [247](#), [416](#)  
PMIX\_TCP\_IF\_EXCLUDE, [77](#), [122](#), [247](#), [416](#)  
PMIX\_TCP\_IF\_INCLUDE, [77](#), [122](#), [247](#), [416](#)  
PMIX\_TCP\_IPV4\_PORT, [77](#), [123](#), [247](#), [416](#)  
PMIX\_TCP\_IPV6\_PORT, [77](#), [123](#), [247](#), [416](#)  
PMIX\_TCP\_REPORT\_URI, [77](#), [122](#), [247](#), [416](#)  
PMIX\_TCP\_URI, [77](#), [392](#), [393](#), [415](#), [419](#)  
PMIX\_TDIR\_RMCLEAN, [78](#)  
PMIX\_THREADING\_MODEL, [76](#)  
PMIX\_TIME\_REMAINING, [92](#), [179](#), [185](#), [190](#), [315](#)  
PMIX\_TIMEOUT, [4](#), [13](#), [84](#), [129](#), [130](#), [132](#), [133](#), [139](#), [140](#), [142](#), [144](#), [146–148](#), [150](#), [151](#), [153](#), [168](#),

169, 171, 172, 174, 175, 238, 240, 242, 244, 289, 292, 293, 296, 298, 303, 304, 307, 329,  
331, 364, 366, 368, 371–373, 375, 377, 381, 383, 385, 386  
**PMIX\_TIMEOUT\_REPORT\_STATE**, [90](#)  
**PMIX\_TIMEOUT\_STACKTRACES**, [90](#)  
**PMIX\_TIMESTAMP\_OUTPUT**, [89](#), [158](#), [164](#), [302](#)  
**PMIX\_TMPDIR**, [78](#), [78](#), [256](#)  
**PMIX\_TOOL\_ATTACHMENT\_FILE**, [392](#), [393](#), [395](#), [415](#), [419](#)  
**PMIX\_TOOL\_ATTRIBUTES**, [11](#), [99](#), [183](#), [188](#)  
**PMIX\_TOOL\_CONNECT\_OPTIONAL**, [395](#)  
**PMIX\_TOOL\_DO\_NOT\_CONNECT**, [392](#), [395](#), [415](#), [417](#)  
**PMIX\_TOOL\_FUNCTIONS**, [92](#), [99](#), [182](#), [187](#)  
**PMIX\_TOOL\_NSPACE**, [391](#), [394](#), [415](#), [417](#)  
**PMIX\_TOOL\_RANK**, [391](#), [394](#), [415](#), [417](#)  
**PMIX\_TOPOLOGY (Deprecated)**, [245](#)  
**PMIX\_TOPOLOGY2**, [245](#), [248](#)  
**PMIX\_TOPOLOGY\_FILE (Deprecated)**, [84](#)  
**PMIX\_TOPOLOGY\_SIGNATURE (Deprecated)**, [84](#)  
**PMIX\_TOPOLOGY\_XML (Deprecated)**, [84](#)  
**PMIX\_UNIV\_SIZE**, [8](#), [10](#), [82](#), [134](#), [252](#), [261](#)  
**PMIX\_UNSET\_ENVAR**, [94](#), [159](#), [164](#)  
**PMIX\_USERID**, [76](#), [114](#), [143](#), [145](#), [147](#), [149](#), [151](#), [153](#), [183](#), [188](#), [193](#), [196](#), [200](#), [202](#), [206](#), [208](#),  
    [210](#), [213](#), [238](#), [239](#), [241](#), [243](#), [293–298](#), [300](#), [308](#), [310](#), [314](#), [316](#), [318](#), [320](#), [323](#), [326](#), [328](#), [330](#),  
    [332](#), [335](#)  
**PMIX\_USOCK\_DISABLE**, [77](#), [122](#), [247](#)  
**PMIX\_VERSION\_INFO**, [76](#)  
**PMIX\_WAIT**, [84](#), [147](#), [148](#), [150](#), [295](#)  
**PMIX\_WDIR**, [88](#), [156](#), [162](#), [254](#), [300](#)