



Process Management Interface for Exascale (PMIx) Standard

Version 4.0 (Draft)

Created on June 22, 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

1. Introduction	1
1.1. Charter	2
1.2. PMIx Standard Overview	2
1.2.1. Who should use the standard?	2
1.2.2. What is defined in the standard?	3
1.2.3. What is <i>not</i> defined in the standard?	3
1.2.4. General Guidance for PMIx Users and Implementors	4
1.3. PMIx Architecture Overview	4
1.3.1. The PMIx Reference Implementation (PRI)	6
1.3.2. The PMIx Reference RunTime Environment (PRRTE)	7
1.3.3. PMIx Roles	7
1.4. Organization of this document	8
1.5. Version 1.0: June 12, 2015	9
1.6. Version 2.0: Sept. 2018	9
1.7. Version 2.1: Dec. 2018	10
1.8. Version 2.2: Jan 2019	11
1.9. Version 3.0: Dec. 2018	11
1.10. Version 3.1: Jan. 2019	12
1.11. Version 3.2: Oct. 2019	13
1.12. Version 4.0: June 2019	13
2. PMIx Terms and Conventions	15
2.1. Notational Conventions	17
2.2. Semantics	18
2.3. Naming Conventions	19
2.4. Procedure Conventions	19
2.5. Standard vs Reference Implementation	20

3. General Information Interfaces	21
3.1. Initialization Status	21
3.1.1. PMIx_Initialized	21
3.2. Library Information	21
3.2.1. PMIx_Get_version	21
4. Client-Specific Interfaces	23
4.1. Client Initialization and Finalization	23
4.1.1. PMIx_Init	23
4.1.2. PMIx_Finalize	25
4.2. Tool Initialization and Finalization	26
4.2.1. PMIx_tool_init	27
4.2.2. PMIx_tool_finalize	30
4.2.3. PMIx_tool_connect_to_server	31
5. Key/Value Management	33
5.1. Setting and Accessing Key/Value Pairs	33
5.1.1. PMIx_Put	33
5.1.2. PMIx_Get	34
5.1.3. PMIx_Get_nb	37
5.1.4. PMIx_Store_internal	40
5.1.5. Accessing information: examples	41
5.2. Exchanging Key/Value Pairs	45
5.2.1. PMIx_Commit	46
5.2.2. PMIx_Fence	46
5.2.3. PMIx_Fence_nb	48
5.3. Publish and Lookup Data	51
5.3.1. PMIx_Publish	51
5.3.2. PMIx_Publish_nb	53
5.3.3. PMIx_Lookup	54
5.3.4. PMIx_Lookup_nb	57
5.3.5. PMIx_Unpublish	58
5.3.6. PMIx_Unpublish_nb	60

6. Process Management	62
6.1. Abort	62
6.1.1. PMIx_Abort	62
6.2. Process Creation	63
6.2.1. PMIx_Spawn	63
6.2.2. PMIx_Spawn_nb	68
6.3. Connecting and Disconnecting Processes	72
6.3.1. PMIx_Connect	73
6.3.2. PMIx_Connect_nb	75
6.3.3. PMIx_Disconnect	77
6.3.4. PMIx_Disconnect_nb	79
6.4. IO Forwarding	81
6.4.1. PMIx_IOF_pull	82
6.4.2. PMIx_IOF_deregister	84
6.4.3. PMIx_IOF_push	85
7. Job Management and Reporting	88
7.1. Query	88
7.1.1. PMIx_Resolve_peers	88
7.1.2. PMIx_Resolve_nodes	89
7.1.3. PMIx_Query_info	90
7.1.4. PMIx_Query_info_nb	94
7.2. Allocation Requests	100
7.2.1. PMIx_Allocation_request	101
7.2.2. PMIx_Allocation_request_nb	104
7.3. Job Control	107
7.3.1. PMIx_Job_control	107
7.3.2. PMIx_Job_control_nb	110
7.4. Process and Job Monitoring	113
7.4.1. PMIx_Process_monitor	113
7.4.2. PMIx_Process_monitor_nb	115
7.4.3. PMIx_Heartbeat	117
7.5. Logging	118
7.5.1. PMIx_Log	118

7.5.2.	PMIx_Log_nb	120
8.	Event Notification	124
8.1.	Notification and Management	124
8.1.1.	PMIx_Register_event_handler	126
8.1.2.	PMIx_Deregister_event_handler	129
8.1.3.	PMIx_Notify_event	130
9.	Data Packing and Unpacking	134
9.1.	Data Buffer Type	134
9.2.	Support Macros	135
9.2.1.	PMIX_DATA_BUFFER_CREATE	135
9.2.2.	PMIX_DATA_BUFFER_RELEASE	135
9.2.3.	PMIX_DATA_BUFFER_CONSTRUCT	135
9.2.4.	PMIX_DATA_BUFFER_DESTRUCT	136
9.2.5.	PMIX_DATA_BUFFER_LOAD	136
9.2.6.	PMIX_DATA_BUFFER_UNLOAD	137
9.3.	General Routines	137
9.3.1.	PMIx_Data_pack	137
9.3.2.	PMIx_Data_unpack	139
9.3.3.	PMIx_Data_copy	141
9.3.4.	PMIx_Data_print	141
9.3.5.	PMIx_Data_copy_payload	142
10.	Security	144
10.1.	Obtaining Credentials	145
10.1.1.	PMIx_Get_credential	145
10.1.2.	PMIx_Get_credential_nb	146
10.2.	Validating Credentials	148
10.2.1.	PMIx_Validate_credential	148
10.2.2.	PMIx_Validate_credential_nb	150
11.	Server-Specific Interfaces	153
11.1.	Server Initialization and Finalization	153
11.1.1.	PMIx_server_init	153

11.1.2.	PMIx_server_finalize	156
11.2.	Server Support Functions	156
11.2.1.	PMIx_generate_regex	157
11.2.2.	PMIx_generate_ppn	158
11.2.3.	PMIx_server_register_namespace	158
11.2.4.	PMIx_server_deregister_namespace	172
11.2.5.	PMIx_server_register_client	173
11.2.6.	PMIx_server_deregister_client	174
11.2.7.	PMIx_server_setup_fork	175
11.2.8.	PMIx_server_dmodex_request	175
11.2.9.	PMIx_server_setup_application	177
11.2.10.	PMIx_Register_attributes	179
11.2.11.	PMIx_server_setup_local_support	181
11.2.12.	PMIx_server_IOF_deliver	182
11.2.13.	PMIx_server_collect_inventory	183
11.2.14.	PMIx_server_deliver_inventory	184
11.3.	Server Function Pointers	186
11.3.1.	pmix_server_module_t Module	186
11.3.2.	pmix_server_client_connected_fn_t	187
11.3.3.	pmix_server_client_finalized_fn_t	189
11.3.4.	pmix_server_abort_fn_t	190
11.3.5.	pmix_server_fence_nb_fn_t	191
11.3.6.	pmix_server_dmodex_req_fn_t	195
11.3.7.	pmix_server_publish_fn_t	196
11.3.8.	pmix_server_lookup_fn_t	198
11.3.9.	pmix_server_unpublish_fn_t	200
11.3.10.	pmix_server_spawn_fn_t	202
11.3.11.	pmix_server_connect_fn_t	207
11.3.12.	pmix_server_disconnect_fn_t	209
11.3.13.	pmix_server_register_events_fn_t	211
11.3.14.	pmix_server_deregister_events_fn_t	213
11.3.15.	pmix_server_notify_event_fn_t	215
11.3.16.	pmix_server_listener_fn_t	216

11.3.17.	<code>pmix_server_query_fn_t</code>	217
11.3.18.	<code>pmix_server_tool_connection_fn_t</code>	220
11.3.19.	<code>pmix_server_log_fn_t</code>	221
11.3.20.	<code>pmix_server_alloc_fn_t</code>	223
11.3.21.	<code>pmix_server_job_control_fn_t</code>	225
11.3.22.	<code>pmix_server_monitor_fn_t</code>	228
11.3.23.	<code>pmix_server_get_cred_fn_t</code>	231
11.3.24.	<code>pmix_server_validate_cred_fn_t</code>	232
11.3.25.	<code>pmix_server_iof_fn_t</code>	234
11.3.26.	<code>pmix_server_stdin_fn_t</code>	237
11.3.27.	<code>pmix_server_grp_fn_t</code>	238
11.3.28.	<code>pmix_server_fabric_fn_t</code>	240
12.	Fabric Support Definitions	243
12.1.	Fabric Support Constants	243
12.2.	Fabric Support Datatypes	243
12.2.1.	Fabric Coordinate Structure	244
12.2.2.	Fabric Coordinate Support Macros	245
12.2.3.	Fabric Coordinate Views	246
12.2.4.	Fabric Link State	246
12.2.5.	Fabric Operation Constants	247
12.2.6.	Fabric registration structure	247
12.3.	Fabric Support Attributes	250
12.4.	Fabric Support Functions	253
12.4.1.	<code>PMIx_Fabric_register</code>	253
12.4.2.	<code>PMIx_Fabric_update</code>	255
12.4.3.	<code>PMIx_Fabric_deregister</code>	255
12.4.4.	<code>PMIx_Fabric_get_vertex_info</code>	255
12.4.5.	<code>PMIx_Fabric_get_index</code>	258
13.	Process Sets and Groups	259
13.1.	Process Sets	259
13.2.	Process Groups	260
13.2.1.	Group Operation Constants	262

13.2.2.	PMIx_Group_construct	263
13.2.3.	PMIx_Group_construct_nb	266
13.2.4.	PMIx_Group_destruct	269
13.2.5.	PMIx_Group_destruct_nb	271
13.2.6.	PMIx_Group_invite	273
13.2.7.	PMIx_Group_invite_nb	277
13.2.8.	PMIx_Group_join	279
13.2.9.	PMIx_Group_join_nb	282
13.2.10.	PMIx_Group_leave	284
13.2.11.	PMIx_Group_leave_nb	285
14. Data Structures and Types		287
14.1.	Constants	288
14.1.1.	PMIx Error Constants	289
14.1.2.	Macros for use with PMIx constants	293
14.2.	Data Types	293
14.2.1.	Key Structure	294
14.2.2.	Namespace Structure	295
14.2.3.	Rank Structure	296
14.2.4.	Process Structure	296
14.2.5.	Process structure support macros	296
14.2.6.	Process State Structure	298
14.2.7.	Process Information Structure	299
14.2.8.	Process Information Structure support macros	300
14.2.9.	Scope of Put Data	301
14.2.10.	Job State Structure	302
14.2.11.	Range of Published Data	302
14.2.12.	Data Persistence Structure	303
14.2.13.	Data Array Structure	303
14.2.14.	Data array structure support macros	303
14.2.15.	Value Structure	305
14.2.16.	Value structure support macros	306
14.2.17.	Info Structure	310
14.2.18.	Info structure support macros	310

14.2.19.	Info Type Directives	313
14.2.20.	Info Directive support macros	313
14.2.21.	Job Allocation Directives	315
14.2.22.	IO Forwarding Channels	316
14.2.23.	Environmental Variable Structure	316
14.2.24.	Environmental variable support macros	316
14.2.25.	Lookup Returned Data Structure	318
14.2.26.	Lookup data structure support macros	318
14.2.27.	Application Structure	321
14.2.28.	App structure support macros	322
14.2.29.	Query Structure	323
14.2.30.	Query structure support macros	323
14.2.31.	Attribute registration structure	325
14.2.32.	Attribute registration structure support macros	326
14.2.33.	PMIx Group Directives	328
14.2.34.	Byte Object Type	328
14.2.35.	Byte object support macros	328
14.2.36.	Data Array Structure	330
14.2.37.	Data array support macros	330
14.2.38.	Argument Array Macros	331
14.2.39.	Set Environment Variable	335
14.3.	Generalized Data Types Used for Packing/Unpacking	336
14.4.	Reserved attributes	337
14.4.1.	Initialization attributes	338
14.4.2.	Tool-related attributes	338
14.4.3.	Identification attributes	339
14.4.4.	Programming model attributes	340
14.4.5.	UNIX socket rendezvous socket attributes	340
14.4.6.	TCP connection attributes	341
14.4.7.	Global Data Storage (GDS) attributes	341
14.4.8.	General process-level attributes	341
14.4.9.	Scratch directory attributes	342
14.4.10.	Relative Rank Descriptive Attributes	342

14.4.11. Information retrieval attributes	344
14.4.12. Information storage attributes	344
14.4.13. Size information attributes	345
14.4.14. Memory information attributes	346
14.4.15. Topology information attributes	346
14.4.16. Request-related attributes	347
14.4.17. Server-to-PMIx library attributes	349
14.4.18. Server-to-Client attributes	349
14.4.19. Event handler registration and notification attributes	349
14.4.20. Fault tolerance attributes	350
14.4.21. Spawn attributes	351
14.4.22. Query attributes	353
14.4.23. Log attributes	354
14.4.24. Debugger attributes	356
14.4.25. Resource manager attributes	356
14.4.26. Environment variable attributes	357
14.4.27. Job Allocation attributes	357
14.4.28. Job control attributes	359
14.4.29. Monitoring attributes	360
14.4.30. Security attributes	361
14.4.31. IO Forwarding attributes	361
14.4.32. Application setup attributes	362
14.4.33. Attribute support level attributes	362
14.4.34. Descriptive attributes	362
14.4.35. Process group attributes	363
14.5. Callback Functions	363
14.5.1. Release Callback Function	364
14.5.2. Modex Callback Function	364
14.5.3. Spawn Callback Function	365
14.5.4. Op Callback Function	366
14.5.5. Lookup Callback Function	366
14.5.6. Value Callback Function	367
14.5.7. Info Callback Function	367

14.5.8.	Event Handler Registration Callback Function	368
14.5.9.	Notification Handler Completion Callback Function	369
14.5.10.	Notification Function	370
14.5.11.	Server Setup Application Callback Function	371
14.5.12.	Server Direct Modex Response Callback Function	372
14.5.13.	PMIx Client Connection Callback Function	373
14.5.14.	PMIx Tool Connection Callback Function	373
14.5.15.	Credential callback function	374
14.5.16.	Credential validation callback function	375
14.5.17.	IOF delivery function	376
14.5.18.	IOF and Event registration function	377
14.6.	Constant String Functions	378
A.	Python Bindings	382
A.1.	Design Considerations	382
A.1.1.	Error Codes vs Python Exceptions	382
A.1.2.	Representation of Structured Data	382
A.2.	Datatype Definitions	383
A.2.1.	Example	388
A.3.	Callback Function Definitions	388
A.3.1.	IOF Delivery Function	388
A.3.2.	Event Handler	389
A.3.3.	Server Module Functions	390
A.4.	PMIxClient	403
A.4.1.	Client.init	403
A.4.2.	Client.initialized	403
A.4.3.	Client.get_version	404
A.4.4.	Client.finalize	404
A.4.5.	Client.abort	404
A.4.6.	Client.store_internal	405
A.4.7.	Client.put	405
A.4.8.	Client.commit	406
A.4.9.	Client.fence	406
A.4.10.	Client.get	407

A.4.11.	Client.publish	407
A.4.12.	Client.lookup	408
A.4.13.	Client.unpublish	408
A.4.14.	Client.spawn	409
A.4.15.	Client.connect	409
A.4.16.	Client.disconnect	410
A.4.17.	Client.resolve_peers	410
A.4.18.	Client.resolve_nodes	411
A.4.19.	Client.query	411
A.4.20.	Client.log	412
A.4.21.	Client.allocate	412
A.4.22.	Client.job_ctrl	413
A.4.23.	Client.monitor	413
A.4.24.	Client.get_credential	414
A.4.25.	Client.validate_credential	414
A.4.26.	Client.group_construct	415
A.4.27.	Client.group_invite	415
A.4.28.	Client.group_join	416
A.4.29.	Client.group_leave	417
A.4.30.	Client.group_destruct	417
A.4.31.	Client.register_event_handler	417
A.4.32.	Client.deregister_event_handler	418
A.4.33.	Client.notify_event	418
A.4.34.	Client.fabric_register	419
A.4.35.	Client.fabric_update	419
A.4.36.	Client.fabric_deregister	420
A.4.37.	Client.fabric_get_vertex_info	420
A.4.38.	Client.fabric_get_index	421
A.4.39.	Client.error_string	421
A.4.40.	Client.proc_state_string	421
A.4.41.	Client.scope_string	422
A.4.42.	Client.persistence_string	422
A.4.43.	Client.data_range_string	423

A.4.44.	Client.info_directives_string	423
A.4.45.	Client.data_type_string	423
A.4.46.	Client.alloc_directive_string	424
A.4.47.	Client.iof_channel_string	424
A.4.48.	Client.job_state_string	425
A.4.49.	Client.get_attribute_string	425
A.4.50.	Client.get_attribute_name	425
A.4.51.	Client.link_state_string	426
A.5.	PMIxServer	426
A.5.1.	Server.init	426
A.5.2.	Server.finalize	427
A.5.3.	Server.generate_regex	427
A.5.4.	Server.generate_ppn	428
A.5.5.	Server.register_namespace	428
A.5.6.	Server.deregister_namespace	429
A.5.7.	Server.register_client	429
A.5.8.	Server.deregister_client	430
A.5.9.	Server.setup_fork	430
A.5.10.	Server.dmodex_request	431
A.5.11.	Server.setup_application	431
A.5.12.	Server.register_attributes	432
A.5.13.	Server.setup_local_support	432
A.5.14.	Server.iof_deliver	433
A.5.15.	Server.collect_inventory	433
A.5.16.	Server.deliver_inventory	434
A.6.	PMIxTool	434
A.6.1.	Tool.init	434
A.6.2.	Tool.finalize	435
A.6.3.	Tool.connect_to_server	435
A.6.4.	Tool.iof_pull	436
A.6.5.	Tool.iof_deregister	436
A.6.6.	Tool.iof_push	437

A.7. Example Usage	437
A.7.1. Python Client	438
A.7.2. Python Server	440
B. Acknowledgements	444
B.1. Version 3.0	444
B.2. Version 2.0	445
B.3. Version 1.0	446
Bibliography	447
Index	448
Index of APIs	449
Index of Support Macros	455
Index of Data Structures	458
Index of Constants	460
Index of Attributes	466

CHAPTER 1

Introduction

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

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

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

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

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

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

1.1 Charter

The charter of the PMIx community is to:

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

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

1.2 PMIx Standard Overview

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

1.2.1 Who should use the standard?

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

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

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

6 **1.2.2 What is defined in the standard?**

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

11 **1.2.3 What is *not* defined in the standard?**

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

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

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

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

32 It is therefore critical that users and application implementers:

- 33 a) consider whether or not a given attribute is required, marking it accordingly; and
- 34 b) check the return status on all PMIx function calls to ensure support was present and that the
35 request was accepted. Note that for non-blocking APIs, a return of [PMIX_SUCCESS](#) only
36 indicates that the request had no obvious errors and is being processed – the eventual callback
37 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. The PMIx
5 community is attempting to help differentiate the attributes by indicating those that are generally
6 used (and therefore, of higher importance to support) vs those that a “complete implementation”
7 would support.

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

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

19 **1.2.4 General Guidance for PMIx Users and Implementors**

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

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

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

36 **1.3 PMIx Architecture Overview**

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

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

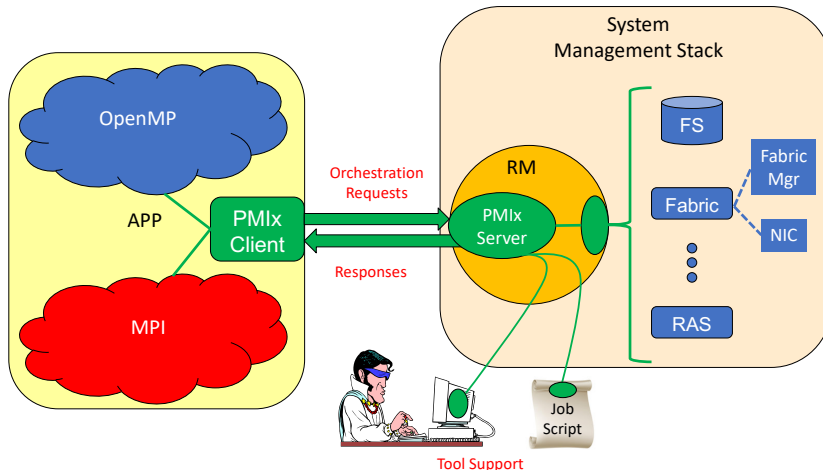


Figure 1.1.: PMIx-SMS Interactions

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

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

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

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

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

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

18 1.3.1 The PMIx Reference Implementation (PRI)

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

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

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

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

8 1.3.2 The PMIx Reference RunTime Environment (PRRTE)

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

18 1.3.3 PMIx Roles

19 The role of a PMIx process in the PMIx universe is grouped into one of three categories based on
20 how it operates in the PMIx environment namely as a *client*, *server*, or *tool*. As a result, there are
21 three corresponding sets of initialization and finalization functions. If a process initializes as either
22 a *server* or a *tool* that process may also access all of the *client* APIs.

23 A process operating as a *client* is started (directly or indirectly, for example, by an intermediate
24 script) by the RM and is connected to the PMIx server instance within that RM when the client
25 calls the client PMIx initialization routine. A process operating as a *server* is responsible for
26 starting client processes and coordinating with other server and tool processes in the same PMIx
27 universe. Often processes operating as a *server* are part of the resource manager (RM)
28 infrastructure. A process operating as a *tool* will connect to a PMIx *server* to interact with the
29 processes in the PMIx universe. An example of a *tool* process is a parallel debugger that will
30 connect to the server to assist with attaching to a set of client processes.

31 PMIx serves as a conduit between processes acting in these three different roles. As such, an API is
32 often described in how it interacts with processes operating in other roles in the PMIx universe.

▼ Advice to PMIx library implementers ▼

33 A PMIx implementation may support all or a subset of the API role groupings defined in the
34 standard. A common nomenclature is defined here to aid in identifying levels of conformance of an
35 implementation.

1 A PMIx implementation that supports all three sets of the API role groupings is said to be *fully*
2 *PMIx standard compliant*. These *fully PMIx standard compliant* implementations have the
3 advantage of being able to support a broad set of PMIx consumers in the different roles.

4 Alternatively, a PMIx implementation may choose to support fewer than all three sets of the API
5 role groupings. PMIx implementations that support only the *client* APIs are said to be *client-only*
6 *PMIx standard compliant*. Similarly, an implementation that only supports the *client* and *tool* APIs
7 are said to be *client-and-tool-only PMIx standard compliant*. Finally, an implementation that only
8 supports the *client* and *server* APIs are said to be *client-and-server-only PMIx standard compliant*.
9 Note that it would not make sense for an implementation to exclude the *client* interfaces from their
10 implementation since they are also used by the *server* and *tool* roles.

11 1.4 Organization of this document

12 The remainder of this document is structured as follows:

- 13 • Introduction and Overview in Chapter 1 on page 1
- 14 • Terms and Conventions in Chapter 2 on page 15
- 15 • Data Structures and Types in Chapter 14 on page 287
- 16 • PMIx Initialization and Finalization in Chapter 4 on page 23
- 17 • Key/Value Management in Chapter 5 on page 33
- 18 • Process Management in Chapter 6 on page 62
- 19 • Job Management in Chapter 7 on page 88
- 20 • Event Notification in Chapter 8 on page 124
- 21 • Data Packing and Unpacking in Chapter 9 on page 134
- 22 • Security in Chapter 10 on page 144
- 23 • PMIx Server Specific Interfaces in Chapter 11 on page 153
- 24 • Scheduler-Specific Interface in Chapter ?? on page ??
- 25 • Process Sets and Groups in Chapter 13 on page 259
- 26 • Network Coordinates in Chapter ?? on page ??
- 27 • Python Bindings in Appendix A on page 382

1 1.5 Version 1.0: June 12, 2015

2 The PMIx version 1.0 *ad hoc* standard was defined in the PMIx Reference Implementation (PRI)
3 header files as part of the PRI v1.0.0 release prior to the creation of the formal PMIx 2.0 standard.
4 Below are a summary 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**
 - 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_namespace** , **PMIx_server_deregister_namespace**
 - 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

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

13 1.8 Version 2.2: Jan 2019

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

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

27 1.9 Version 3.0: Dec. 2018

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

- 29 • Client APIs
- 30 – `PMIx_Log` , `PMIx_Job_control`

- 1 – `PMIx_Allocation_request` , `PMIx_Process_monitor`
- 2 – `PMIx_Get_credential` , `PMIx_Validate_credential`
- 3 • Server APIs
- 4 – `PMIx_server_IOF_deliver`
- 5 – `PMIx_server_collect_inventory` , `PMIx_server_deliver_inventory`
- 6 • Tool APIs
- 7 – `PMIx_IOF_pull` , `PMIx_IOF_push` , `PMIx_IOF_deregister`
- 8 – `PMIx_tool_connect_to_server`
- 9 • Common APIs
- 10 – `PMIx_IOF_channel_string`

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

15 1.10 Version 3.1: Jan. 2019

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

- 17 • Direct modex upcall function (`pmix_server_dmodex_req_fn_t`) cannot complete
 18 atomically as the API cannot return the requested information except via the provided callback
 19 function
- 20 • Fix typo in name of `PMIX_FWD_STDDIAG` attribute
- 21 • Correctly identify the information retrieval and storage attributes as “new” to v3 of the standard
- 22 • Add missing `pmix_data_array_t` definition and support macros
- 23 • Add a rule divider between implementer and host environment required attributes for clarity
- 24 • Add `PMIX_QUERY_QUALIFIERS_CREATE` macro to simplify creation of `pmix_query_t`
 25 qualifiers
- 26 • Add `PMIX_APP_INFO_CREATE` macro to simplify creation of `pmix_app_t` directives
- 27 • Add new attributes to specify the level of information being requested where ambiguity may exist
 28 (see 14.4.11)
- 29 • Add new attributes to assemble information by its level for storage where ambiguity may exist
 30 (see 14.4.12)

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

15 1.11 Version 3.2: Oct. 2019

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

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

24 1.12 Version 4.0: June 2019

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

- 26 • Clarified that the `PMIx_Fence_nb` operation can immediately return
27 `PMIX_OPERATION_SUCCEEDED` in lieu of passing the request to a PMIx server if only the
28 calling process is involved in the operation
- 29 • Added the `PMIx_Register_attributes` API by which a host environment can register
30 the attributes it supports for each server-to-host operation
- 31 • Added the ability to query supported attributes from the PMIx tool, client and server libraries, as
32 well as the host environment via the new `pmix_regattr_t` structure. Both human-readable
33 and machine-parsable output is supported. New attributes to support this operation include:

- 1 – `PMIX_CLIENT_ATTRIBUTES` , `PMIX_SERVER_ATTRIBUTES` ,
2 `PMIX_TOOL_ATTRIBUTES` , and `PMIX_HOST_ATTRIBUTES` to identify which library
3 supports the attribute; and
- 4 – `PMIX_MAX_VALUE` , `PMIX_MIN_VALUE` , and `PMIX_ENUM_VALUE` to provide
5 machine-parsable description of accepted values
- 6 • Add `PMIX_APP_WILDCARD` to reference all applications within a given job
- 7 • Fix signature of blocking APIs `PMIx_Allocation_request` , `PMIx_Job_control` ,
8 `PMIx_Process_monitor` , `PMIx_Get_credential` , and
9 `PMIx_Validate_credential` to allow return of results
- 10 • Update description to provide an option for blocking behavior of the
11 `PMIx_Register_event_handler` , `PMIx_Deregister_event_handler` ,
12 `PMIx_Notify_event` , `PMIx_IOF_pull` , `PMIx_IOF_deregister` , and
13 `PMIx_IOF_push` APIs. The need for blocking forms of these functions was not initially
14 anticipated but has emerged over time. For these functions, the return value is sufficient to
15 provide the caller with information otherwise returned via callback. Thus, use of a **NULL** value
16 as the callback function parameter was deemed a minimal disruption method for providing the
17 desired capability

CHAPTER 2

PMIx Terms and Conventions

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

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

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

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

Advice to users

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

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

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

- 10 • *session* refers to an allocated set of resources assigned to a particular user by the system WLM.
11 Historically, HPC sessions have consisted of a static allocation of resources - i.e., a block of
12 resources are assigned to a user in response to a specific request and managed as a unified
13 collection. However, this is changing in response to the growing use of dynamic programming
14 models that require on-the-fly allocation and release of system resources. Accordingly, the term
15 *session* in this document refers to the current block of assigned resources and is a potentially
16 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, “*mpirexec -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...


27 **Rationale**

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





Advice to users

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




Advice to PMIx library implementers

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



Advice to PMIx server hosts

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



11 2.2 Semantics

12 The following terms will be taken to mean:

- 13 • *shall*, *must* and *will* indicate that the specified behavior is *required* of all conforming
14 implementations
- 15 • *should* and *may* indicate behaviors that a complete implementation would include, but are not
16 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_namespace** .
- 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 PMIx Reference Implementation (PRI) is solely based on the C programming
17 language, it is not the intent of the PMIx Standard to preclude the use of other languages.
18 Accordingly, the procedure specifications in the PMIx Standard are written in a
19 language-independent syntax with the arguments marked as IN, OUT, or INOUT. The meanings of
20 these are:

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

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

32 A similar model is used for the server module functions (see 11.3.1). In this case, the PMIx library
33 is making an upcall into its host via the PMIx server module function and passing a specific cbfunc

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

6 2.5 Standard vs Reference Implementation

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

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

CHAPTER 3

General Information Interfaces

1 The APIs defined in this chapter can be used by any PMIx process, regardless of their role in the
2 PMIx universe.

3 3.1 Initialization Status

4 The APIs defined in this section return information about the status of the PMIx library.

5 3.1.1 `PMIx_Initialized`

6 **Format**

PMIx v1.0

7 `int PMIx_Initialized(void)`

C

C

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

9 **Rationale**

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

10 **Description**

11 Check to see if the PMIx library has been initialized using any of the initialization functions:
12 `PMIx_Init`, `PMIx_server_init`, or `PMIx_tool_init`. It is valid to call this API
13 outside of a region of initialization.

14 3.2 Library Information

15 The APIs defined in this section return information about the PMIx library.

16 3.2.1 `PMIx_Get_version`

17 **Summary**

18 Get the PMIx version information.

1
2
3
4
5

Format

PMIx v1.0

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

Description

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

CHAPTER 4

Client-Specific Interfaces

1 The APIs defined in this chapter are dedicated to PMIx consumers in the *client* role.

2 4.1 Client Initialization and Finalization

3 The PMIx APIs may only be used between the completion of the initialization function and the start
4 of the finalization function, unless otherwise noted. The initialization and finalization functions are
5 paired, and the initialized regions defined by them must not overlap.

6 4.1.1 PMIx_Init

7 Summary

8 Initialize the PMIx client library

9 Format

PMIx v1.2

C

```
10 pmix_status_t  
11 PMIx_Init (pmix_proc_t *proc,  
12           pmix_info_t info[], size_t ninfo)
```

C

13 INOUT *proc*

14 `pmix_proc_t` structure (handle)

15 IN *info*

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

17 IN *ninfo*

18 Number of element in the *info* array (`size_t`)

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

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)

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

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)

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

PMIX_SINGLE_LISTENER "pmix.sing.listnr" (bool)

Use only one rendezvous socket, letting priorities and/or environment parameters select the active transport. If the library supports multiple methods for clients to connect to servers, this attribute may be supported for disabling all but one of them.

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

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

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

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

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

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

PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)

The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)

The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)

Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)

Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)

1 Pointer to libevent¹ `event_base` to use in place of the internal progress thread.

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

3 Comma-delimited string of desired modules. This attribute is specific to the PRI and
4 controls only the selection of global data storage (GDS) module for internal use by the
5 process. Module selection for interacting with the server is performed dynamically during
6 the connection process.



7 **Description**

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

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

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

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

22 Each call to `PMIx_Init` may contain an array of `pmix_info_t` structures passing directives to
23 the PMIx client library as per the above attributes. Multiple calls to `PMIx_Init` shall not include
24 conflicting directives. The `PMIx_Init` function will return an error when directives that conflict
25 with prior directives are detected.

▼ **Advice to users** ▼

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



29 **4.1.2 PMIx_Finalize**

30 **Summary**

31 Finalize the PMIx client library.

¹<http://libevent.org/>

1
PMIx v1.0

Format

C

2 `pmix_status_t`
3 `PMIx_Finalize(const pmix_info_t info[], size_t ninfo)`

C

4 **IN** `info`
5 Array of `pmix_info_t` structures (array of handles)
6 **IN** `ninfo`
7 Number of element in the `info` array (`size_t`)

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

Optional Attributes

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

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

Description

14 Decrement the PMIx client library reference count. When the reference count reaches zero, the
15 library will finalize the PMIx client, closing the connection with the local PMIx server and
16 releasing internally allocated resources.
17

4.2 Tool Initialization and Finalization

18 The APIs defined in this chapter are dedicated to PMIx consumers in the *client* role.

19 **NOTE: THIS SECTION WILL MOVE TO THE NEW TOOLS CHAPTER WHEN**
20 **MERGED**

21 The PMIx APIs may only be used between the completion of the initialization function and the start
22 of the finalization function, unless otherwise noted. The initialization and finalization functions are
23 paired, and the initialized regions defined by them must not overlap.
24

Advice to users

25 Tool initialization automatically searches for a server to which it can connect. If the tool is declared
26 as a *launcher* (via `PMIX_LAUNCHER`), the PMIx library sets up the required “hooks” for other
27 tools (e.g., debuggers) to attach to it.

1 4.2.1 PMIx_tool_init

2 Summary

3 Initialize the PMIx library for operating as a tool.

4 Format

PMIx v2.0

C

5 `pmix_status_t`

6 `PMIx_tool_init` (`pmix_proc_t *proc,`
7 `pmix_info_t info[], size_t ninfo)`

C

8 **INOUT** `proc`

9 `pmix_proc_t` structure (handle)

10 **IN** `info`

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

12 **IN** `ninfo`

13 Number of element in the *info* array (`size_t`)

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

Required Attributes

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

16 **PMIX_TOOL_NAMESPACE** "`pmix.tool.namespace`" (`char*`)

17 Name of the namespace to use for this tool.

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

19 Rank of this tool.

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

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

22 **PMIX_SERVER_URI** "`pmix.srvr.uri`" (`char*`)

23 URI of the PMIx server to be contacted.

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_CONNECT_TO_SYSTEM "pmix.cnct.sys" (bool)

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

PMIX_CONNECT_SYSTEM_FIRST "pmix.cnct.sys.first" (bool)

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

PMIX_SERVER_PIDINFO "pmix.srvr.pidinfo" (pid_t)

process identifier (PID) of the target PMIx server for a tool.

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

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

PMIX_CONNECT_RETRY_DELAY "pmix.tool.retry" (uint32_t)

Time in seconds between connection attempts to a PMIx server.

PMIX_CONNECT_MAX_RETRIES "pmix.tool.mretries" (uint32_t)

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

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)

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

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

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

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

Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

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

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

PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)

The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)

The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

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

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

7 **PMIX_EVENT_BASE** "pmix.evbase" (struct event_base *)
 8 Pointer to libevent² **event_base** to use in place of the internal progress thread.

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



13 Description

14 Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided
 15 **pmix_proc_t** struct. The *info* array is used to pass user requests pertaining to the initialization
 16 and subsequent operations. Passing a **NULL** value for the array pointer is supported if no directives
 17 are desired.

18 If called with the **PMIX_TOOL_DO_NOT_CONNECT** attribute, the PMIx tool library will fully
 19 initialize but not attempt to connect to a PMIx server. The tool can connect to a server at a later
 20 point in time, if desired, by calling the **PMIx_tool_connect_to_server** function. In all
 21 other cases, the PMIx tool library will attempt to connect to a PMIx server according to the
 22 following precedence chain:

- 23 • if **PMIX_SERVER_URI** or **PMIX_TCP_URI** is given, then connection will be attempted to the
 24 server at the specified URI. Note that it is an error for both of these attributes to be specified.
 25 **PMIX_SERVER_URI** is the preferred method as it is more generalized — **PMIX_TCP_URI** is
 26 provided for those cases where the user specifically wants to use a TCP transport for the
 27 connection and wants to error out if it is not available or cannot succeed. The PMIx library will
 28 return an error if connection fails — it will not proceed to check for other connection options as
 29 the user specified a particular one to use
- 30 • if **PMIX_SERVER_PIDINFO** was provided, then the tool will search under the directory
 31 provided by the **PMIX_SERVER_TMPDIR** environmental variable for a rendezvous file created
 32 by the process corresponding to that PID. The PMIx library will return an error if the rendezvous
 33 file cannot be found, or the connection is refused by the server

²<http://libevent.org/>

- 1 • if `PMIX_CONNECT_TO_SYSTEM` is given, then the tool will search for a system-level
2 rendezvous file created by a PMIx server in the directory specified by the
3 `PMIX_SYSTEM_TMPDIR` environmental variable. If found, then the tool will attempt to
4 connect to it. An error is returned if the rendezvous file cannot be found or the connection is
5 refused.
- 6 • if `PMIX_CONNECT_SYSTEM_FIRST` is given, then the tool will search for a system-level
7 rendezvous file created by a PMIx server in the directory specified by the
8 `PMIX_SYSTEM_TMPDIR` environmental variable. If found, then the tool will attempt to
9 connect to it. In this case, no error will be returned if the rendezvous file is not found or
10 connection is refused — the PMIx library will silently continue to the next option
- 11 • lastly and by default, the tool will search the directory tree under the directory provided by the
12 `PMIX_SERVER_TMPDIR` environmental variable for rendezvous files of PMIx servers,
13 attempting to connect to each it finds until one accepts the connection. If no rendezvous files are
14 found, or all contacted servers refuse connection, then the PMIx library will return an error.


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


19 Note that the PMIx tool library is referenced counted, and so multiple calls to `PMIx_tool_init`
20 are allowed. Thus, one way to obtain the namespace and rank of the process is to simply call
21 `PMIx_tool_init` with a non-NULL parameter.

22 4.2.2 `PMIx_tool_finalize`

23 **Summary**
24 Finalize the PMIx library for a tool connection.

25 **Format**

PMIx v2.0 

26 `pmix_status_t`
27 `PMIx_tool_finalize(void)` 

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

29 **Description**
30 Finalize the PMIx tool library, closing the connection to the server. An error code will be returned
31 if, for some reason, the connection cannot be cleanly terminated — in this case, the connection is
32 dropped.

1 4.2.3 PMIx_tool_connect_to_server

2 Summary

3 Switch connection from the current PMIx server to another one, or initialize a connection to a
4 specified server.

5 Format

PMIx v3.0

```
6 pmix_status_t  
7 PMIx_tool_connect_to_server(pmix_proc_t *proc,  
8                             pmix_info_t info[], size_t ninfo)
```

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

Required Attributes

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

11 **PMIX_CONNECT_TO_SYSTEM** "pmix.cnct.sys" (bool)

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

13 **PMIX_CONNECT_SYSTEM_FIRST** "pmix.cnct.sys.first" (bool)

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

15 **PMIX_SERVER_URI** "pmix.srvr.uri" (char*)

16 URI of the PMIx server to be contacted.

17 **PMIX_SERVER_NAMESPACE** "pmix.srv.namespace" (char*)

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

19 **PMIX_SERVER_PIDINFO** "pmix.srvr.pidinfo" (pid_t)

20 PID of the target PMIx server for a tool.

Description

A tool may call `PMIx_tool_init` with the `PMIX_TOOL_DO_NOT_CONNECT` attribute in which case they can use this function to connect to a specific server. Additionally, a tool may use this function to switch connection from the current PMIx server to another one. Closes the connection, if existing, to a server and establishes a connection to the specified server. This function can be called at any time by a PMIx tool to shift connections between servers. The process identifier assigned to this tool is returned in the provided `pmix_proc_t` struct. Passing a value of `NULL` for this parameter is allowed if the user wishes solely to connect to the PMIx server and does not require return of the identifier at that time.

Advice to PMIx library implementers

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

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

Advice to users

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

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

CHAPTER 5

Key/Value Management

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

5.1 Setting and Accessing Key/Value Pairs

5.1.1 `PMIx_Put`

Summary

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

Format

PMIx v1.0

```
pmix_status_t  
PMIx_Put (pmix_scope_t scope,  
          const pmix_key_t key,  
          pmix_value_t *val)
```

IN scope

Distribution scope of the provided value (handle)

IN key

key (`pmix_key_t`)

IN value

Reference to a `pmix_value_t` structure (handle)

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

Description

Push a value into the client's namespace. The client's PMIx library will cache the information locally until `PMIx_Commit` is called.

The provided *scope* is passed to the local PMIx server, which will distribute the data to other processes according to the provided scope. The `pmix_scope_t` values are defined in Section 14.2.9 on page 301. Specific implementations may support different scope values, but all implementations must support at least `PMIX_GLOBAL`.

The `pmix_value_t` structure supports both string and binary values. PMIx implementations will support heterogeneous environments by properly converting binary values between host architectures, and will copy the provided *value* into internal memory.

Advice to PMIx library implementers

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

Advice to users

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

Note that keys starting with a string of “`pmix`” are exclusively reserved for the PMIx standard and must not be used in calls to `PMIx_Put`. Thus, applications should never use a defined “`PMIX_`” attribute as the key in a call to `PMIx_Put`.

5.1.2 `PMIx_Get`

Summary

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

1
PMIx v1.0

Format

C

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

C

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

16 Returns [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

Required Attributes

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

- 18 **PMIX_OPTIONAL** "pmix.optional" (bool)
19 Look only in the client's local data store for the requested value - do not request data from
20 the PMIx server if not found.
- 21 **PMIX_IMMEDIATE** "pmix.immediate" (bool)
22 Specified operation should immediately return an error from the PMIx server if the requested
23 data cannot be found - do not request it from the host RM.
- 24 **PMIX_DATA_SCOPE** "pmix.scope" ([pmix_scope_t](#))
25 Scope of the data to be found in a [PMIx_Get](#) call.
- 26 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)
27 Return information about the specified session. If information about a session other than the
28 one containing the requesting process is desired, then the attribute array must contain a
29 [PMIX_SESSION_ID](#) attribute identifying the desired target.
- 30 **PMIX_JOB_INFO** "pmix.job.info" (bool)

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

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

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

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

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

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* as published by the process identified in the given `pmix_proc_t`, returning a pointer to the value in the given address.

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

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

Advice to users

Information provided by the PMIx server at time of process start is accessed by providing the namespace of the job with the rank set to `PMIX_RANK_WILDCARD`. The list of data referenced in this way is maintained on the PMIx web site at <https://pmix.org/support/faq/wildcard-rank-access/> but includes items such as the number of processes in the namespace (`PMIX_JOB_SIZE`), total available slots in the allocation (`PMIX_UNIV_SIZE`), and the number of nodes in the allocation (`PMIX_NUM_NODES`).

Data posted by a process via `PMIx_Put` needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of `PMIX_RANK_WILDCARD` when the information being retrieved refers to something non-rank specific (e.g., number of processes on a node, number of processes in a job), and using the rank of the relevant process when requesting information that is rank-specific (e.g., the URI of the process, or the node upon which it is executing). Each subsection of Section 14.4 indicates the appropriate rank value for referencing the defined attribute.

5.1.3 PMIx_Get_nb

Summary

Nonblocking `PMIx_Get` operation.

1
PMIx v1.0

Format

C

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

C

6 **IN** **proc**
7 process reference (handle)
8 **IN** **key**
9 key to retrieve (string)
10 **IN** **info**
11 Array of info structures (array of handles)
12 **IN** **ninfo**
13 Number of elements 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:

- 19 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
20 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
21 function prior to returning from the API.
- 22 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
23 returned *success* - the *cbfunc* will *not* be called
- 24 • a PMIx error constant indicating either an error in the input or that the request was immediately
25 processed and failed - the *cbfunc* will *not* be called

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

- 28 • **PMIX_SUCCESS** The requested data has been returned
- 29 • **PMIX_ERR_NOT_FOUND** The requested data was not available
- 30 • a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

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

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

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

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

6 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)
7 Return information about the specified session. If information about a session other than the
8 one containing the requesting process is desired, then the attribute array must contain a
9 **PMIX_SESSION_ID** attribute identifying the desired target.

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

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

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

27 **PMIX_GET_STATIC_VALUES** "pmix.get.static" (bool)
28 Request that any pointers in the returned value point directly to values in the key-value store
29 and indicate that user takes responsibility for properly releasing memory on the returned
30 value (i.e., free'ing the value structure but not the pointer fields). Note that a return status of
31 **PMIX_ERR_GET_MALLOC_REQD** indicates that direct pointers could not be supported -
32 thus, the returned data contains allocated memory that the user must release.

▲-----▲

▼-----▼

Optional Attributes

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

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

Advice to PMIx library implementers

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

Description

The callback function will be executed once the specified data becomes available from the identified process and retrieved by the local server. The *info* array is used as described by the `PMIx_Get` routine.

Advice to users

Information provided by the PMIx server at time of process start is accessed by providing the namespace of the job with the rank set to `PMIX_RANK_WILDCARD`. Attributes referenced in this way are identified in 14.4 but includes items such as the number of processes in the namespace (`PMIX_JOB_SIZE`), total available slots in the allocation (`PMIX_UNIV_SIZE`), and the number of nodes in the allocation (`PMIX_NUM_NODES`).

In general, data posted by a process via `PMIx_Put` and data that refers directly to a process-related value needs to be retrieved by specifying the rank of the posting process. All other information is retrievable using a rank of `PMIX_RANK_WILDCARD`, as illustrated in 5.1.5. See 14.4.11 for an explanation regarding use of the *level* attributes.

5.1.4 PMIx_Store_internal

Summary

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

1
PMIx v1.0

Format

C

```
2 pmix_status_t  
3 PMIx_Store_internal(const pmix_proc_t *proc,  
4                   const pmix_key_t key,  
5                   pmix_value_t *val);
```

C

```
6 IN proc  
7   process reference (handle)  
8 IN key  
9   key to retrieve (string)  
10 IN val  
11   Value to store (handle)
```

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

Description

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

16 5.1.5 Accessing information: examples

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

21 5.1.5.1 Session-level information

22 The **PMIx_Get** API does not include an argument for specifying the **session** associated with
23 the information being requested. Information regarding the session containing the requestor can be
24 obtained by the following methods:

- 25 • for session-level attributes (e.g., **PMIX_UNIV_SIZE**), specifying the requestor’s namespace
26 and a rank of **PMIX_RANK_WILDCARD**; or
- 27 • for non-specific attributes (e.g., **PMIX_NUM_NODES**), including the **PMIX_SESSION_INFO**
28 attribute to indicate that the session-level information for that attribute is being requested

29 Example requests are shown below:

C

```

1  pmix_info_t info;
2  pmix_value_t *value;
3  pmix_status_t rc;
4  pmix_proc_t myproc, wildcard;
5
6  /* initialize the client library */
7  PMIx_Init(&myproc, NULL, 0);
8
9  /* get the #slots in our session */
10 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);
11 rc = PMIx_Get(&wildcard, PMIX_UNIV_SIZE, NULL, 0, &value);
12
13 /* get the #nodes in our session */
14 PMIX_INFO_LOAD(&info, PMIX_SESSION_INFO, NULL, PMIX_BOOL);
15 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);

```

C

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

C

```

20 pmix_info_t info[2];
21 pmix_value_t *value;
22 pmix_status_t rc;
23 pmix_proc_t myproc;
24 uint32_t sid;
25
26 /* initialize the client library */
27 PMIx_Init(&myproc, NULL, 0);
28
29 /* get the #nodes in a different session */
30 sid = 12345;
31 PMIX_INFO_LOAD(&info[0], PMIX_SESSION_INFO, NULL, PMIX_BOOL);
32 PMIX_INFO_LOAD(&info[1], PMIX_SESSION_ID, &sid, PMIX_UINT32);
33 rc = PMIx_Get(&myproc, PMIX_NUM_NODES, info, 2, &value);

```

C

1 5.1.5.2 Job-level information

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

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

8 Example requests are shown below:

```
9 pmix_info_t info;  
10 pmix_value_t *value;  
11 pmix_status_t rc;  
12 pmix_proc_t myproc, wildcard;  
13  
14 /* initialize the client library */  
15 PMIx_Init(&myproc, NULL, 0);  
16  
17 /* get the #apps in our job */  
18 PMIX_PROC_LOAD(&wildcard, myproc.nspace, PMIX_RANK_WILDCARD);  
19 rc = PMIx_Get(&wildcard, PMIX_JOB_NUM_APPS, NULL, 0, &value);  
20  
21 /* get the #nodes in our job */  
22 PMIX_INFO_LOAD(&info, PMIX_JOB_INFO, NULL, PMIX_BOOL);  
23 rc = PMIx_Get(&wildcard, PMIX_NUM_NODES, &info, 1, &value);
```

24 5.1.5.3 Application-level information

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

- 26 • for application-level attributes (e.g., `PMIX_APP_SIZE`), specifying the namespace and rank of
27 a process within that application;
- 28 • for application-level attributes (e.g., `PMIX_APP_SIZE`), including the `PMIX_APPNUM`
29 attribute specifying the application whose information is being requested. In this case, the
30 namespace field of the *proc* argument is used to reference the *job* containing the application -
31 the *rank* field is ignored;
- 32 • or application-level attributes (e.g., `PMIX_APP_SIZE`), including the `PMIX_APPNUM` and
33 `PMIX_NAMESPACE` or `PMIX_JOBID` attributes specifying the job/application whose information
34 is being requested. In this case, the *proc* argument is ignored;
- 35 • for non-specific attributes (e.g., `PMIX_NUM_NODES`), including the `PMIX_APP_INFO`
36 attribute to indicate that the application-level information for that attribute is being requested

1 Example requests are shown below:

```
2 pmix_info_t info;
3 pmix_value_t *value;
4 pmix_status_t rc;
5 pmix_proc_t myproc, otherproc;
6 uint32_t appsize, appnum;
7
8 /* initialize the client library */
9 PMIx_Init(&myproc, NULL, 0);
10
11 /* get the #processes in our application */
12 rc = PMIx_Get(&myproc, PMIX_APP_SIZE, NULL, 0, &value);
13 appsize = value->data.uint32;
14
15 /* get the #nodes in an application containing "otherproc".
16  * Note that the rank of a process in the other application
17  * must be obtained first - a simple method is shown here */
18
19 /* assume for this example that we are in the first application
20  * and we want the #nodes in the second application - use the
21  * rank of the first process in that application, remembering
22  * that ranks start at zero */
23 PMIX_PROC_LOAD(&otherproc, myproc.namespace, appsize);
24
25 PMIX_INFO_LOAD(&info, PMIX_APP_INFO, NULL, PMIX_BOOL);
26 rc = PMIx_Get(&otherproc, PMIX_NUM_NODES, &info, 1, &value);
27
28 /* alternatively, we can directly ask for the #nodes in
29  * the second application in our job, again remembering that
30  * application numbers start with zero */
31 appnum = 1;
32 PMIX_INFO_LOAD(&appinfo[0], PMIX_APP_INFO, NULL, PMIX_BOOL);
33 PMIX_INFO_LOAD(&appinfo[1], PMIX_APPNUM, &appnum, PMIX_UINT32);
34 rc = PMIx_Get(&myproc, PMIX_NUM_NODES, appinfo, 2, &value);
35
```

36 5.1.5.4 Process-level information

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

1 5.1.5.5 Node-level information

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

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

10 Example requests are shown below:

```
▼────────────────────────────────────────── C ───────────────────────────────────────────▼  
11 pmix_info_t info[2];  
12 pmix_value_t *value;  
13 pmix_status_t rc;  
14 pmix_proc_t myproc, otherproc;  
15 uint32_t nodeid;  
16  
17 /* initialize the client library */  
18 PMIx_Init(&myproc, NULL, 0);  
19  
20 /* get the #procs on our node */  
21 rc = PMIx_Get(&myproc, PMIX_NODE_SIZE, NULL, 0, &value);  
22  
23 /* get the #slots on another node */  
24 PMIX_INFO_LOAD(&info[0], PMIX_NODE_INFO, NULL, PMIX_BOOL);  
25 PMIX_INFO_LOAD(&info[1], PMIX_HOSTNAME, "remotehost", PMIX_STRING);  
26 rc = PMIx_Get(&myproc, PMIX_MAX_PROCS, info, 2, &value);  
27
```

```
▲────────────────────────────────────────── C ───────────────────────────────────────────▲  
▼────────────────────────────────────────── Advice to users ───────────────────────────────────────────▼  
28 An explanation of the use of PMIx_Get versus PMIx_Query_info_nb is provided in 7.1.4.1.  
▲──────────────────────────────────────────▲
```

29 5.2 Exchanging Key/Value Pairs

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

1 5.2.1 PMIx_Commit

2 Summary

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

4 Format

PMIx v1.0

C

5 `pmix_status_t PMIx_Commit(void)`

C

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

7 Description

8 This is an asynchronous operation. The PRI will immediately return to the caller while the data is
9 transmitted to the local server in the background.

Advice to users

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

13 5.2.2 PMIx_Fence

14 Summary

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

1
PMIx v1.0

Format

C

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

C

- 5 **IN** `procs`
Array of `pmix_proc_t` structures (array of handles)
- 7 **IN** `nprocs`
Number of element in the `procs` array (integer)
- 9 **IN** `info`
Array of info structures (array of handles)
- 11 **IN** `ninfo`
Number of element in the `info` array (integer)

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

Required Attributes

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

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

Optional Attributes

17 The following attributes are optional for host environments:

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

22 **PMIX_COLLECTIVE_ALGO** "pmix.calgo" (char*)
23 Comma-delimited list of algorithms to use for the collective operation. PMIx does not
24 impose any requirements on a host environment's collective algorithms. Thus, the
25 acceptable values for this attribute will be environment-dependent - users are encouraged to
26 check their host environment for supported values.

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

Advice to PMIx library implementers

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

Description

Passing a `NULL` pointer as the *procs* parameter indicates that the fence is to span all processes in the client's namespace. Each provided `pmix_proc_t` struct can pass `PMIX_RANK_WILDCARD` to indicate that all processes in the given namespace are participating.

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

Note that for scalability reasons, the default behavior for `PMIx_Fence` is to not collect the data.

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

5.2.3 `PMIx_Fence_nb`

Summary

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

1
PMIx v1.0

Format

C

```

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

```

C

- 6 **IN procs**
Array of `pmix_proc_t` structures (array of handles)
- 7
- 8 **IN nprocs**
Number of element in the *procs* array (integer)
- 9
- 10 **IN info**
Array of info structures (array of handles)
- 11
- 12 **IN ninfo**
Number of element in the *info* array (integer)
- 13
- 14 **IN cbfunc**
Callback function (function reference)
- 15
- 16 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 17

18 Returns one of the following:

- 19 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
20 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
21 function prior to returning from the API.
- 22 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
23 returned *success* - the *cbfunc* will *not* be called. This can occur if the collective involved only
24 processes on the local node.
- 25 • a PMIx error constant indicating either an error in the input or that the request was immediately
26 processed and failed - the *cbfunc* will *not* be called

Required Attributes

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

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx library implementers

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

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

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

Description

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

Note that for scalability reasons, the default behavior for **PMIx_Fence_nb** is to not collect the data.

See the **PMIx_Fence** description for further details.

1 5.3 Publish and Lookup Data

2 The APIs defined in this section publish data from one client that can be later exchanged and looked
3 up by another client.

▼ **Advice to PMIx library implementers** ▼

4 PMIx libraries that support any of the functions in this section are required to support *all* of them.

▲

▼ **Advice to PMIx server hosts** ▼

5 Host environments that support any of the functions in this section are required to support *all* of
6 them.

▲

7 5.3.1 PMIx_Publish

8 **Summary**

9 Publish data for later access via [PMIx_Lookup](#) .

10 **Format**

PMIx v1.0

▼ **C** ▼

11 `pmix_status_t`

12 `PMIx_Publish(const pmix_info_t info[], size_t ninfo)`

▲ **C** ▲

13 **IN** `info`

14 Array of info structures (array of handles)

15 **IN** `ninfo`

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

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

▼ **Required Attributes** ▼

18 PMIx libraries are not required to directly support any attributes for this function. However, any
19 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
20 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that
21 published the info.

▲

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)

Value for calls to **PMIx_Publish**.

Advice to PMIx library implementers

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

Description

Publish the data in the *info* array for subsequent lookup. By default, the data will be published into the **PMIX_RANGE_SESSION** range and with **PMIX_PERSIST_APP** persistence. Changes to those values, and any additional directives, can be included in the **pmix_info_t** array. Attempts to access the data by processes outside of the provided data range will be rejected. The persistence parameter instructs the server as to how long the data is to be retained.

The blocking form will block until the server confirms that the data has been sent to the PMIx server and that it has obtained confirmation from its host SMS daemon that the data is ready to be looked up. Data is copied into the backing key-value data store, and therefore the *info* array can be released upon return from the blocking function call.

Advice to users

Publishing duplicate keys is permitted provided they are published to different ranges.

Advice to PMIx library implementers

Implementations should, to the best of their ability, detect duplicate keys being posted on the same data range and protect the user from unexpected behavior by returning the **PMIX_ERR_DUPLICATE_KEY** error.

1 5.3.2 PMIx_Publish_nb

2 Summary

3 Nonblocking `PMIx_Publish` routine.

4 Format

PMIx v1.0

```
5 pmix_status_t
6 PMIx_Publish_nb(const pmix_info_t info[], size_t ninfo,
7                 pmix_op_cbfunc_t cbfunc, void *cbdata)
```

8 IN info

9 Array of info structures (array of handles)

10 IN ninfo

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

12 IN cbfunc

13 Callback function `pmix_op_cbfunc_t` (function reference)

14 IN cbdata

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

16 Returns one of the following:

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

Required Attributes

24 PMIx libraries are not required to directly support any attributes for this function. However, any
25 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
26 *required* to add the `PMIX_USERID` and the `PMIX_GRPID` attributes of the client process that
27 published the info.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

PMIX_PERSISTENCE "pmix.persist" (pmix_persistence_t)

Value for calls to **PMIx_Publish**.

Advice to PMIx library implementers

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

Description

Nonblocking **PMIx_Publish** routine. The non-blocking form will return immediately, executing the callback when the PMIx server receives confirmation from its host SMS daemon.

Note that the function will return an error if a **NULL** callback function is given, and that the *info* array must be maintained until the callback is provided.

5.3.3 PMIx_Lookup

Summary

Lookup information published by this or another process with **PMIx_Publish** or **PMIx_Publish_nb**.

1
PMIx v1.0

Format

```

2 pmix_status_t
3 PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
4             const pmix_info_t info[], size_t ninfo)

```

INOUT data

Array of publishable data structures (array of handles)

IN ndata

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

IN info

Array of info structures (array of handles)

IN ninfo

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

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

Required Attributes

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is requesting the info.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

PMIX_WAIT "pmix.wait" (int)

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

Advice to PMIx library implementers

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

Description

Lookup information published by this or another process. By default, the search will be conducted across the `PMIX_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 5.3.4 PMIx_Lookup_nb

2 Summary

3 Nonblocking version of [PMIx_Lookup](#) .

4 Format

PMIx v1.0

C

5 `pmix_status_t`

```
6 PMIx_Lookup_nb(char **keys,  
7           const pmix_info_t info[], size_t ninfo,  
8           pmix_lookup_cbfunc_t cbfunc, void *cbdata)
```

C

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

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 is
27 requesting the info.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

PMIX_WAIT "pmix.wait" (int)

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

Advice to PMIx library implementers

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

Description

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

5.3.5 PMIx_Unpublish

Summary

Unpublish data posted by this process using the given keys.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Unpublish(char **keys,
4                 const pmix_info_t info[], size_t ninfo)
```

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 5.3.6 PMIx_Unpublish_nb

8 **Summary**
 9 Nonblocking version of **PMIx_Unpublish** .

10 **Format**

PMIx v1.0

```

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)
  
```

- 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

PMIx libraries are not required to directly support any attributes for this function. However, any provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process that is requesting the operation.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_RANGE "pmix.range" (pmix_data_range_t)

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

Advice to PMIx library implementers

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

Description

Non-blocking form of the **PMIx_Unpublish** function. The callback function will be executed once the server confirms removal of the specified data. The *info* array must be maintained until the callback is provided.

CHAPTER 6

Process Management

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

3 6.1 Abort

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

6 6.1.1 PMIx_Abort

7 Summary

8 Abort the specified processes

9 Format

PMIx v1.0

C

```
10 pmix_status_t  
11 PMIx_Abort(int status, const char msg[],  
12           pmix_proc_t procs[], size_t nprocs)
```

C

13 **IN** **status**

14 Error code to return to invoking environment (integer)

15 **IN** **msg**

16 String message to be returned to user (string)

17 **IN** **procs**

18 Array of [pmix_proc_t](#) structures (array of handles)

19 **IN** **nprocs**

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

21 Returns [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

Description

Request that the host resource manager print the provided message and abort the provided array of *procs*. A Unix or POSIX environment should handle the provided status as a return error code from the main program that launched the application. A **NULL** for the *procs* array indicates that all processes in the caller's namespace are to be aborted, including itself. Passing a **NULL** *msg* parameter is allowed.

Advice to users

The response to this request is somewhat dependent on the specific resource manager and its configuration (e.g., some resource managers will not abort the application if the provided status is zero unless specifically configured to do so, and some cannot abort subsets of processes in an application), and thus lies outside the control of PMIx itself. However, the PMIx client library shall inform the RM of the request that the specified *procs* be aborted, regardless of the value of the provided status.

Note that race conditions caused by multiple processes calling **PMIx_Abort** are left to the server implementation to resolve with regard to which status is returned and what messages (if any) are printed.

6.2 Process Creation

The **PMIx_Spawn** commands spawn new processes and/or applications in the PMIx universe. This may include requests to extend the existing resource allocation or obtain a new one, depending upon provided and supported attributes.

6.2.1 PMIx_Spawn

Summary

Spawn a new job.

1
PMIx v1.0

Format

C

```
2 pmix_status_t  
3 PMIx_Spawn(const pmix_info_t job_info[], size_t ninfo,  
4           const pmix_app_t apps[], size_t napps,  
5           char nspace[])
```

C

6 **IN** `job_info`
7 Array of info structures (array of handles)
8 **IN** `ninfo`
9 Number of elements in the `job_info` array (integer)
10 **IN** `apps`
11 Array of `pmix_app_t` structures (array of handles)
12 **IN** `napps`
13 Number of elements in the `apps` array (integer)
14 **OUT** `nspace`
15 Namespace of the new job (string)

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

Required Attributes

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

20 **PMIX_SPAWNED** "pmix.spawned" (bool)
21 `true` if this process resulted from a call to `PMIx_Spawn`.
22 **PMIX_PARENT_ID** "pmix.parent" (pmix_proc_t)
23 Process identifier of the parent process of the calling process.
24 **PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)
25 The requesting process is a PMIx client.
26 **PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)
27 The requesting process is a PMIx tool.

28
29 Host environments that implement support for `PMIx_Spawn` are required to pass the
30 `PMIX_SPAWNED` and `PMIX_PARENT_ID` attributes to all PMIx servers launching new child
31 processes so those values can be returned to clients upon connection to the PMIx server. In
32 addition, they are required to support the following attributes when present in either the `job_info` or
33 the `info` array of an element of the `apps` array:

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

1 Working directory for spawned processes.

2 **PMIX_SET_SESSION_CWD** "pmix.ssn cwd" (bool)

3 Set the application's current working directory to the session working directory assigned by
4 the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for
5 the rank to discover the session working directory assigned to the provided namespace

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

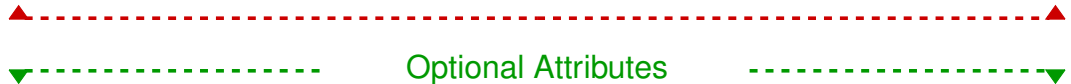
7 Prefix to use for starting spawned processes.

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

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

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

11 Hostfile to use for spawned processes.



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

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

14 Hostfile listing hosts to add to existing allocation.

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

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

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

18 Preload binaries onto nodes.

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

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

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

22 Name of personality to use.

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

24 Mapping mechanism to use for placing spawned processes - when accessed using
25 **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping
26 mechanism used for the provided namespace.

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

28 Display process mapping upon spawn.

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

30 Number of processes to spawn on each identified resource.

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

1 Process mapping policy - when accessed using `PMIx_Get` , use the
2 `PMIX_RANK_WILDCARD` value for the rank to discover the mapping policy used for the
3 provided namespace

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

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

12 `PMIX_NON_PMI` "pmix.nonpmi" (bool)
13 Spawned processes will not call `PMIx_Init` .

14 `PMIX_STDIN_TGT` "pmix.stdin" (uint32_t)
15 Spawned process rank that is to receive `stdin`.

16 `PMIX_FWD_STDIN` "pmix.fwd.stdin" (bool)
17 Forward this process's `stdin` to the designated process.

18 `PMIX_FWD_STDOUT` "pmix.fwd.stdout" (bool)
19 Forward `stdout` from spawned processes to this process.

20 `PMIX_FWD_STDERR` "pmix.fwd.stderr" (bool)
21 Forward `stderr` from spawned processes to this process.

22 `PMIX_DEBUGGER_DAEMONS` "pmix.debugger" (bool)
23 Spawned application consists of debugger daemons.

24 `PMIX_TAG_OUTPUT` "pmix.tagout" (bool)
25 Tag application output with the identity of the source process.

26 `PMIX_TIMESTAMP_OUTPUT` "pmix.tsout" (bool)
27 Timestamp output from applications.

28 `PMIX_MERGE_STDERR_STDOUT` "pmix.mergeerrout" (bool)
29 Merge `stdout` and `stderr` streams from application processes.

30 `PMIX_OUTPUT_TO_FILE` "pmix.outfile" (char*)
31 Output application output to the specified file.

32 `PMIX_INDEX_ARGV` "pmix.indxargv" (bool)
33 Mark the `argv` with the rank of the process.

34 `PMIX_CPUS_PER_PROC` "pmix.cpusperproc" (uint32_t)

1 Number of cpus to assign to each rank - when accessed using `PMIx_Get` , use the
2 `PMIX_RANK_WILDCARD` value for the rank to discover the cpus/process assigned to the
3 provided namespace

4 `PMIX_NO_PROCS_ON_HEAD` "pmix.nolocal" (bool)

5 Do not place processes on the head node.

6 `PMIX_NO_OVERSUBSCRIBE` "pmix.noover" (bool)

7 Do not oversubscribe the cpus.

8 `PMIX_REPORT_BINDINGS` "pmix.repbinding" (bool)

9 Report bindings of the individual processes.

10 `PMIX_CPU_LIST` "pmix.cpulist" (char*)

11 List of cpus to use for this job - when accessed using `PMIx_Get` , use the
12 `PMIX_RANK_WILDCARD` value for the rank to discover the cpu list used for the provided
13 namespace

14 `PMIX_JOB_RECOVERABLE` "pmix.recover" (bool)

15 Application supports recoverable operations.

16 `PMIX_JOB_CONTINUOUS` "pmix.continuous" (bool)

17 Application is continuous, all failed processes should be immediately restarted.

18 `PMIX_MAX_RESTARTS` "pmix.maxrestarts" (uint32_t)

19 Maximum number of times to restart a job - when accessed using `PMIx_Get` , use the
20 `PMIX_RANK_WILDCARD` value for the rank to discover the max restarts for the provided
21 namespace

22 `PMIX_NOTIFY_COMPLETION` "pmix.notecomp" (bool)

23 Notify the parent process upon termination of child job.



24 **Description**

25 Spawn a new job. The assigned namespace of the spawned applications is returned in the *nspc*
26 parameter. A `NULL` value in that location indicates that the caller doesn't wish to have the
27 namespace returned. The *nspc* array must be at least of size one more than `PMIX_MAX_NSLN` .

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.

6.2.2 PMIx_Spawn_nb

Summary

Nonblocking version of the [PMIx_Spawn](#) routine.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_Spawn_nb(const pmix_info_t job_info[], size_t ninfo,
              const pmix_app_t apps[], size_t napps,
              pmix_spawn_cbfunc_t cbfunc, void *cbdata)
```

C

IN job_info

Array of info structures (array of handles)

IN ninfo

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

IN apps

Array of [pmix_app_t](#) structures (array of handles)

IN cbfunc

Callback function [pmix_spawn_cbfunc_t](#) (function reference)

IN cbdata

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

Returns one of the following:

- [PMIX_SUCCESS](#), indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- a PMIx error constant indicating an error in the request - the *cbfunc* will *not* be called

Required Attributes

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

PMIX_SPAWNED "pmix.spawned" (bool)

true if this process resulted from a call to [PMIx_Spawn](#).

PMIX_PARENT_ID "pmix.parent" (pmix_proc_t)

Process identifier of the parent process of the calling process.

PMIX_REQUESTOR_IS_CLIENT "pmix.req.client" (bool)

The requesting process is a PMIx client.

PMIX_REQUESTOR_IS_TOOL "pmix.req.tool" (bool)

The requesting process is a PMIx tool.

Host environments that implement support for [PMIx_Spawn](#) are required to pass the [PMIX_SPAWNED](#) and [PMIX_PARENT_ID](#) attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX_WDIR "pmix.wdir" (char*)

Working directory for spawned processes.

PMIX_SET_SESSION_CWD "pmix.ssn cwd" (bool)

Set the application's current working directory to the session working directory assigned by the RM - when accessed using [PMIx_Get](#), use the [PMIX_RANK_WILDCARD](#) value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX "pmix.prefix" (char*)

Prefix to use for starting spawned processes.

PMIX_HOST "pmix.host" (char*)

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

PMIX_HOSTFILE "pmix.hostfile" (char*)

Hostfile to use for spawned processes.

Optional Attributes

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

PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)

Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST "pmix.addhost" (char*)

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

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)

Preload binaries onto nodes.

PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)

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

PMIX_PERSONALITY "pmix.pers" (char*)

Name of personality to use.

PMIX_MAPPER "pmix.mapper" (char*)

Mapping mechanism to use for placing spawned processes - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping mechanism used for the provided namespace.

PMIX_DISPLAY_MAP "pmix.dispmap" (bool)

Display process mapping upon spawn.

PMIX_PPR "pmix.ppr" (char*)

Number of processes to spawn on each identified resource.

PMIX_MAPBY "pmix.mapby" (char*)

Process mapping policy - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the provided namespace

PMIX_RANKBY "pmix.rankby" (char*)

Process ranking policy - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the provided namespace

PMIX_BINDTO "pmix.bindto" (char*)

Process binding policy - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the provided namespace

PMIX_NON_PMI "pmix.nonpmi" (bool)

Spawned processes will not call **PMIx_Init**.

PMIX_STDIN_TGT "pmix.stdin" (uint32_t)

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

1 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)
2 Forward this process's **stdin** to the designated process.

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

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

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

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

11 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
12 Timestamp output from applications.

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

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

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

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

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

25 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)
26 Do not oversubscribe the cpus.

27 **PMIX_REPORT_BINDINGS** "pmix.repbinding" (bool)
28 Report bindings of the individual processes.

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

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

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

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

Description

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

Advice to users

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

6.3 Connecting and Disconnecting Processes

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

Advice to PMIx server hosts

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

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

Advice to users

1 Attempting to *connect* processes solely within the same namespace is essentially a *no-op* operation.
2 While not explicitly prohibited, users are advised that a PMIx implementation or host environment
3 may return an error in such cases.

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

6.3.1 PMIx_Connect

Summary

8 Connect namespaces.

Format

10 *PMIx v1.0*

```
11 pmix_status_t  
12 PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,  
13              const pmix_info_t info[], size_t ninfo)
```

- 14 **IN** `procs`
15 Array of proc structures (array of handles)
- 16 **IN** `nprocs`
17 Number of elements in the *procs* array (integer)
- 18 **IN** `info`
19 Array of info structures (array of handles)
- 20 **IN** `ninfo`
21 Number of elements in the *info* array (integer)

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

Required Attributes

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx library implementers

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

Description

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

Advice to users

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

Processes that combine via **PMIx_Connect** must call **PMIx_Disconnect** prior to finalizing and/or terminating - any process in the assemblage failing to meet this requirement will cause a **PMIX_ERR_INVALID_TERMINATION** event to be generated.

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

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

6.3.2 PMIx_Connect_nb

Summary

Nonblocking **PMIx_Connect_nb** routine.

1
PMIx v1.0

Format

C

```

2 pmix_status_t
3 PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,
4                 const pmix_info_t info[], size_t ninfo,
5                 pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN procs**
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

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx library implementers

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

Description

Nonblocking version of **PMIx_Connect**. The callback function is called once all processes identified in *procs* have called either **PMIx_Connect** or its non-blocking version, *and* the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes. See the advice provided in the description for **PMIx_Connect** for more information.

6.3.3 PMIx_Disconnect

Summary

Disconnect a previously connected set of processes.

1
PMIx v1.0

Format

C

```

2 pmix_status_t
3 PMIx_Disconnect(const pmix_proc_t procs[], size_t nprocs,
4                 const pmix_info_t info[], size_t ninfo);

```

C

- 5 **IN** `procs`
- 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.

Description

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

Advice to users

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

Advice to PMIx library implementers

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

Advice to PMIx server hosts

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

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

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

6.3.4 `PMIx_Disconnect_nb`

Summary

Nonblocking `PMIx_Disconnect` routine.

1
PMIx v1.0

Format

C

```
2 pmix_status_t
3 PMIx_Disconnect_nb(const pmix_proc_t procs[], size_t nprocs,
4                   const pmix_info_t info[], size_t ninfo,
5                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

- 6 **IN procs**
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 IO Forwarding

This section defines functions by which tools (e.g., debuggers) can request forwarding of input/output to/from other processes. The term “tool” widely refers to non-computational programs executed by the user or system administrator to monitor or control a principal computational program. Tools almost always interact with either the host environment, user applications, or both to perform administrative and support functions. For example, a debugger tool might be used to remotely control the processes of a parallel application, monitoring their behavior on a step-by-step basis.

Underlying the operation of many tools is a common need to forward stdin from the tool to targeted processes, and to return stdout/stderr from those processes for display on the user’s console. Historically, each tool developer was responsible for creating their own IO forwarding subsystem. However, with the introduction of PMIx as a standard mechanism for interacting between applications and the host environment, it has become possible to relieve tool developers of this burden.

Advice to PMIx server hosts

The responsibility of the host environment in forwarding of IO falls into the following areas:

- Capturing output from specified child processes
- Forwarding that output to the host of the PMIx server library that requested it
- Delivering that payload to the PMIx server library via the `PMIx_server_IOF_deliver` API for final dispatch

It is the responsibility of the PMIx library to buffer, format, and deliver the payload to the requesting client.

Advice to users

The forwarding of IO via PMI_x requires that both the host environment and the tool support PMI_x, but does not impose any similar requirements on the application itself.

6.4.1 PMI_x_IOF_pull

Summary

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

Format

PMI_x v3.0

C

`pmix_status_t`

```
PMIx_IOF_pull(const pmix_proc_t procs[], size_t nprocs,  
              const pmix_info_t directives[], size_t ndirs,  
              pmix_iof_channel_t channel, pmix_iof_cbfunc_t cbfunc,  
              pmix_hdlr_reg_cbfunc_t regcbfunc, void *regcbdata)
```

C

IN `procs`

Array of proc structures identifying desired source processes (array of handles)

IN `nprocs`

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

IN `directives`

Array of `pmix_info_t` structures (array of handles)

IN `ndirs`

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

IN `channel`

Bitmask of IO channels included in the request (`pmix_iof_channel_t`)

IN `cbfunc`

Callback function for delivering relevant output (`pmix_iof_cbfunc_t` function reference)

IN `regcbfunc`

Function to be called when registration is completed (`pmix_hdlr_reg_cbfunc_t` function reference)

IN `regcbdata`

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

1 If *regcbfunc* is **NULL**, the function call will be treated as a *blocking* call. In this case, the returned
2 status will be either (a) the IOF handler reference identifier if the value is greater than or equal to
3 zero, or (b) a negative error code indicative of the reason for the failure.

4 If the *regcbfunc* is non-**NULL**, the function call will be treated as a *non-blocking* call and will return
5 the following:

6 **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
7 callback function will be executed upon completion of the operation. Note that the library
8 must not invoke the callback function prior to returning from the API. The IOF handler
9 identifier will be returned in the callback

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

Required Attributes

12 The following attributes are required for PMIx libraries that support IO forwarding:

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

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

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

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

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

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

Optional Attributes

21 The following attributes are optional for PMIx libraries that support IO forwarding:

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

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

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

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

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

32 Tag output with the channel it comes from.

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

1 Timestamp output
2 **PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)
3 Format output in eXtensible Markup Language (XML)



4 **Description**
5 Register to receive output forwarded from a set of remote processes.



6 Providing a **NULL** function pointer for the *cbfunc* parameter will cause output for the indicated
7 channels to be written to their corresponding stdout/stderr file descriptors. Use of
8 **PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but should
9 be used carefully due to bandwidth considerations.



10 6.4.2 PMIx_IOF_deregister

11 **Summary**
12 Deregister from output forwarded from a set of remote processes.

13 Format

PMIx v3.0

```
14     pmix_status_t  
15     PMIx_IOF_deregister(size_t iofhdlr,  
16                        const pmix_info_t directives[], size_t ndirs,  
17                        pmix_op_cbfunc_t cbfunc, void *cbdata)
```

- 18 **IN iofhdlr**
19 Registration number returned from the **pmix_hdlr_reg_cbfunc_t** callback from the
20 call to **PMIx_IOF_pull** (**size_t**)
- 21 **IN directives**
22 Array of **pmix_info_t** structures (array of handles)
- 23 **IN ndirs**
24 Number of elements in the *directives* array (integer)
- 25 **IN cbfunc**
26 Callback function to be called when deregistration has been completed. (function reference)
- 27 **IN cbdata**
28 Data to be passed to the *cbfunc* callback function (memory reference)

1 If *cbfunc* is **NULL**, the function will be treated as a *blocking* call and the result of the operation
2 returned in the status code.

3 If *cbfunc* is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the
4 following:

- 5 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
6 provided *cbfunc*. Note that the library must not invoke the callback function prior to returning
7 from the API.
- 8 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
9 returned *success* - the *cbfunc* will *not* be called
- 10 • a PMIx error constant indicating either an error in the input or that the request was immediately
11 processed and failed - the *cbfunc* will *not* be called

12 The returned status code will be one of the following:

13 **PMIX_SUCCESS** The IOF handler was successfully deregistered.

14 **PMIX_ERR_BAD_PARAM** The provided *iofhdr* was unrecognized.

15 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.

16 **Description**

17 Deregister from output forwarded from a set of remote processes.

▼ **Advice to PMIx library implementers** ▼

18 Any currently buffered IO should be flushed upon receipt of a deregistration request. All received
19 IO after receipt of the request shall be discarded.

▲

20 **6.4.3 PMIx_IOF_push**

21 **Summary**

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

1
PMIx v3.0

Format

C

```
2 pmix_status_t
3 PMIx_IOF_push(const pmix_proc_t targets[], size_t ntargets,
4               pmix_byte_object_t *bo,
5               const pmix_info_t directives[], size_t ndirs,
6               pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

7 **IN targets**
8 Array of proc structures identifying desired target processes (array of handles)

9 **IN ntargets**
10 Number of elements in the *targets* array (integer)

11 **IN bo**
12 Pointer to [pmix_byte_object_t](#) containing the payload to be delivered (handle)

13 **IN directives**
14 Array of [pmix_info_t](#) structures (array of handles)

15 **IN ndirs**
16 Number of elements in the *directives* array (integer)

17 **IN directives**
18 Array of [pmix_info_t](#) structures (array of handles)

19 **IN cbfunc**
20 Callback function to be called when operation has been completed. ([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**, indicating that the request is being processed - result will be returned in the
29 provided *cbfunc*. Note that the library must not invoke the callback function prior to returning
30 from the API.
- 31 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
32 returned *success* - the *cbfunc* will *not* be called
- 33 • a PMIx error constant indicating either an error in the input or that the request was immediately
34 processed and failed - the *cbfunc* will *not* be called

35 The returned status code will be one of the following:

36 **PMIX_SUCCESS** The provided data has been accepted for transmission - it is not indicative of
37 the payload being delivered to any member of the provided *targets*

1 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function.
2 a PMIx error constant indicating the nature of the error

▼----- Required Attributes -----▼

3 The following attributes are required for PMIx libraries that support IO forwarding:

4 **PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (uint32_t)
5 The requested size of the server cache in bytes for each specified channel. By default, the
6 server is allowed (but not required) to drop all bytes received beyond the max size.

7 **PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (bool)
8 In an overflow situation, drop the oldest bytes to make room in the cache.

9 **PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (bool)
10 In an overflow situation, drop any new bytes received until room becomes available in the
11 cache (default).



▼----- Optional Attributes -----▼

12 The following attributes are optional for PMIx libraries that support IO forwarding:

13 **PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (uint32_t)
14 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
15 IO arrives. The library will execute the callback whenever the specified number of bytes
16 becomes available. Any remaining buffered data will be “flushed” upon call to deregister the
17 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 **Description**

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

▼----- Advice to users -----▼

24 Execution of the *cbfunc* callback function serves as notice that the PMIx library no longer requires
25 the caller to maintain the *bo* data object - it does *not* indicate delivery of the payload to the targets.
26 Use of **PMIX_RANK_WILDCARD** to specify all processes in a given namespace is supported but
27 should be used carefully due to bandwidth considerations.



CHAPTER 7

Job Management and Reporting

1 The job management APIs provide an application with the ability to orchestrate its operation in
2 partnership with the SMS. Members of this category include the
3 [PMIx_Allocation_request_nb](#), [PMIx_Job_control_nb](#), and
4 [PMIx_Process_monitor_nb](#) APIs.

5 7.1 Query

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

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

20 PMIx supports this use-case by defining an attribute key ([PMIX_TIME_REMAINING](#)) that can be
21 used with the [PMIx_Query_info_nb](#) interface to obtain the number of seconds remaining in
22 the current job allocation. Note that one could alternatively use the
23 [PMIx_Register_event_handler](#) API to register for an event indicating incipient job
24 termination, and then use the [PMIx_Job_control_nb](#) API to request that the host SMS
25 generate an event a specified amount of time prior to reaching the maximum run time. PMIx
26 provides such alternate methods as a means of maximizing the probability of a host system
27 supporting at least one method by which the application can obtain the desired service.

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


29 7.1.1 [PMIx_Resolve_peers](#)

30 Summary

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

1 **Format**
PMIx v1.0 

```
2 pmix_status_t  
3 PMIx_Resolve_peers(const char *nodename,  
4                   const pmix_namespace_t nspace,  
5                   pmix_proc_t **procs, size_t *nprocs)
```



- 6 **IN** **nodename**
7 Name of the node to query (string)
- 8 **IN** **nspace**
9 namespace (string)
- 10 **OUT** **procs**
11 Array of process structures (array of handles)
- 12 **OUT** **nprocs**
13 Number of elements in the *procs* array (integer)

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


15 **Description**
16 Given a *nodename*, return the array of processes within the specified *nspace* that are executing on
17 that node. If the *nspace* is **NULL**, then all processes on the node will be returned. If the specified
18 node does not currently host any processes, then the returned array will be **NULL**, and *nprocs* will
19 be 0. The caller is responsible for releasing the *procs* array when done with it. The
20 **PMIX_PROC_FREE** macro is provided for this purpose.

21 7.1.2 PMIx_Resolve_nodes

22 **Summary**
23 Return a list of nodes hosting processes within the given namespace.

24 **Format**
PMIx v1.0 

```
25 pmix_status_t  
26 PMIx_Resolve_nodes(const char *nnamespace, char **nodelist)
```



- 27 **IN** **nnamespace**
28 Namespace (string)
- 29 **OUT** **nodelist**
30 Comma-delimited list of nodenames (string)

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

1 **Description**
2 Given a *namespace*, return the list of nodes hosting processes within that namespace. The returned
3 string will contain a comma-delimited list of nodenames. The caller is responsible for releasing the
4 string when done with it.

5 7.1.3 PMIx_Query_info

6 **Summary**
7 Query information about the system in general.

8 **Format**

PMIx v4.0

```
▼----- C -----▼  
9       pmix_status_t  
10       PMIx_Query_info(pmix_query_t queries[], size_t nqueries,  
11                       pmix_info_t *info[], size_t *ninfo)  
▲----- C -----▲
```

12 **IN queries**
13 Array of query structures (array of handles)
14 **IN nqueries**
15 Number of elements in the *queries* array (integer)
16 **INOUT info**
17 Address where a pointer to an array of **pmix_info_t** containing the results of the query
18 can be returned (memory reference)
19 **INOUT ninfo**
20 Address where the number of elements in *info* can be returned (handle)

21 Returns one of the following:

- 22 • **PMIX_SUCCESS** All data has been returned
- 23 • **PMIX_ERR_NOT_FOUND** None of the requested data was available
- 24 • **PMIX_ERR_PARTIAL_SUCCESS** Some of the data has been returned
- 25 • **PMIX_ERR_NOT_SUPPORTED** The host RM does not support this function
- 26 • a non-zero PMIx error constant indicating a reason for the request's failure

▼----- Required Attributes -----▼

27 PMIx libraries that support this API are required to support the following attributes:

28 **PMIX_QUERY_REFRESH_CACHE** "pmix.qry.rfsh" (bool)
29 Retrieve updated information from server.
30 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)

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

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

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

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

12 Return information about the specified application. If information about an application other
13 than the one containing the requesting process is desired, then the attribute array must
14 contain a **PMIX_APPNUM** attribute identifying the desired target. Similarly, if information 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.procid" (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_NAMESPACE** "pmix.namespace" (char*)

26 Namespace of the job. Specifies the namespace of the process whose information is being
27 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
28 be accompanied by the **PMIX_RANK** attribute. Only required when the request is for
29 information on a specific process.

30 **PMIX_RANK** "pmix.rank" (pmix_rank_t)

31 Process rank within the job. Specifies the rank of the process whose information is being
32 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
33 be accompanied by the **PMIX_NAMESPACE** attribute. Only required when the request is for
34 information on a specific process.

35 **PMIX_QUERY_ATTRIBUTE_SUPPORT** "pmix.qry.attrs" (bool)

36 Query list of supported attributes for specified APIs

37 **PMIX_CLIENT_ATTRIBUTES** "pmix.client.attrs" (bool)

38 Request attributes supported by the PMIx client library

39 **PMIX_SERVER_ATTRIBUTES** "pmix.srvr.attrs" (bool)

1 Request attributes supported by the PMIx server library

2 **PMIX_HOST_ATTRIBUTES** "pmix.host.attrs" (bool)

3 Request attributes supported by the host environment

4 **PMIX_TOOL_ATTRIBUTES** "pmix.setup.env" (bool)

5 Request attributes supported by the PMIx tool library functions

6 Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NAMESPACE** or the
7 **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a
8 process identifier must apply to all keys in that **pmix_query_t**. Queries for information on
9 multiple specific processes therefore requires submitting multiple **pmix_query_t** structures,
10 each referencing one process.

11 PMIx libraries are not required to directly support any other attributes for this function. However,
12 any provided attributes must be passed to the host SMS daemon for processing, and the PMIx
13 library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client
14 process making the request.

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

18 **PMIX_PROCID** "pmix.procid" (**pmix_proc_t**)

19 Process identifier Specifies the process ID whose information is being requested - e.g., a
20 query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the
21 request is for information on a specific process.

22 **PMIX_NAMESPACE** "pmix.namespace" (**char***)

23 Namespace of the job. Specifies the namespace of the process whose information is being
24 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
25 be accompanied by the **PMIX_RANK** attribute. Only required when the request is for
26 information on a specific process.

27 **PMIX_RANK** "pmix.rank" (**pmix_rank_t**)

28 Process rank within the job. Specifies the rank of the process whose information is being
29 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
30 be accompanied by the **PMIX_NAMESPACE** attribute. Only required when the request is for
31 information on a specific process.

32 Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NAMESPACE** or the
33 **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a
34 process identifier must apply to all keys in that **pmix_query_t**. Queries for information on
35 multiple specific processes therefore requires submitting multiple **pmix_query_t** structures,
36 each referencing one process.

Optional Attributes

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

PMIX_QUERY_NAMESPACES "pmix.qry.ns" (char*)

Request a comma-delimited list of active namespaces.

PMIX_QUERY_JOB_STATUS "pmix.qry.jst" (pmix_status_t)

Status of a specified, currently executing job.

PMIX_QUERY_QUEUE_LIST "pmix.qry qlst" (char*)

Request a comma-delimited list of scheduler queues.

PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD)

Status of a specified scheduler queue.

PMIX_QUERY_PROC_TABLE "pmix.qry.phtable" (char*)

Input namespace of the job whose information is being requested returns (**pmix_data_array_t**) an array of **pmix_proc_info_t** .

PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)

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

PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)

Return a comma-delimited list of supported spawn attributes.

PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)

Return a comma-delimited list of supported debug attributes.

PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)

Return information on memory usage for the processes indicated in the qualifiers.

PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)

Report only average values for sampled information.

PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)

Report minimum and maximum values.

PMIX_QUERY_ALLOC_STATUS "pmix.query.alloc" (char*)

String identifier of the allocation whose status is being requested.

PMIX_TIME_REMAINING "pmix.time.remaining" (char*)

Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

PMIX_SERVER_URI "pmix.srvr.uri" (char*)

URI of the PMIx server to be contacted. Requests the URI of the specified PMIx server's PMIx connection. Defaults to requesting the information for the local PMIx server.

1 **PMIX_PROC_URI** "pmix.puri" (char*)

2 URI containing contact information for a given process. Requests the URI of the specified
3 PMIx server's out-of-band connection. Defaults to requesting the information for the local
4 PMIx server.



5 **Description**

6 Query information about the system in general. This can include a list of active namespaces, fabric
7 topology, etc. Also can be used to query node-specific info such as the list of peers executing on a
8 given node. We assume that the host RM will exercise appropriate access control on the
9 information.

10 The returned *status* indicates if requested data was found or not. The returned array of
11 **pmix_info_t** will contain each key that was provided and the corresponding value that was
12 found. Requests for keys that are not found will return the key paired with a value of type
13 **PMIX_UNDEF**. The caller is responsible for releasing the returned array.

14 **Advice to PMIx library implementers**

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



20 **7.1.4 PMIx_Query_info_nb**

21 **Summary**

22 Query information about the system in general.

1
PMIx v2.0

Format

C

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

C

- 5 **IN queries**
- 6 Array of query structures (array of handles)
- 7 **IN nqueries**
- 8 Number of elements in the *queries* array (integer)
- 9 **IN cbfunc**
- 10 Callback function `pmix_info_cbfunc_t` (function reference)
- 11 **IN cbdata**
- 12 Data to be passed to the callback function (memory reference)

13 Returns one of the following:

- 14 • **PMIX_SUCCESS** indicating that the request has been accepted for processing and the provided
- 15 callback function will be executed upon completion of the operation. Note that the library must
- 16 not invoke the callback function prior to returning from the API.
- 17 • 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

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

- 21 • **PMIX_SUCCESS** All data has been returned
- 22 • **PMIX_ERR_NOT_FOUND** None of the requested data was available
- 23 • **PMIX_ERR_PARTIAL_SUCCESS** Some of the data has been returned
- 24 • **PMIX_ERR_NOT_SUPPORTED** The host RM does not support this function
- 25 • a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

26 PMIx libraries that support this API are required to support the following attributes:

- 27 **PMIX_QUERY_REFRESH_CACHE** "`pmix.qry.rfsh`" (bool)
- 28 Retrieve updated information from server.
- 29 **PMIX_SESSION_INFO** "`pmix.ssn.info`" (bool)
- 30 Return information about the specified session. If information about a session other than the
- 31 one containing the requesting process is desired, then the attribute array must contain a
- 32 **PMIX_SESSION_ID** attribute identifying the desired target.

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

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

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

18 **PMIX_PROCID** "pmix.procid" (pmix_proc_t)
19 Process identifier Specifies the process ID whose information is being requested - e.g., a
20 query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the
21 request is for information on a specific process.

22 **PMIX_NAMESPACE** "pmix.namespace" (char*)
23 Namespace of the job. Specifies the namespace of the process whose information is being
24 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
25 be accompanied by the **PMIX_RANK** attribute. Only required when the request is for
26 information on a specific process.

27 **PMIX_RANK** "pmix.rank" (pmix_rank_t)
28 Process rank within the job. Specifies the rank of the process whose information is being
29 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
30 be accompanied by the **PMIX_NAMESPACE** attribute. Only required when the request is for
31 information on a specific process.

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

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

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

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

1 **PMIX_TOOL_ATTRIBUTES** "pmix.setup.env" (bool)

2 Request attributes supported by the PMIx tool library functions

3 Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NAMESPACE** or the
4 **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a
5 process identifier must apply to all keys in that **pmix_query_t**. Queries for information on
6 multiple specific processes therefore requires submitting multiple **pmix_query_t** structures,
7 each referencing one process.

8 PMIx libraries are not required to directly support any other attributes for this function. However,
9 any provided attributes must be passed to the host SMS daemon for processing, and the PMIx
10 library is *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client
11 process making the request.

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

15 **PMIX_PROCID** "pmix.procid" (**pmix_proc_t**)

16 Process identifier Specifies the process ID whose information is being requested - e.g., a
17 query asking for the **PMIX_LOCAL_RANK** of a specified process. Only required when the
18 request is for information on a specific process.

19 **PMIX_NAMESPACE** "pmix.namespace" (**char***)

20 Namespace of the job. Specifies the namespace of the process whose information is being
21 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
22 be accompanied by the **PMIX_RANK** attribute. Only required when the request is for
23 information on a specific process.

24 **PMIX_RANK** "pmix.rank" (**pmix_rank_t**)

25 Process rank within the job. Specifies the rank of the process whose information is being
26 requested - e.g., a query asking for the **PMIX_LOCAL_RANK** of a specified process. Must
27 be accompanied by the **PMIX_NAMESPACE** attribute. Only required when the request is for
28 information on a specific process.

29 Note that inclusion of the **PMIX_PROCID** directive and either the **PMIX_NAMESPACE** or the
30 **PMIX_RANK** attribute will return a **PMIX_ERR_BAD_PARAM** result, and that the inclusion of a
31 process identifier must apply to all keys in that **pmix_query_t**. Queries for information on
32 multiple specific processes therefore requires submitting multiple **pmix_query_t** structures,
33 each referencing one process.

Optional Attributes

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

35 **PMIX_QUERY_NAMESPACES** "pmix.qry.ns" (**char***)

36 Request a comma-delimited list of active namespaces.

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

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

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

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

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

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

16 **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
17 Return a comma-delimited list of supported debug attributes.

18 **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)
19 Return information on memory usage for the processes indicated in the qualifiers.

20 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
21 Report only average values for sampled information.

22 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)
23 Report minimum and maximum values.

24 **PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (char*)
25 String identifier of the allocation whose status is being requested.

26 **PMIX_TIME_REMAINING** "pmix.time.remaining" (char*)
27 Query number of seconds (uint32_t) remaining in allocation for the specified namespace.
28

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

32 **PMIX_PROC_URI** "pmix.puri" (char*)
33 URI containing contact information for a given process. Requests the URI of the specified
34 PMIx server's out-of-band connection. Defaults to requesting the information for the local
35 PMIx server.

▲-----▲

Description

Non-blocking form of the `PMIx_Query_info` API

7.1.4.1 Using `PMIx_Get` vs `PMIx_Query_info`

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

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

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

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

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

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

7.1.4.2 Accessing attribute support information

Information as to attributes supported by either the PMIx implementation or its host environment can be obtained via the `PMIx_Query_info_nb` API. The `PMIX_QUERY_ATTRIBUTE_SUPPORT` attribute must be listed as the first entry in the *keys* field of the `pmix_query_t` structure, followed by the name of the function whose attribute support is being requested - support for multiple functions can be requested simultaneously by simply adding

1 the function names to the array of *keys*. Function names *must* be given as user-level API names -
2 e.g., “PMIx_Get”, “PMIx_server_setup_application”, or “PMIx_tool_connect_to_server”.

3 The desired levels (see 14.4.33) of attribute support are provided as qualifiers. Multiple levels can
4 be requested simultaneously by simply adding elements to the *qualifiers* array. Each qualifier
5 should contain the desired level attribute with the boolean value set to indicate whether or not that
6 level is to be included in the returned information. Failure to provide any levels is equivalent to a
7 request for all levels.

8 Unlike other queries, queries for attribute support can result in the number of returned
9 `pmix_info_t` structures being different from the number of queries. Each element in the
10 returned array will correspond to a pair of specified attribute level and function in the query, where
11 the *key* is the function and the *value* contains a `pmix_data_array_t` of `pmix_info_t`.
12 Each element of the array is marked by a *key* indicating the requested attribute *level* with a *value*
13 composed of a `pmix_data_array_t` of `pmix_regattr_t`, each describing a supported
14 attribute for that function, as illustrated in Fig. 7.1 below where the requestor asked for supported
15 attributes of `PMIx_Get` at the *client* and *server* levels, plus attributes of
16 `PMIx_Allocation_request` at all levels:

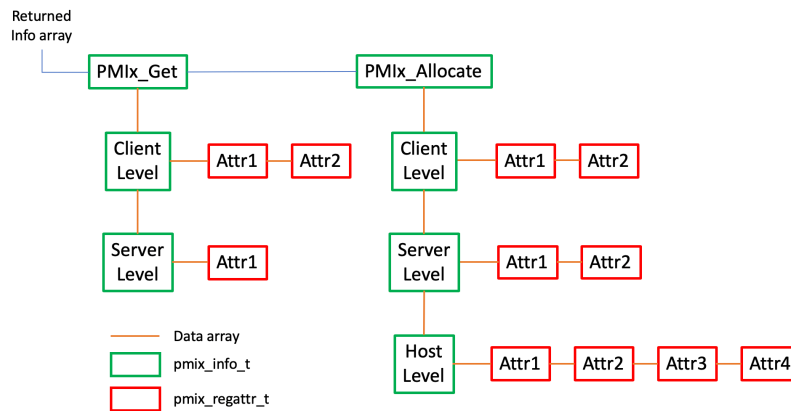


Figure 7.1.: Returned information hierarchy for attribute support request

17 The array of returned structures, and their child arrays, are subject to the return rules for the
18 `PMIx_Query_info_nb` API. For example, a request for supported attributes of the `PMIx_Get`
19 function that includes the *host* level will return values for the *client* and *server* levels, plus an array
20 element with a *key* of `PMIX_HOST_ATTRIBUTES` and a value type of `PMIX_UNDEF` indicating
21 that no attributes are supported at that level.

22 7.2 Allocation Requests

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

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

8 7.2.1 PMIx_Allocation_request

9 Summary

10 Request an allocation operation from the host resource manager.

11 Format

PMIx v3.0

C

```
12 pmix_status_t
13 PMIx_Allocation_request(pmix_alloc_directive_t directive,
14                        pmix_info_t info[], size_t ninfo,
15                        pmix_info_t *results[], size_t *nresults);
```

C

16 **IN directive**

17 Allocation directive (handle)

18 **IN info**

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

20 **IN ninfo**

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

22 **INOUT results**

23 Address where a pointer to an array of `pmix_info_t` containing the results of the request

24 can be returned (memory reference)

25 **INOUT nresults**

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

27 Returns one of the following:

- 28 • **PMIX_SUCCESS**, indicating that the request was processed and returned *success*
- 29 • a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)

User-provided string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)

The number of nodes.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)

Number of cpus.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)

Time in seconds.

Optional Attributes

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

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

Regular expression of the specific nodes.

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

Regular expression of the number of cpus for each node.

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

Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)

Number of Megabytes.

PMIX_ALLOC_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*)

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 Mbits/sec.

16 `PMIX_ALLOC_FABRIC_QOS` "pmix.alloc.netqos" (char*)
17 Quality of service level.

18 `PMIX_ALLOC_FABRIC_TYPE` "pmix.alloc.nettype" (char*)
19 Type of desired transport (e.g., "tcp", "udp")

20 `PMIX_ALLOC_FABRIC_PLANE` "pmix.alloc.netplane" (char*)
21 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

22 `PMIX_ALLOC_FABRIC_ENDPTS` "pmix.alloc.endpts" (size_t)
23 Number of endpoints to allocate per process

24 `PMIX_ALLOC_FABRIC_ENDPTS_NODE` "pmix.alloc.endpts.nd" (size_t)
25 Number of endpoints to allocate per node

26 `PMIX_ALLOC_FABRIC_SEC_KEY` "pmix.alloc.nsec" (pmix_byte_object_t)
27 Fabric security key



28 Description

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

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

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

If successful, the returned results for a request for additional resources must include the host resource manager’s identifier (`PMIX_ALLOC_ID`) that the requester can use to specify the resources in, for example, a call to `PMIx_Spawn` .

7.2.2 `PMIx_Allocation_request_nb`

Summary

Request an allocation operation from the host resource manager.

Format

PMIx v2.0

```

pmix_status_t
PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,
                           pmix_info_t info[], size_t ninfo,
                           pmix_info_cbfunc_t cbfunc, void *cbdata);

```

- IN directive**
Allocation directive (handle)
- IN info**
Array of `pmix_info_t` structures (array of handles)
- IN ninfo**
Number of elements in the *info* array (integer)
- IN cbfunc**
Callback function `pmix_info_cbfunc_t` (function reference)
- IN cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

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

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)

User-provided string identifier for this allocation request which can later be used to query status of the request.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)

The number of nodes.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)

Number of cpus.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)

Time in seconds.

Optional Attributes

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

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

Regular expression of the specific nodes.

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

Regular expression of the number of cpus for each node.

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

Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)

Number of Megabytes.

PMIX_ALLOC_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*)

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 Mbits/sec.

16 `PMIX_ALLOC_FABRIC_QOS` "pmix.alloc.netqos" (char*)
17 Quality of service level.

18 `PMIX_ALLOC_FABRIC_TYPE` "pmix.alloc.nettype" (char*)
19 Type of desired transport (e.g., "tcp", "udp")

20 `PMIX_ALLOC_FABRIC_PLANE` "pmix.alloc.netplane" (char*)
21 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

22 `PMIX_ALLOC_FABRIC_ENDPTS` "pmix.alloc.endpts" (size_t)
23 Number of endpoints to allocate per process

24 `PMIX_ALLOC_FABRIC_ENDPTS_NODE` "pmix.alloc.endpts.nd" (size_t)
25 Number of endpoints to allocate per node

26 `PMIX_ALLOC_FABRIC_SEC_KEY` "pmix.alloc.nsec" (pmix_byte_object_t)
27 Fabric security key



28 Description

29 Non-blocking form of the `PMIx_Allocation_request` API.

1 7.3 Job Control

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

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

23 7.3.1 PMIx_Job_control

24 Summary

25 Request a job control action.

26 Format

PMIx v3.0

27 `pmix_status_t`

```
28 PMIx_Job_control(const pmix_proc_t targets[], size_t ntargets,  
29                  const pmix_info_t directives[], size_t ndirs,  
30                  pmix_info_t *results[], size_t *nresults)
```

31 **IN** `targets`

32 Array of proc structures (array of handles)

33 **IN** `ntargets`

34 Number of element in the *targets* array (integer)

35 **IN** `directives`

36 Array of info structures (array of handles)

1 **IN ndirs**
2 Number of element in the *directives* 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 by the host environment and
10 returned *success*. Details of the result will be returned in the *results* array
- 11 • a PMIx error constant indicating either an error in the input or that the request was refused

▼----- Required Attributes -----▼

12 PMIx libraries are not required to directly support any attributes for this function. However, any
13 provided attributes must be passed to the host SMS daemon for processing, and the PMIx library is
14 *required* to add the **PMIX_USERID** and the **PMIX_GRPID** attributes of the client process making
15 the request.

17 Host environments that implement support for this operation are required to support the following
18 attributes:

- 19 **PMIX_JOB_CTRL_ID** "pmix.jctrl.id" (**char***)
20 Provide a string identifier for this request. The user can provide an identifier for the
21 requested operation, thus allowing them to later request status of the operation or to
22 terminate it. The host, therefore, shall track it with the request for future reference.
- 23 **PMIX_JOB_CTRL_PAUSE** "pmix.jctrl.pause" (**bool**)
24 Pause the specified processes.
- 25 **PMIX_JOB_CTRL_RESUME** "pmix.jctrl.resume" (**bool**)
26 Resume ("un-pause") the specified processes.
- 27 **PMIX_JOB_CTRL_KILL** "pmix.jctrl.kill" (**bool**)
28 Forcibly terminate the specified processes and cleanup.
- 29 **PMIX_JOB_CTRL_SIGNAL** "pmix.jctrl.sig" (**int**)
30 Send given signal to specified processes.
- 31 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (**bool**)
32 Politely terminate the specified processes.
- 33 **PMIX_REGISTER_CLEANUP** "pmix.reg.cleanup" (**char***)
34 Comma-delimited list of files to be removed upon process termination
- 35 **PMIX_REGISTER_CLEANUP_DIR** "pmix.reg.cleanupdir" (**char***)

1 Comma-delimited list of directories to be removed upon process termination

2 **PMIX_CLEANUP_RECURSIVE** "pmix.clnup.recurse" (bool)

3 Recursively cleanup all subdirectories under the specified one(s)

4 **PMIX_CLEANUP_EMPTY** "pmix.clnup.empty" (bool)

5 Only remove empty subdirectories

6 **PMIX_CLEANUP_IGNORE** "pmix.clnup.ignore" (char*)

7 Comma-delimited list of filenames that are not to be removed

8 **PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)

9 When recursively cleaning subdirectories, do not remove the top-level directory (the one
10 given in the cleanup request)



▼----- Optional Attributes -----▼

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

12 **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)

13 Cancel the specified request - the provided request ID must match the

14 **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of
15 **NULL** implies cancel all requests from this requestor.

16 **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)

17 Restart the specified processes using the given checkpoint ID.

18 **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)

19 Checkpoint the specified processes and assign the given ID to it.

20 **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)

21 Use event notification to trigger a process checkpoint.

22 **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)

23 Use the given signal to trigger a process checkpoint.

24 **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)

25 Time in seconds to wait for a checkpoint to complete.

26 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**

27 "pmix.jctrl.ckmethod" (pmix_data_array_t)

28 Array of **pmix_info_t** declaring each method and value supported by this application.

29 **PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)

30 Regular expression identifying nodes that are to be provisioned.

31 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvning" (char*)

32 Name of the image that is to be provisioned.

33 **PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

1 Indicate that the job can be pre-empted.



2 Description

3 Request a job control action. The *targets* array identifies the processes to which the requested job
4 control action is to be applied. A **NULL** value can be used to indicate all processes in the caller's
5 namespace. The use of **PMIX_RANK_WILDCARD** can also be used to indicate that all processes in
6 the given namespace are to be included.

7 The directives are provided as **pmix_info_t** structures in the *directives* array. The callback
8 function provides a *status* to indicate whether or not the request was granted, and to provide some
9 information as to the reason for any denial in the **pmix_info_cbfunc_t** array of
10 **pmix_info_t** structures.

11 7.3.2 PMIx_Job_control_nb

12 Summary

13 Request a job control action.

14 Format

PMIx v2.0

```
15 pmix_status_t  
16 PMIx_Job_control_nb(const pmix_proc_t targets[], size_t ntargets,  
17                    const pmix_info_t directives[], size_t ndirs,  
18                    pmix_info_cbfunc_t cbfunc, void *cbdata)
```

19 IN targets

20 Array of proc structures (array of handles)

21 IN ntargets

22 Number of element in the *targets* array (integer)

23 IN directives

24 Array of info structures (array of handles)

25 IN ndirs

26 Number of element in the *directives* array (integer)

27 IN cbfunc

28 Callback function **pmix_info_cbfunc_t** (function reference)

29 IN cbdata

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

31 Returns one of the following:

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

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

Required Attributes

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

Host environments that implement support for this operation are required to support the following attributes:

PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)

Provide a string identifier for this request. The user can provide an identifier for the requested operation, thus allowing them to later request status of the operation or to terminate it. The host, therefore, shall track it with the request for future reference.

PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)

Pause the specified processes.

PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)

Resume ("un-pause") the specified processes.

PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)

Forcibly terminate the specified processes and cleanup.

PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)

Send given signal to specified processes.

PMIX_JOB_CTRL_TERMINATE "pmix.jctrl.term" (bool)

Politely terminate the specified processes.

PMIX_REGISTER_CLEANUP "pmix.reg.cleanup" (char*)

Comma-delimited list of files to be removed upon process termination

PMIX_REGISTER_CLEANUP_DIR "pmix.reg.cleanupdir" (char*)

Comma-delimited list of directories to be removed upon process termination

PMIX_CLEANUP_RECURSIVE "pmix.clnup.recurse" (bool)

Recursively cleanup all subdirectories under the specified one(s)

PMIX_CLEANUP_EMPTY "pmix.clnup.empty" (bool)

Only remove empty subdirectories

PMIX_CLEANUP_IGNORE "pmix.clnup.ignore" (char*)

1 Comma-delimited list of filenames that are not to be removed

2 **PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)

3 When recursively cleaning subdirectories, do not remove the top-level directory (the one
4 given in the cleanup request)



▼----- Optional Attributes -----▼

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

6 **PMIX_JOB_CTRL_CANCEL** "pmix.jctrl.cancel" (char*)

7 Cancel the specified request - the provided request ID must match the

8 **PMIX_JOB_CTRL_ID** provided to a previous call to **PMIx_Job_control** . An ID of

9 **NULL** implies cancel all requests from this requestor.

10 **PMIX_JOB_CTRL_RESTART** "pmix.jctrl.restart" (char*)

11 Restart the specified processes using the given checkpoint ID.

12 **PMIX_JOB_CTRL_CHECKPOINT** "pmix.jctrl.ckpt" (char*)

13 Checkpoint the specified processes and assign the given ID to it.

14 **PMIX_JOB_CTRL_CHECKPOINT_EVENT** "pmix.jctrl.ckptev" (bool)

15 Use event notification to trigger a process checkpoint.

16 **PMIX_JOB_CTRL_CHECKPOINT_SIGNAL** "pmix.jctrl.ckptsig" (int)

17 Use the given signal to trigger a process checkpoint.

18 **PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT** "pmix.jctrl.ckptsig" (int)

19 Time in seconds to wait for a checkpoint to complete.

20 **PMIX_JOB_CTRL_CHECKPOINT_METHOD**

21 "pmix.jctrl.ckmethod" (pmix_data_array_t)

22 Array of **pmix_info_t** declaring each method and value supported by this application.

23 **PMIX_JOB_CTRL_PROVISION** "pmix.jctrl.pvn" (char*)

24 Regular expression identifying nodes that are to be provisioned.

25 **PMIX_JOB_CTRL_PROVISION_IMAGE** "pmix.jctrl.pvning" (char*)

26 Name of the image that is to be provisioned.

27 **PMIX_JOB_CTRL_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)

28 Indicate that the job can be pre-empted.



Description

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

The directives are provided as `pmix_info_t` structures in the *directives* array. The callback function provides a *status* to indicate whether or not the request was granted, and to provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of `pmix_info_t` structures.

7.4 Process and Job Monitoring

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

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

7.4.1 `PMIx_Process_monitor`

Summary

Request that application processes be monitored.

Format

PMIx v3.0

```
pmix_status_t
PMIx_Process_monitor(const pmix_info_t *monitor, pmix_status_t error,
                    const pmix_info_t directives[], size_t ndirs,
                    pmix_info_t *results[], size_t *nresults)
```

IN `monitor`
info (handle)

IN `error`
status (integer)

1 **IN directives**
 2 Array of info structures (array of handles)
 3 **IN ndirs**
 4 Number of elements in the *directives* array (integer)
 5 **INOUT results**
 6 Address where a pointer to an array of `pmix_info_t` containing the results of the request
 7 can be returned (memory reference)
 8 **INOUT nresults**
 9 Address where the number of elements in *results* can be returned (handle)

10 Returns one of the following:

- 11 • `PMIX_SUCCESS`, indicating that the request was processed and returned *success*. Details of the
 12 result will be returned in the *results* array
- 13 • a PMIx error constant indicating either an error in the input or that the request was refused

▼----- Optional Attributes -----▼

14 The following attributes may be implemented by a PMIx library or by the host environment. If
 15 supported by the PMIx server library, then the library must not pass the supported attributes to the
 16 host environment. All attributes not directly supported by the server library must be passed to the
 17 host environment if it supports this operation, and the library is *required* to add the
 18 `PMIX_USERID` and the `PMIX_GRPID` attributes of the requesting process:

- 19 `PMIX_MONITOR_ID` "pmix.monitor.id" (char*)
 20 Provide a string identifier for this request.
- 21 `PMIX_MONITOR_CANCEL` "pmix.monitor.cancel" (char*)
 22 Identifier to be canceled (`NULL` means cancel all monitoring for this process).
- 23 `PMIX_MONITOR_APP_CONTROL` "pmix.monitor.appctrl" (bool)
 24 The application desires to control the response to a monitoring event.
- 25 `PMIX_MONITOR_HEARTBEAT` "pmix.monitor.mbeat" (void)
 26 Register to have the PMIx server monitor the requestor for heartbeats.
- 27 `PMIX_MONITOR_HEARTBEAT_TIME` "pmix.monitor.btime" (uint32_t)
 28 Time in seconds before declaring heartbeat missed.
- 29 `PMIX_MONITOR_HEARTBEAT_DROPS` "pmix.monitor.bdrop" (uint32_t)
 30 Number of heartbeats that can be missed before generating the event.
- 31 `PMIX_MONITOR_FILE` "pmix.monitor.fmon" (char*)
 32 Register to monitor file for signs of life.
- 33 `PMIX_MONITOR_FILE_SIZE` "pmix.monitor.fsize" (bool)
 34 Monitor size of given file is growing to determine if the application is running.
- 35 `PMIX_MONITOR_FILE_ACCESS` "pmix.monitor.faccess" (char*)

1 Monitor time since last access of given file to determine if the application is running.

2 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)

3 Monitor time since last modified of given file to determine if the application is running.

4 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)

5 Time in seconds between checking the file.

6 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)

7 Number of file checks that can be missed before generating the event.



8 **Description**

9 Request that application processes be monitored via several possible methods. For example, that
 10 the server monitor this process for periodic heartbeats as an indication that the process has not
 11 become “wedged”. When a monitor detects the specified alarm condition, it will generate an event
 12 notification using the provided error code and passing along any available relevant information. It
 13 is up to the caller to register a corresponding event handler.

14 The *monitor* argument is an attribute indicating the type of monitor being requested. For example,
 15 **PMIX_MONITOR_FILE** to indicate that the requestor is asking that a file be monitored.

16 The *error* argument is the status code to be used when generating an event notification alerting that
 17 the monitor has been triggered. The range of the notification defaults to
 18 **PMIX_RANGE_NAMESPACE**. This can be changed by providing a **PMIX_RANGE** directive.

19 The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency
 20 of checking to be done

21 **7.4.2 PMIx_Process_monitor_nb**

22 **Summary**

23 Request that application processes be monitored.

24 **Format**

PMIx v2.0



25 **pmix_status_t**
 26 **PMIx_Process_monitor_nb**(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 monitor**
 2 info (handle)
 3 **IN error**
 4 status (integer)
 5 **IN directives**
 6 Array of info structures (array of handles)
 7 **IN ndirs**
 8 Number of elements in the *directives* array (integer)
 9 **IN cbfunc**
 10 Callback function `pmix_info_cbfunc_t` (function reference)
 11 **IN cbdata**
 12 Data to be passed to the callback function (memory reference)

13 Returns one of the following:

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

Optional Attributes

21 The following attributes may be implemented by a PMIx library or by the host environment. If
 22 supported by the PMIx server library, then the library must not pass the supported attributes to the
 23 host environment. All attributes not directly supported by the server library must be passed to the
 24 host environment if it supports this operation, and the library is *required* to add the
 25 **PMIX_USERID** and the **PMIX_GRPID** attributes of the requesting process:

26 **PMIX_MONITOR_ID** "pmix.monitor.id" (**char***)
 27 Provide a string identifier for this request.

28 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (**char***)
 29 Identifier to be canceled (**NULL** means cancel all monitoring for this process).

30 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (**bool**)
 31 The application desires to control the response to a monitoring event.

32 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (**void**)
 33 Register to have the PMIx server monitor the requestor for heartbeats.

34 **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (**uint32_t**)
 35 Time in seconds before declaring heartbeat missed.

1 **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)
 2 Number of heartbeats that can be missed before generating the event.

3 **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)
 4 Register to monitor file for signs of life.

5 **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)
 6 Monitor size of given file is growing to determine if the application is running.

7 **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)
 8 Monitor time since last access of given file to determine if the application is running.

9 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)
 10 Monitor time since last modified of given file to determine if the application is running.

11 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)
 12 Time in seconds between checking the file.

13 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)
 14 Number of file checks that can be missed before generating the event.

▲-----▲

15 **Description**

16 Non-blocking form of the [PMIx_Process_monitor](#) API. The *cbfunc* function provides a
 17 *status* to indicate whether or not the request was granted, and to provide some information as to the
 18 reason for any denial in the [pmix_info_cbfunc_t](#) array of [pmix_info_t](#) structures.

19 **7.4.3 PMIx_Heartbeat**

20 **Summary**

21 Send a heartbeat to the PMIx server library

22 **Format**

PMIx v2.0

23 **PMIx_Heartbeat** (void)



24 **Description**

25 A simplified macro wrapping [PMIx_Process_monitor_nb](#) that sends a heartbeat to the
 26 PMIx server library.

1 7.5 Logging

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

6 7.5.1 PMIx_Log

7 Summary

8 Log data to a data service.

9 Format

PMIx v3.0

C

```
10 pmix_status_t  
11 PMIx_Log(const pmix_info_t data[], size_t ndata,  
12          const pmix_info_t directives[], size_t ndirs)
```

C

13 IN data

14 Array of info structures (array of handles)

15 IN ndata

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

17 IN directives

18 Array of info structures (array of handles)

19 IN ndirs

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

21 Return codes are one of the following:

22 **PMIX_SUCCESS** The logging request was successful.

23 **PMIX_ERR_BAD_PARAM** The logging request contains at least one incorrect entry.

24 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation or host environment does not
25 support this function.

Required Attributes

26 If the PMIx library does not itself perform this operation, then it is required to pass any attributes
27 provided by the client to the host environment for processing. In addition, it must include the
28 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.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

- PMIX_LOG_STDERR** "pmix.log.stderr" (char*)
Log string to **stderr**.
- PMIX_LOG_STDOUT** "pmix.log.stdout" (char*)
Log string to **stdout**.
- PMIX_LOG_SYSLOG** "pmix.log.syslog" (char*)
Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog
- PMIX_LOG_LOCAL_SYSLOG** "pmix.log.lsys" (char*)
Log data to local syslog. Defaults to **ERROR** priority.
- PMIX_LOG_GLOBAL_SYSLOG** "pmix.log.gsys" (char*)
Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.
- PMIX_LOG_SYSLOG_PRI** "pmix.log.syspri" (int)
Syslog priority level
- PMIX_LOG_ONCE** "pmix.log.once" (bool)
Only log this once with whichever channel can first support it, taking the channels in priority order

▲-----▲
▼-----▼ **Optional Attributes** -----▼

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

- PMIX_LOG_SOURCE** "pmix.log.source" (pmix_proc_t*)
ID of source of the log request
- PMIX_LOG_TIMESTAMP** "pmix.log.tstamp" (time_t)
Timestamp for log report
- PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstamp" (bool)
Generate timestamp for log
- PMIX_LOG_TAG_OUTPUT** "pmix.log.tag" (bool)
Label the output stream with the channel name (e.g., "stdout")
- PMIX_LOG_TIMESTAMP_OUTPUT** "pmix.log.tsout" (bool)
Print timestamp in output string
- PMIX_LOG_XML_OUTPUT** "pmix.log.xml" (bool)
Print the output stream in XML format

1 **PMIX_LOG_EMAIL** "pmix.log.email" (pmix_data_array_t)
2 Log via email based on **pmix_info_t** containing directives.
3 **PMIX_LOG_EMAIL_ADDR** "pmix.log.emaddr" (char*)
4 Comma-delimited list of email addresses that are to receive the message.
5 **PMIX_LOG_EMAIL_SUBJECT** "pmix.log.emsub" (char*)
6 Subject line for email.
7 **PMIX_LOG_EMAIL_MSG** "pmix.log.emmsg" (char*)
8 Message to be included in email.
9 **PMIX_LOG_JOB_RECORD** "pmix.log.jrec" (bool)
10 Log the provided information to the host environment's job record
11 **PMIX_LOG_GLOBAL_DATASTORE** "pmix.log.gstore" (bool)
12 Store the log data in a global data store (e.g., database)



Description

Log data subject to the services offered by the host environment. The data to be logged is provided in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel.

Advice to users

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



21 7.5.2 PMIx_Log_nb

22 Summary

23 Log data to a data service.

1
PMIx v2.0

Format

C

```

2 pmix_status_t
3 PMIx_Log_nb(const pmix_info_t data[], size_t ndata,
4             const pmix_info_t directives[], size_t ndirs,
5             pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN data**
Array of info structures (array of handles)
- 8 **IN ndata**
Number of elements in the *data* array (**size_t**)
- 10 **IN directives**
Array of info structures (array of handles)
- 12 **IN ndirs**
Number of elements in the *directives* array (**size_t**)
- 14 **IN cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)
- 16 **IN cbdata**
Data to be passed to the callback function (memory reference)

18 Return codes are one of the following:

- 19 **PMIX_SUCCESS** The logging request is valid and is being processed. The resulting status from
20 the operation will be provided in the callback function. Note that the library must not invoke
21 the callback function prior to returning from the API.
- 22 **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
23 returned *success* - the *cbfunc* will *not* be called
- 24 **PMIX_ERR_BAD_PARAM** The logging request contains at least one incorrect entry that prevents
25 it from being processed. The callback function will not be called.
- 26 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support this function. The
27 callback function will not be called.

Required Attributes

28 If the PMIx library does not itself perform this operation, then it is required to pass any attributes
29 provided by the client to the host environment for processing. In addition, it must include the
30 following attributes in the passed *info* array:

- 31 **PMIX_USERID** "pmix.euid" (**uint32_t**)
32 Effective user id.
- 33 **PMIX_GRPID** "pmix.egid" (**uint32_t**)
34 Effective group id.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

Host environments or PMIx libraries that implement support for this operation are required to support the following attributes:

- PMIX_LOG_STDERR** "pmix.log.stderr" (char*)
Log string to **stderr**.
- PMIX_LOG_STDOUT** "pmix.log.stdout" (char*)
Log string to **stdout**.
- PMIX_LOG_SYSLOG** "pmix.log.syslog" (char*)
Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog
- PMIX_LOG_LOCAL_SYSLOG** "pmix.log.lsys" (char*)
Log data to local syslog. Defaults to **ERROR** priority.
- PMIX_LOG_GLOBAL_SYSLOG** "pmix.log.gsys" (char*)
Forward data to system "gateway" and log msg to that syslog Defaults to **ERROR** priority.
- PMIX_LOG_SYSLOG_PRI** "pmix.log.syspri" (int)
Syslog priority level
- PMIX_LOG_ONCE** "pmix.log.once" (bool)
Only log this once with whichever channel can first support it, taking the channels in priority order

▲-----▲
▼-----▼ **Optional Attributes** -----▼

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

- PMIX_LOG_SOURCE** "pmix.log.source" (pmix_proc_t*)
ID of source of the log request
- PMIX_LOG_TIMESTAMP** "pmix.log.tstamp" (time_t)
Timestamp for log report
- PMIX_LOG_GENERATE_TIMESTAMP** "pmix.log.gtstamp" (bool)
Generate timestamp for log
- PMIX_LOG_TAG_OUTPUT** "pmix.log.tag" (bool)
Label the output stream with the channel name (e.g., "stdout")
- PMIX_LOG_TIMESTAMP_OUTPUT** "pmix.log.tsout" (bool)
Print timestamp in output string
- PMIX_LOG_XML_OUTPUT** "pmix.log.xml" (bool)
Print the output stream in XML format

1 **PMIX_LOG_EMAIL** "pmix.log.email" (pmix_data_array_t)
2 Log via email based on **pmix_info_t** containing directives.
3 **PMIX_LOG_EMAIL_ADDR** "pmix.log.emaddr" (char*)
4 Comma-delimited list of email addresses that are to receive the message.
5 **PMIX_LOG_EMAIL_SUBJECT** "pmix.log.emsub" (char*)
6 Subject line for email.
7 **PMIX_LOG_EMAIL_MSG** "pmix.log.emmsg" (char*)
8 Message to be included in email.
9 **PMIX_LOG_JOB_RECORD** "pmix.log.jrec" (bool)
10 Log the provided information to the host environment's job record
11 **PMIX_LOG_GLOBAL_DATASTORE** "pmix.log.gstore" (bool)
12 Store the log data in a global data store (e.g., database)



Description

13 Log data subject to the services offered by the host environment. The data to be logged is provided
14 in the *data* array. The (optional) *directives* can be used to direct the choice of logging channel. The
15 callback function will be executed when the log operation has been completed. The *data* and
16 *directives* arrays must be maintained until the callback is provided.
17

Advice to users

18 It is strongly recommended that the **PMIx_Log_nb** API not be used by applications for streaming
19 data as it is not a “performant” transport and can perturb the application since it involves the local
20 PMIx server and host SMS daemon. Note that a return of **PMIX_SUCCESS** only denotes that the
21 data was successfully handed to the appropriate system call (for local channels) or the host
22 environment and does not indicate receipt at the final destination.



CHAPTER 8

Event Notification

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

4 8.1 Notification and Management

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

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

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

▼ Advice to PMIx library implementers ▼

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

Advice to users

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

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

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

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

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

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

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

Advice to users

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

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

17 8.1.1 PMIx_Register_event_handler

18 Summary

19 Register an event handler

20 Format

PMIx v2.0

C

```
21 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

27 **IN** `codes`
28 Array of status codes (array of [pmix_status_t](#))
29 **IN** `ncodes`
30 Number of elements in the `codes` array (**size_t**)
31 **IN** `info`
32 Array of info structures (array of handles)

1 **IN ninfo**
 2 Number of elements in the *info* array (**size_t**)
 3 **IN evhdlr**
 4 Event handler to be called **pmix_notification_fn_t** (function reference)
 5 **IN cbfunc**
 6 Callback function **pmix_evhdlr_reg_cbfunc_t** (function reference)
 7 **IN cbdata**
 8 Data to be passed to the cbfunc callback function (memory reference)

9 If *cbfunc* is **NULL**, the function call will be treated as a *blocking* call. In this case, the returned
 10 status will be either (a) the event handler reference identifier if the value is greater than or equal to
 11 zero, or (b) a negative error code indicative of the reason for the failure.

12 If the *cbfunc* is non-**NULL**, the function call will be treated as a *non-blocking* call and will return
 13 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.

20 The callback function must not be executed prior to returning from the API, and no events
 21 corresponding to this registration may be delivered prior to the completion of the registration
 22 callback function (*cbfunc*).

▼----- Required Attributes -----▼

23 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.evfirst" (**bool**)
 27 Invoke this event handler before any other handlers.
- 28 **PMIX_EVENT_HDLR_LAST** "pmix.evlast" (**bool**)
 29 Invoke this event handler after all other handlers have been called.
- 30 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY** "pmix.evfirstcat" (**bool**)
 31 Invoke this event handler before any other handlers in this category.
- 32 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY** "pmix.evlastcat" (**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.evaffected" (**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.evtimeout" (**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.

▼ **Advice to users** ▼

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

▼ **Advice to users** ▼

10 As previously stated, upon completing its work, and prior to returning, each handler *must* call the
11 event handler completion function provided when it was invoked (including a status code plus any
12 information to be passed to later handlers) so that the chain can continue being progressed. An
13 event handler can terminate all further progress along the chain by passing the
14 `PMIX_EVENT_ACTION_COMPLETE` status to the completion callback function. Note that the
15 parameters passed to the event handler (e.g., the *info* and *results* arrays) will cease to be valid once
16 the completion function has been called - thus, any information in the incoming parameters that
17 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.

1
PMIx v2.0

Format

C

```
2 pmix_status_t
3 PMIx_Deregister_event_handler(size_t evhdlr_ref,
4                               pmix_op_cbfunc_t cbfunc,
5                               void *cbdata);
```

C

- 6 **IN** `evhdlr_ref`
Event handler ID returned by registration (`size_t`)
- 8 **IN** `cbfunc`
Callback function to be executed upon completion of operation `pmix_op_cbfunc_t`
(function reference)
- 11 **IN** `cbdata`
Data to be passed to the `cbfunc` callback function (memory reference)

13 If `cbfunc` is **NULL**, the function will be treated as a *blocking* call and the result of the operation
14 returned in the status code.

15 If `cbfunc` is non-**NULL**, the function will be treated as a *non-blocking* call and return one of the
16 following:

- 17 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
18 provided `cbfunc`. Note that the library must not invoke the callback function prior to returning
19 from the API.
- 20 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
21 returned *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

24 The returned status code will be one of the following:

- 25 **PMIX_SUCCESS** The event handler was successfully deregistered.
- 26 **PMIX_ERR_BAD_PARAM** The provided `evhdlr_ref` was unrecognized.
- 27 **PMIX_ERR_NOT_SUPPORTED** The PMIx implementation does not support event notification.

Description

29 Deregister an event handler. Note that no events corresponding to the referenced registration may
30 be delivered following completion of the deregistration operation (either return from the API with
31 **PMIX_OPERATION_SUCCEEDED** or execution of the `cbfunc`).

32 8.1.3 PMIx_Notify_event

33 Summary

34 Report an event for notification via any registered event handler.

1
PMIx v2.0

Format

C

```
2 pmix_status_t
3 PMIx_Notify_event (pmix_status_t status,
4                   const pmix_proc_t *source,
5                   pmix_data_range_t range,
6                   pmix_info_t info[], size_t ninfo,
7                   pmix_op_cbfunc_t cbfunc, void *cbdata);
```

C

8 **IN status**
9 Status code of the event ([pmix_status_t](#))

10 **IN source**
11 Pointer to a [pmix_proc_t](#) identifying the original reporter of the event (handle)

12 **IN range**
13 Range across which this notification shall be delivered ([pmix_data_range_t](#))

14 **IN info**
15 Array of [pmix_info_t](#) structures containing any further info provided by the originator of
16 the event (array of handles)

17 **IN ninfo**
18 Number of elements in the *info* array ([size_t](#))

19 **IN cbfunc**
20 Callback function to be executed upon completion of operation [pmix_op_cbfunc_t](#)
21 (function reference)

22 **IN cbdata**
23 Data to be passed to the cbfunc callback function (memory reference)

24 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.evnondef" (bool)

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

8 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrage" (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.evaffected" (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.
24

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 server's own internal processing.

27 Client application processes can call this function to notify the SMS and/or other application
28 processes of an event it encountered. Note that processes are not constrained to report status values
29 defined in the official PMIx standard — any integer value can be used. Thus, applications are free
30 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

```
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;
```

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**

PMIx v2.0 ▼ C _____ ▼

2 **PMIX_DATA_BUFFER_CONSTRUCT(buffer);**

▲ C _____ ▲

3 **IN buffer**

4 Pointer to the allocated [pmix_data_buffer_t](#) that is to be initialized (handle)

5 **Description**

6 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**

PMIx v2.0 ▼ C _____ ▼

11 **PMIX_DATA_BUFFER_DESTRUCT(buffer);**

▲ C _____ ▲

12 **IN buffer**

13 Pointer to the [pmix_data_buffer_t](#) whose data is to be released (handle)

14 **Description**

15 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**

PMIx v2.0 ▼ C _____ ▼

20 **PMIX_DATA_BUFFER_LOAD(buffer, data, size);**

▲ C _____ ▲

21 **IN buffer**

22 Pointer to a pre-allocated [pmix_data_buffer_t](#) (handle)

23 **IN data**

24 Pointer to a blob (**char***)

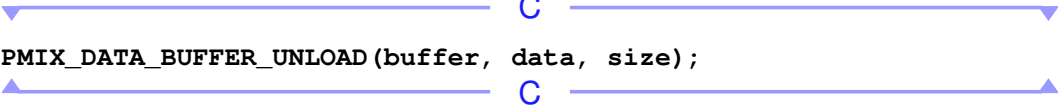
25 **IN size**

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**
9 
10 *PMIx v2.0*

11 **IN** **buffer**
12 Pointer to the `pmix_data_buffer_t` whose data is to be extracted (handle)
13 **OUT** **data**
14 Variable to be assigned the pointer to the extracted blob (**void***)
15 **OUT** **size**
16 Variable to be assigned the number of bytes in the blob **size_t**

17 **Description**
18 Extract the data in a buffer, assigning the pointer to the data (and the number of bytes in the blob) to
19 the provided variables, usually done to transmit the blob to a remote process for unpacking. The
20 buffer's internal pointer will be set to NULL to protect the data upon buffer destruct or release -
21 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
PMIx v2.0

Format

C

```

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);

```

C

- 7 **IN target**
8 Pointer to a [pmix_proc_t](#) containing the nspace/rank of the process that will be unpacking
9 the final buffer. A NULL value may be used to indicate that the target is based on the same
10 PMIx version as the caller. Note that only the target’s nspace is relevant. (handle)
- 11 **IN buffer**
12 Pointer to a [pmix_data_buffer_t](#) where the packed data is to be stored (handle)
- 13 **IN src**
14 Pointer to a location where the data resides. Strings are to be passed as (char **) — i.e., the
15 caller must pass the address of the pointer to the string as the (void*). This allows the caller to
16 pass multiple strings in a single call. (memory reference)
- 17 **IN num_vals**
18 Number of elements pointed to by the *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

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

PMIx v2.0

C

```
14 pmix_status_t  
15 PMIx_Data_unpack(const pmix_proc_t *source,  
16                 pmix_data_buffer_t *buffer, void *dest,  
17                 int32_t *max_num_values,  
18                 pmix_data_type_t type);  
19
```

C

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`)

IN type

The type of the data to be unpacked — must be one of the PMI_x defined data types (`pmix_data_type_t`)

Returns one of the following:

`PMIX_SUCCESS` The data has been unpacked as requested

`PMIX_ERR_NOT_SUPPORTED` The PMI_x implementation does not support this function.

`PMIX_ERR_BAD_PARAM` The provided buffer or dest is **NULL**

`PMIX_ERR_UNKNOWN_DATA_TYPE` The specified data type is not known to this implementation

`PMIX_ERR_OUT_OF_RESOURCE` Not enough memory to support the operation

`PMIX_ERROR` General error

Description

The unpack function unpacks the next value (or values) of a specified type from the given buffer.

The buffer must have already been initialized via an `PMIX_DATA_BUFFER_CREATE` or `PMIX_DATA_BUFFER_CONSTRUCT` call (and assumedly filled with some data) — otherwise, the `unpack_value` function will return an error. Providing an unsupported type flag will likewise be reported as an error, as will specifying a data type that *does not* match the type of the next item in the buffer. An attempt to read beyond the end of the stored data held in the buffer will also return an error.

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

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

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

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

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

C

```
7 pmix_status_t  
8 PMIx_Data_copy(void **dest, void *src,  
9 pmix_data_type_t type);
```

C

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
PMIx v2.0

Format

C

```
2 pmix_status_t  
3 PMIx_Data_print(char **output, char *prefix,  
4                 void *src, pmix_data_type_t type);
```

C

5 **IN** **output**

The address of a pointer into which the address of the resulting output is to be stored.

6
7 (**char****)

8 **IN** **prefix**

String to be prepended to the resulting output (**char***)

9
10 **IN** **src**

A pointer to the memory location of the data value to be printed (handle)

11
12 **IN** **type**

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

[pmix_data_type_t](#))

13
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.

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

C

```
1 pmix_status_t
2 PMIx_Data_copy_payload(pmix_data_buffer_t *dest,
3                       pmix_data_buffer_t *src);
```

C

```
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 C
```

17 **IN info**

18 Array of **pmix_info_t** structures (array of handles)

19 **IN ninfo**

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

21 **IN 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

PMIx libraries that choose not to support this operation *must* return **PMIX_ERR_NOT_SUPPORTED** when the function is called.

There are no required attributes for this API. Note that implementations may choose to internally execute integration for some security environments (e.g., directly contacting a *munge* server).

Implementations that support the operation but cannot directly process the client's request must pass any attributes that are provided by the client to the host environment for processing. In addition, the following attributes are required to be included in the *info* array passed from the PMIx library to the host environment:

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

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

Optional Attributes

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

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

Advice to PMIx library implementers

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

Description

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

10.1.2 PMIx_Get_credential_nb

Summary

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

1
PMIx v3.0

Format

C

```

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)

```

C

- 5 **IN info**
Array of [pmix_info_t](#) structures (array of handles)
- 6 **IN ninfo**
Number of elements in the *info* array ([size_t](#))
- 7 **IN cbfunc**
Callback function to return credential ([pmix_credential_cbfunc_t](#) function
- 8 reference)
- 9 **IN cbdata**
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

Required Attributes

19 PMIx libraries that choose not to support this operation *must* return
20 [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_GRPID** "pmix.egid" ([uint32_t](#))
30 Effective group id.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

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

10.2 Validating Credentials

The API for validating a credential is a non-blocking operation since the host environment may have to contact a remote credential service. Provision is made for the system to return additional information regarding possible authorization limitations beyond simple authentication.

10.2.1 PMIx_Validate_credential

Summary

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

1
PMIx v3.0

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**
Pointer to `pmix_byte_object_t` containing the credential (handle)
- 7
- 8 **IN info**
Array of `pmix_info_t` structures (array of handles)
- 9
- 10 **IN ninfo**
Number of elements in the *info* array (`size_t`)
- 11
- 12 **INOUT results**
Address where a pointer to an array of `pmix_info_t` containing the results of the request
can be returned (memory reference)
- 13
- 14
- 15 **INOUT nresults**
Address where the number of elements in *results* can be returned (handle)
- 16

17 Returns one of the following:

- 18 • **PMIX_SUCCESS**, indicating that the request was processed and returned *success*. Details of the
19 result will be returned in the *results* array
- 20 • a PMIx error constant indicating either an error in the input or that the request was refused

Required Attributes

21 PMIx libraries that choose not to support this operation *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`)
Effective user id.
- 30
- 31 **PMIX_GRPID** "pmix.egid" (`uint32_t`)
Effective group id.
- 32

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

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

10.2.2 PMIx_Validate_credential_nb

Summary

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

1
PMIx v3.0

Format

C

```

2 pmix_status_t
3 PMIx_Validate_credential_nb(const pmix_byte_object_t *cred,
4                             const pmix_info_t info[], size_t ninfo,
5                             pmix_validation_cbfunc_t cbfunc,
6                             void *cbdata)

```

C

- 7 **IN cred**
Pointer to `pmix_byte_object_t` containing the credential (handle)
- 8 **IN info**
Array of `pmix_info_t` structures (array of handles)
- 9 **IN ninfo**
Number of elements in the *info* array (`size_t`)
- 10 **IN cbfunc**
Callback function to return result (`pmix_validation_cbfunc_t` function reference)
- 11 **IN cbdata**
Data to be passed to the callback function (memory reference)

17 Returns one of the following:

- 18 • **PMIX_SUCCESS**, indicating that the request has been communicated to the local PMIx server -
- 19 result will be returned in the provided *cbfunc*
- 20 • a PMIx error constant indicating either an error in the input or that the request is unsupported -
- 21 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_GRPID** "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

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

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

11.1 Server Initialization and Finalization

12 The PMIx APIs may only be used between the completion of the initialization function and the start
13 of the finalization function, unless otherwise noted. The initialization and finalization functions are
14 paired, and the initialized regions defined by them must not overlap.

▼ **Advice to users** ▼

15 Server initialization includes setting up the infrastructure to support local clients, Therefore, server
16 initialization will likely result in additional overhead and an increased memory footprint than client
17 initialization alone.

11.1.1 PMIx_server_init

Summary

Initialize the PMIx server.

1
PMIx v1.0

Format

```

2  pmix_status_t
3  PMIx_server_init(pmix_server_module_t *module,
4                  pmix_info_t info[], size_t ninfo)

```

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_NAMESPACE "pmix.srv.namespace" (`char*`)

Name of the namespace to use for this PMIx server.

PMIX_SERVER_RANK "pmix.srv.rank" (`pmix_rank_t`)

Rank of this PMIx server

PMIX_SERVER_TMPDIR "pmix.srvr.tmpdir" (`char*`)

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

PMIX_SYSTEM_TMPDIR "pmix.sys.tmpdir" (`char*`)

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

PMIX_SERVER_TOOL_SUPPORT "pmix.srvr.tool" (`bool`)

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

PMIX_SERVER_SYSTEM_SUPPORT "pmix.srvr.sys" (`bool`)

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

Optional Attributes

The following attributes are optional for implementers of PMIx libraries:

PMIX_USOCK_DISABLE "pmix.usock.disable" (bool)

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

PMIX_SOCKET_MODE "pmix.sockmode" (uint32_t)

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

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

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

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

Comma-delimited list of devices and/or CIDR notation to include when establishing the TCP connection. If the library supports TCP socket connections, this attribute may be supported for specifying the interfaces to be used.

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

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

PMIX_TCP_IPV4_PORT "pmix.tcp.ipv4" (int)

The IPv4 port to be used. If the library supports IPV4 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_IPV6_PORT "pmix.tcp.ipv6" (int)

The IPv6 port to be used. If the library supports IPV6 connections, this attribute may be supported for specifying the port to be used.

PMIX_TCP_DISABLE_IPV4 "pmix.tcp.disipv4" (bool)

Set to **true** to disable IPv4 family of addresses. If the library supports IPV4 connections, this attribute may be supported for disabling it.

PMIX_TCP_DISABLE_IPV6 "pmix.tcp.disipv6" (bool)

Set to **true** to disable IPv6 family of addresses. If the library supports IPV6 connections, this attribute may be supported for disabling it.

PMIX_SERVER_REMOTE_CONNECTIONS "pmix.srvr.remote" (bool)

Allow connections from remote tools. Forces the PMIx server to not exclusively use loopback device. If the library supports connections from remote tools, this attribute may be supported for enabling or disabling it.

PMIX_EVENT_BASE "pmix.evbase" (struct event_base *)

1 Pointer to libevent¹ `event_base` to use in place of the internal progress thread.
2 **PMIX_GDS_MODULE** "`pmix.gds.mod`" (`char*`)
3 Comma-delimited string of desired modules. This attribute is specific to the PRI and
4 controls only the selection of GDS module for internal use by the process. Module selection
5 for interacting with the server is performed dynamically during the connection process.



6 **Description**
7 Initialize the PMIx server support library, and provide a pointer to a `pmix_server_module_t`
8 structure containing the caller's callback functions. The array of `pmix_info_t` structs is used to
9 pass additional info that may be required by the server when initializing. For example, it may
10 include the **PMIX_SERVER_TOOL_SUPPORT** attribute, thereby indicating that the daemon is
11 willing to accept connection requests from PMIx tools.

▼ **Advice to PMIx server hosts** ▼

12 Providing a value of **NULL** for the *module* argument is permitted, as is passing an empty *module*
13 structure. Doing so indicates that the host environment will not provide support for multi-node
14 operations such as **PMIx_Fence**, but does intend to support local clients access to information.



15 **11.1.2 PMIx_server_finalize**

16 **Summary**
17 Finalize the PMIx server library.

18 **Format**
19 `pmix_status_t`
20 `PMIx_server_finalize(void)`

PMIx v1.0 ▼ C ▼

▲ C ▲

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

22 **Description**
23 Finalize the PMIx server support library, terminating all connections to the attached tools and any
24 local clients. All allocated resources are released.

25 **11.2 Server Support Functions**

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

¹<http://libevent.org/>

1 11.2.1 PMIx_generate_regex

2 Summary

3 Generate a compressed representation of the input string.

4 Format

PMIx v1.0

C

5 `pmix_status_t`

6 `PMIx_generate_regex(const char *input, char **output)`

C

7 **IN** `input`

8 String to process (string)

9 **OUT** `output`

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

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

Advice to PMIx server hosts

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

26 Communicating the resulting output should be done by first packing the returned expression using
27 the `PMIx_Data_pack`, declaring the input to be of type `PMIX_REGEX`, and then obtaining the
28 resulting blob to be communicated using the `PMIX_DATA_BUFFER_UNLOAD` macro. The
29 reciprocal method can be used on the remote end prior to passing the regex into
30 `PMIx_server_register_namespace`. The pack/unpack routines will ensure proper handling of
31 the data based on the regex prefix.

1 11.2.2 `PMIx_generate_ppn`

2 Summary

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

4 Format

PMIx v1.0

C

5 `pmix_status_t PMIx_generate_ppn(const char *input, char **ppn)`

C

6 **IN** `input`

7 String to process (string)

8 **OUT** `ppn`

9 Compressed representation of `input` (array of bytes)

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

11 Description

12 The input shall consist of a semicolon-separated list of ranges representing the ranks of processes
13 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
14 to the node name provided at that position in the input to `PMIx_generate_regex`. Thus, in the
15 example, ranks 1-4 would be located on the first node of the comma-separated list of names
16 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 11.2.3 `PMIx_server_register_namespace`

28 Summary

29 Setup the data about a particular namespace.

1
PMIx v1.0

Format

C

```

2 pmix_status_t
3 PMIx_server_register_namespace(const pmix_namespace_t nspace,
4                               int nlocalprocs,
5                               pmix_info_t info[], size_t ninfo,
6                               pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 7 **IN nspace**
Character array of maximum size **PMIX_MAX_NSLEN** containing the namespace identifier (string)
- 8
- 9
- 10 **IN nlocalprocs**
number of local processes (integer)
- 11
- 12 **IN info**
Array of info structures (array of handles)
- 13
- 14 **IN ninfo**
Number of elements in the *info* array (integer)
- 15
- 16 **IN cbfunc**
Callback function **pmix_op_cbfunc_t** (function reference)
- 17
- 18 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 19

20 Returns one of the following:

- 21 • **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.
- 22
- 23
- 24 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- 25
- 26 • 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

Required Attributes

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

- 29 **PMIX_REGISTER_NODATA** "pmix.reg.nodata" (bool)
- 30 Registration is for this namespace only, do not copy job data - this attribute is not accessed
- 31 using the **PMIx_Get**

32
33 Host environments are required to provide the following attributes:

- 34 • for the session containing the given namespace:

- 1 – **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)
2 Number of allocated slots in a session - each slot may or may not be occupied by an
3 executing process. Note that this attribute is the equivalent to the combination of
4 **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it
5 is included in the Standard for historical reasons.
- 6 • for the given namespace:
- 7 – **PMIX_JOBID** "pmix.jobid" (char*)
8 Job identifier assigned by the scheduler.
- 9 – **PMIX_JOB_SIZE** "pmix.job.size" (uint32_t)
10 Total number of processes in this job across all contained applications. Note that this
11 value can be different from **PMIX_MAX_PROCS** . For example, users may choose to
12 subdivide an allocation (running several jobs in parallel within it), and dynamic
13 programming models may support adding and removing processes from a running **job**
14 on-the-fly. In the latter case, PMIx events must be used to notify processes within the
15 job that the job size has changed.
- 16 – **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
17 Maximum number of processes that can be executed in this context (session,
18 namespace, application, or node). Typically, this is a constraint imposed by a scheduler
19 or by user settings in a hostfile or other resource description.
- 20 – **PMIX_NODE_MAP** "pmix.nmap" (char*)
21 Regular expression of nodes - see 11.2.3.1 for an explanation of its generation.
- 22 – **PMIX_PROC_MAP** "pmix.pmap" (char*)
23 Regular expression describing processes on each node - see 11.2.3.1 for an explanation
24 of its generation.
- 25 • for its own node:
- 26 – **PMIX_LOCAL_SIZE** "pmix.local.size" (uint32_t)
27 Number of processes in this job or application on this node.
- 28 – **PMIX_LOCAL_PEERS** "pmix.lpeers" (char*)
29 Comma-delimited list of ranks on this node within the specified namespace - referenced
30 using **PMIX_RANK_WILDCARD** .
- 31 – **PMIX_LOCAL_CPUSSETS** "pmix.lcpus" (char*)
32 Colon-delimited cpusets of local peers within the specified namespace - referenced
33 using **PMIX_RANK_WILDCARD** .
- 34 • for each process in the given namespace:
- 35 – **PMIX_RANK** "pmix.rank" (pmix_rank_t)
36 Process rank within the job.
- 37 – **PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)

- 1 Local rank on this node within this job.
- 2 - **PMIX_NODE_RANK** "pmix.nrank" (uint16_t)
- 3 Process rank on this node spanning all jobs.
- 4 - **PMIX_NODEID** "pmix.nodeid" (uint32_t)
- 5 Node identifier expressed as the node's index (beginning at zero) in an array of nodes
- 6 within the active session. The value must be unique and directly correlate to the
- 7 **PMIX_HOSTNAME** of the node - i.e., users can interchangeably reference the same
- 8 location using either the **PMIX_HOSTNAME** or corresponding **PMIX_NODEID** .

9 If more than one application is included in the namespace, then the host environment is also

10 required to provide the following attributes:

- 11 • for each application:
- 12 - **PMIX_APPNUM** "pmix.appnum" (uint32_t)
- 13 Application number within the job.
- 14 - **PMIX_APPLDR** "pmix.aldr" (pmix_rank_t)
- 15 Lowest rank in this application within this job - referenced using
- 16 **PMIX_RANK_WILDCARD** .
- 17 - **PMIX_APP_SIZE** "pmix.app.size" (uint32_t)
- 18 Number of processes in this application.
- 19 • for each process:
- 20 - **PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)
- 21 Process rank within this application.
- 22 - **PMIX_APPNUM** "pmix.appnum" (uint32_t)
- 23 Application number within the job.



24 The following attributes may be provided by host environments:

- 25 • for the session containing the given namespace:
- 26 - **PMIX_SESSION_ID** "pmix.session.id" (uint32_t)
- 27 Session identifier - referenced using **PMIX_RANK_WILDCARD** .
- 28 • for the given namespace:
- 29 - **PMIX_SERVER_NAMESPACE** "pmix.srv.namespace" (char*)
- 30 Name of the namespace to use for this PMIx server.
- 31 - **PMIX_SERVER_RANK** "pmix.srv.rank" (pmix_rank_t)
- 32 Rank of this PMIx server

- 1 - **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
- 2 Starting global rank of this job - referenced using [PMIX_RANK_WILDCARD](#) .
- 3 - **PMIX_ALLOCATED_NODELIST** "pmix.alist" (char*)
- 4 Comma-delimited list of all nodes in this allocation regardless of whether or not they
- 5 currently host processes - referenced using [PMIX_RANK_WILDCARD](#) .
- 6 - **PMIX_JOB_NUM_APPS** "pmix.job.napps" (uint32_t)
- 7 Number of applications in this job.
- 8 - **PMIX_MAPBY** "pmix.mapby" (char*)
- 9 Process mapping policy - when accessed using [PMIx_Get](#) , use the
- 10 [PMIX_RANK_WILDCARD](#) value for the rank to discover the mapping policy used for
- 11 the provided namespace
- 12 - **PMIX_RANKBY** "pmix.rankby" (char*)
- 13 Process ranking policy - when accessed using [PMIx_Get](#) , use the
- 14 [PMIX_RANK_WILDCARD](#) value for the rank to discover the ranking algorithm used
- 15 for the provided namespace
- 16 - **PMIX_BINDTO** "pmix.bindto" (char*)
- 17 Process binding policy - when accessed using [PMIx_Get](#) , use the
- 18 [PMIX_RANK_WILDCARD](#) value for the rank to discover the binding policy used for
- 19 the provided namespace
- 20 - **PMIX_ANL_MAP** "pmix.anlmap" (char*)
- 21 Process mapping in Argonne National Laboratory's PMI-1/PMI-2 notation.
- 22 • for its own node:
- 23 - **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)
- 24 Total available physical memory on this node.
- 25 - **PMIX_HWLOC_XML_V1** "pmix.hwlocxml1" (char*)
- 26 XML representation of local topology using HWLOC's v1.x format.
- 27 - **PMIX_HWLOC_XML_V2** "pmix.hwlocxml2" (char*)
- 28 XML representation of local topology using HWLOC's v2.x format.
- 29 - **PMIX_LOCALLDR** "pmix.lldr" (pmix_rank_t)
- 30 Lowest rank on this node within this job - referenced using [PMIX_RANK_WILDCARD](#) .
- 31
- 32 - **PMIX_NODE_SIZE** "pmix.node.size" (uint32_t)
- 33 Number of processes across all jobs on this node.
- 34 - **PMIX_LOCAL_PROCS** "pmix.lprocs" (pmix_proc_t array)
- 35 Array of [pmix_proc_t](#) of all processes on the specified node - referenced using
- 36 [PMIX_RANK_WILDCARD](#) .

- 1 • for each process in the given namespace:
- 2 – **PMIX_PROCID** "pmix.procid" (pmix_proc_t)
- 3 Process identifier
- 4 – **PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)
- 5 Process rank spanning across all jobs in this session.
- 6 – **PMIX_HOSTNAME** "pmix.hname" (char*)
- 7 Name of the host (e.g., where a specified process is running, or a given device is
- 8 located).

9 Attributes not directly provided by the host environment may be derived by the PMIx server library
10 from other required information and included in the data made available to the server library's
11 clients.

12 The following optional attributes may be provided by the host environment to identify the
13 programming model (as specified by the user) being executed within the namespace. The PMIx
14 server library may utilize this information to customize the environment to fit that model (e.g.,
15 adding environmental variables specified by the corresponding standard for that model):

- 16 • **PMIX_PROGRAMMING_MODEL** "pmix.pgm.model" (char*)
- 17 Programming model being initialized (e.g., "MPI" or "OpenMP")
- 18 • **PMIX_MODEL_LIBRARY_NAME** "pmix.mdl.name" (char*)
- 19 Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
- 20 • **PMIX_MODEL_LIBRARY_VERSION** "pmix.mld.vrs" (char*)
- 21 Programming model version string (e.g., "2.1.1")



22 Description

23 Pass job-related information to the PMIx server library for distribution to local client processes.

▼ Advice to PMIx server hosts ▼

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

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

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

Advice to users

1 The number of local processes for any given namespace is generally fixed at the time of application
2 launch. Calls to `PMIx_Spawn` result in processes launched in their own namespace, not that of
3 their parent. However, it is possible for processes to *migrate* to another node via a call to
4 `PMIx_Job_control_nb`, thus resulting in a change to the number of local processes on both
5 the initial node and the node to which the process moved. It is therefore critical that applications
6 not migrate processes without first ensuring that PMIx-based collective operations are not in
7 progress, and that no such operations be initiated until process migration has completed.

8 11.2.3.1 Assembling the registration information

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

Advice to PMIx server hosts

14 Creating the *info* array of data requires knowing in advance the number of elements required for the
15 array. This can be difficult to compute and somewhat fragile in practice. One method for resolving
16 the problem is to create a linked list of objects, each containing a single `pmix_info_t` structure.
17 Allocation and manipulation of the list can then be accomplished using existing standard methods.
18 Upon completion, the final *info* array can be allocated based on the number of elements on the list,
19 and then the values in the list object `pmix_info_t` structures transferred to the corresponding
20 array element utilizing the `PMIX_INFO_XFER` macro.

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

```

1  char **nodes = NULL;
2  char *nodelist;
3  char *regex;
4  size_t n;
5  pmix_status_t rc;
6  pmix_info_t info;
7
8  /* loop over an array of nodes, adding each
9   * name to the array */
10 for (n=0; n < num_nodes; n++)
11     /* filter the nodes to ignore those not included
12      * in the target range (session, job, etc.). In
13      * this example, all nodes are accepted */
14     PMIX_ARGV_APPEND(&nodes, node[n]->name);
15
16
17 /* join into a comma-delimited string */
18 nodelist = PMIX_ARGV_JOIN(nodes, ',');
19
20 /* release the array */
21 PMIX_ARGV_FREE(nodes);
22
23 /* generate regex */
24 rc = PMIX_generate_regex(nodelist, &regex);
25
26 /* release list */
27 free(nodelist);
28
29 /* pass the regex as the value to the PMIX_NODE_MAP key */
30 PMIX_INFO_LOAD(&info, PMIX_NODE_MAP, regex, PMIX_STRING);
31 /* release the regex */
32 free(regex);
33

```

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

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

```

1  char **ndppn;
2  char rank[30];
3  char **ppnarray = NULL;
4  char *ppn;
5  char *localranks;
6  char *regex;
7  size_t n, m;
8  pmix_status_t rc;
9  pmix_info_t info;
10
11 /* loop over an array of nodes */
12 for (n=0; n < num_nodes; n++)
13     /* for each node, construct an array of ranks on that node */
14     ndppn = NULL;
15     for (m=0; m < node[n]->num_procs; m++)
16         /* ignore processes that are not part of the target job */
17         if (!PMIX_CHECK_NAMESPACE(targetjob,node[n]->proc[m].nspace))
18             continue;
19
20         snprintf(rank, 30, "%d", node[n]->proc[m].rank);
21         PMIX_ARGV_APPEND(&ndppn, rank);
22
23     /* convert the array into a comma-delimited string of ranks */
24     localranks = PMIX_ARGV_JOIN(ndppn, ',');
25     /* release the local array */
26     PMIX_ARGV_FREE(ndppn);
27     /* add this node's contribution to the overall array */
28     PMIX_ARGV_APPEND(&ppnarray, localranks);
29     /* release the local list */
30     free(localranks);
31
32
33 /* join into a semicolon-delimited string */
34 ppn = PMIX_ARGV_JOIN(ppnarray, ';');
35
36 /* release the array */
37 PMIX_ARGV_FREE(ppnarray);
38
39 /* generate ppn regex */
40 rc = PMIx_generate_ppn(ppn, &regex);
41
42 /* release list */

```

```

1  free(ppn);
2
3  /* pass the regex as the value to the PMIX_PROC_MAP key */
4  PMIX_INFO_LOAD(&info, PMIX_PROC_MAP, regex, PMIX_STRING);
5  /* release the regex */
6  free(regex);
7

```



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

- 13 • Session-level information includes all session-specific values. In many cases, only two values (
14 **PMIX_SESSION_ID** and **PMIX_UNIV_SIZE**) are included in the registration array. Since
15 both of these values are session-specific, they can be specified independently - i.e., in their own
16 **pmix_info_t** elements of the *info* array. Alternatively, they can be provided as a
17 **pmix_data_array_t** array of **pmix_info_t** using the **PMIX_SESSION_INFO_ARRAY**
18 attribute and identified by including the **PMIX_SESSION_ID** attribute in the array - this is
19 required in cases where non-specific attributes (e.g., **PMIX_NUM_NODES** or **PMIX_NODE_MAP**
20) are passed to describe aspects of the session. Note that the node map can include nodes not
21 used by the job being registered as no corresponding process map is specified.

22 The *info* array at this point might look like (where the labels identify the corresponding attribute
23 - e.g., “Session ID” corresponds to the **PMIX_SESSION_ID** attribute):

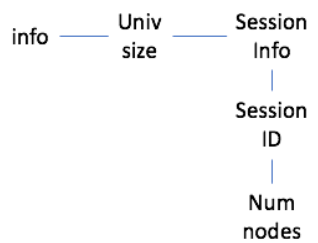


Figure 11.1.: Session-level information elements

- 24 • Job-level information includes all job-specific values such as **PMIX_JOB_SIZE** ,
25 **PMIX_JOB_NUM_APPS** , and **PMIX_JOBID** . Since each invocation of
26 **PMIx_server_register_nspace** describes a single *job* , job-specific values can be
27 specified independently - i.e., in their own **pmix_info_t** elements of the *info* array.
28 Alternatively, they can be provided as a **pmix_data_array_t** array of **pmix_info_t**
29 identified by the **PMIX_JOB_INFO_ARRAY** attribute - this is required in cases where

1 non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the job. Note
 2 that since the invocation only involves a single namespace, there is no need to include the
 3 `PMIX_NAMESPACE` attribute in the array.

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

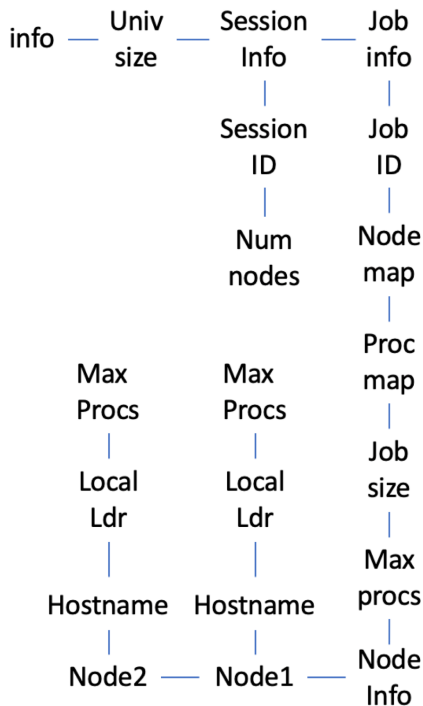


Figure 11.2.: Job-level information elements

5 Note that in this example, `PMIX_NUM_NODES` is not required as that information is contained
 6 in the `PMIX_NODE_MAP` attribute. Similarly, `PMIX_JOB_SIZE` is not technically required as
 7 that information is contained in the `PMIX_PROC_MAP` when combined with the corresponding
 8 node map - however, there is no issue with including the job size as a separate entry.

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

- Application-level information includes all application-specific values such as `PMIX_APP_SIZE` and `PMIX_APPLDR`. If the `job` contains only a single `application`, then the application-specific values can be specified independently - i.e., in their own `pmix_info_t` elements of the `info` array - or as a `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_APP_INFO_ARRAY` attribute and identified by including the `PMIX_APPNUM` attribute in the array. Use of the array format is must in cases where non-specific attributes (e.g., `PMIX_NODE_MAP`) are passed to describe aspects of the application.

However, in the case of a job consisting of multiple applications, all application-specific values for each application must be provided using the `PMIX_APP_INFO_ARRAY` format, each identified by its `PMIX_APPNUM` value.

Upon conclusion of this step, the `info` array might look like that shown in 11.3, assuming there are two applications in the job being registered:

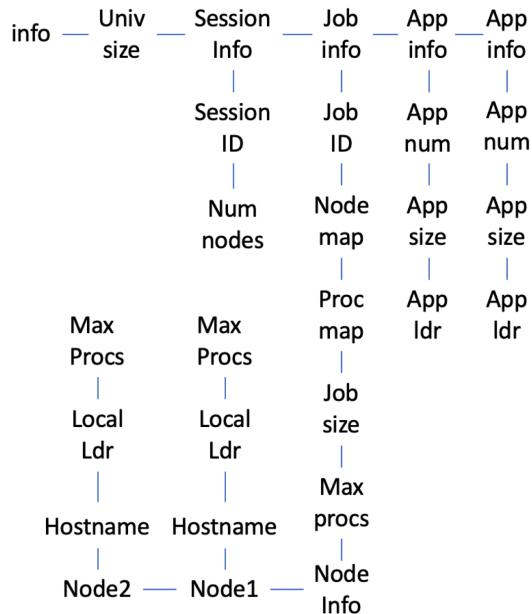


Figure 11.3.: Application-level information elements

- Process-level information includes an entry for each process in the job being registered, each entry marked with the `PMIX_PROC_DATA` attribute. The `rank` of the process must be the first entry in the array - this provides efficiency when storing the data. Upon conclusion of this step, the `info` array might look like the diagram in 11.4:
- For purposes of this example, node-level information only includes values describing the local node - i.e., it does not include information about other nodes in the job or session. In many cases, the values included in this level are unique to it and can be specified independently - i.e., in their

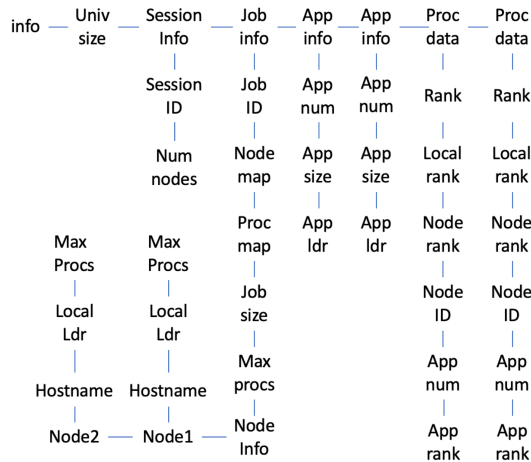


Figure 11.4.: Process-level information elements

own `pmix_info_t` elements of the `info` array. Alternatively, they can be provided as a `pmix_data_array_t` array of `pmix_info_t` using the `PMIX_NODE_INFO_ARRAY` attribute - this is required in cases where non-specific attributes are passed to describe aspects of the node, or where values for multiple nodes are being provided.

The node-level information requires two elements that must be constructed in a manner similar to that used for the node map. The `PMIX_LOCAL_PEERS` value is computed based on the processes on the local node, filtered to select those from the job being registered, as shown below using the tools provided by PMIx:

```

9      char **ndppn = NULL;
10     char rank[30];
11     char *localranks;
12     size_t m;
13     pmix_info_t info;
14
15     for (m=0; m < mynode->num_procs; m++)
16         /* ignore processes that are not part of the target job */
17         if (!PMIX_CHECK_NAMESPACE(targetjob, mynode->proc[m].nspace))
18             continue;
19
20         snprintf(rank, 30, "%d", mynode->proc[m].rank);
21         PMIX_ARGV_APPEND(&ndppn, rank);
22
23     /* convert the array into a comma-delimited string of ranks */
  
```

```

1  localranks = PMIX_ARGV_JOIN(ndppn, ',');
2  /* release the local array */
3  PMIX_ARGV_FREE(ndppn);
4
5  /* pass the string as the value to the PMIX_LOCAL_PEERS key */
6  PMIX_INFO_LOAD(&info, PMIX_LOCAL_PEERS, localranks, PMIX_STRING);
7  /* release the list */
8  free(localranks);
9

```

C

10 The **PMIX_LOCAL_CPUSSETS** value is constructed in a similar manner. In the provided
11 example, it is assumed that the Hardware Locality (HWLOC) cpuset representation (a
12 comma-delimited string of processor IDs) of the processors assigned to each process has
13 previously been generated and stored on the process description. Thus, the value can be
14 constructed as shown below:

C

```

15 char **ndcpus = NULL;
16 char *localcpus;
17 size_t m;
18 pmix_info_t info;
19
20 for (m=0; m < mynode->num_procs; m++)
21     /* ignore processes that are not part of the target job */
22     if (!PMIX_CHECK_NAMESPACE(targetjob, mynode->proc[m].namespace))
23         continue;
24
25     PMIX_ARGV_APPEND(&ndcpus, mynode->proc[m].cpuset);
26
27     /* convert the array into a colon-delimited string */
28     localcpus = PMIX_ARGV_JOIN(ndcpus, ':');
29     /* release the local array */
30     PMIX_ARGV_FREE(ndcpus);
31
32     /* pass the string as the value to the PMIX_LOCAL_CPUSSETS key */
33     PMIX_INFO_LOAD(&info, PMIX_LOCAL_CPUSSETS, localcpus, PMIX_STRING);
34     /* release the list */
35     free(localcpus);
36

```

C

37 Note that for efficiency, these two values can be computed at the same time.

38 The final *info* array might therefore look like the diagram in [11.5](#):

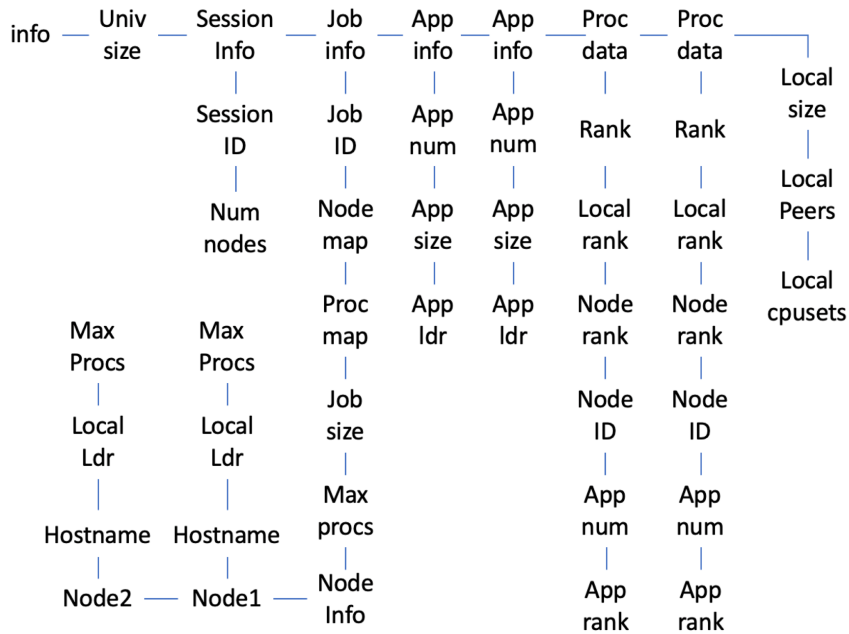


Figure 11.5.: Final information array

1 11.2.4 PMIx_server_deregister_namespace

2 Summary

3 Deregister a namespace.

4 Format

PMIx v1.0

```

5 void PMIx_server_deregister_namespace(const pmix_namespace_t nspace,
6                                     pmix_op_cbfunc_t cbfunc, void *cbdata)

```

7 **IN nspace**

8 Namespace (string)

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)

Description

Deregister the specified *nospace* and purge all objects relating to it, including any client information from that namespace. This is intended to support persistent PMIx servers by providing an opportunity for the host RM to tell the PMIx server library to release all memory for a completed job. Note that the library must not invoke the callback function prior to returning from the API.

11.2.5 PMIx_server_register_client

Summary

Register a client process with the PMIx server library.

Format

PMIx v1.0

C

```
pmix_status_t
PMIx_server_register_client(const pmix_proc_t *proc,
                           uid_t uid, gid_t gid,
                           void *server_object,
                           pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN **proc**
[pmix_proc_t](#) structure (handle)

IN **uid**
user id (integer)

IN **gid**
group id (integer)

IN **server_object**
(memory reference)

IN **cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)

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

Returns one of the following:

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

Description

Register a client process with the PMIx server library.

The host server can also, if it desires, provide an object it wishes to be returned when a server function is called that relates to a specific process. For example, the host server may have an object that tracks the specific client. Passing the object to the library allows the library to provide that object to the host server during subsequent calls related to that client, such as a [pmix_server_client_connected_fn_t](#) function. This allows the host server to access the object without performing a lookup based on the client's namespace and rank.

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting the client process. The expected user ID and group ID of the child process allows the server library to properly authenticate clients as they connect by requiring the two values to match. Accordingly, the detected user and group ID's of the connecting process are not included in the [pmix_server_client_connected_fn_t](#) server module function.

Advice to PMIx library implementers

For security purposes, the PMIx server library should check the user and group ID's of a connecting process against those provided for the declared client process identifier via the [PMIx_server_register_client](#) prior to completing the connection.

11.2.6 PMIx_server_deregister_client

Summary

Deregister a client and purge all data relating to it.

Format

PMIx v1.0

C

```
void
PMIx_server_deregister_client(const pmix_proc_t *proc,
                             pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `proc`
[pmix_proc_t](#) structure (handle)

IN `cbfunc`
Callback function [pmix_op_cbfunc_t](#) (function reference)

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

Description

The `PMIx_server_deregister_nspace` API will delete all client information for that namespace. The PMIx server library will automatically perform that operation upon disconnect of all local clients. This API is therefore intended primarily for use in exception cases, but can be called in non-exception cases if desired. Note that the library must not invoke the callback function prior to returning from the API.

11.2.7 PMIx_server_setup_fork

Summary

Setup the environment of a child process to be forked by the host.

Format

PMIx v1.0

```
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).

11.2.8 PMIx_server_dmodex_request

Summary

Define a function by which the host server can request modex data from the local PMIx server.

1
PMIx v1.0

Format

C

```
2 pmix_status_t PMIx_server_dmodex_request(const pmix_proc_t *proc,  
3 pmix_dmodex_response_fn_t cbfunc,  
4 void *cbdata)
```

C

- 5 **IN** `proc`
- 6 `pmix_proc_t` structure (handle)
- 7 **IN** `cbfunc`
- 8 Callback function `pmix_dmodex_response_fn_t` (function reference)
- 9 **IN** `cbdata`
- 10 Data to be passed to the callback function (memory reference)

11 Returns one of the following:

- 12 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
- 13 will be returned in the provided `cbfunc`. Note that the library must not invoke the callback
- 14 function prior to returning from the API.
- 15 • a PMIx error constant indicating an error in the input - the `cbfunc` will not be called

Description

17 Define a function by which the host server can request modex data from the local PMIx server.
18 Traditional wireup procedures revolve around the per-process posting of data (e.g., location and
19 endpoint information) via the **PMIx_Put** and **PMIx_Commit** functions followed by a
20 **PMIx_Fence** barrier that globally exchanges the posted information. However, the barrier
21 operation represents a significant time impact at large scale.

22 PMIx supports an alternative wireup method known as *Direct Modex* that replaces the
23 barrier-based exchange of all process-posted information with on-demand fetch of a peer's data. In
24 place of the barrier operation, data posted by each process is cached on the local PMIx server.
25 When a process requests the information posted by a particular peer, it first checks the local cache
26 to see if the data is already available. If not, then the request is passed to the local PMIx server,
27 which subsequently requests that its RM host request the data from the RM daemon on the node
28 where the specified peer process is located. Upon receiving the request, the RM daemon passes the
29 request into its PMIx server library using the **PMIx_server_dmodex_request** function,
30 receiving the response in the provided `cbfunc` once the indicated process has posted its information.
31 The RM daemon then returns the data to the requesting daemon, who subsequently passes the data
32 to its PMIx server library for transfer to the requesting client.

Advice to users

33 While direct modex allows for faster launch times by eliminating the barrier operation, per-peer
34 retrieval of posted information is less efficient. Optimizations can be implemented - e.g., by
35 returning posted information from all processes on a node upon first request - but in general direct
36 modex remains best suited for sparsely connected applications.

1 11.2.9 PMIx_server_setup_application

2 Summary

3 Provide a function by which the resource manager can request application-specific setup data prior
4 to launch of a `job`.

5 Format

PMIx v2.0

C

```
6 pmix_status_t  
7 PMIx_server_setup_application(const pmix_namespace_t nspace,  
8                             pmix_info_t info[], size_t ninfo,  
9                             pmix_setup_application_cbfunc_t cbfunc,  
10                            void *cbdata)
```

C

11 **IN nspace**
12 namespace (string)
13 **IN info**
14 Array of info structures (array of handles)
15 **IN ninfo**
16 Number of elements in the *info* array (integer)
17 **IN cbfunc**
18 Callback function `pmix_setup_application_cbfunc_t` (function reference)
19 **IN cbdata**
20 Data to be passed to the *cbfunc* callback function (memory reference)

21 Returns one of the following:

- 22 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
23 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback
24 function prior to returning from the API.
- 25 • a PMIx error constant indicating either an error in the input - the *cbfunc* will not be called

Required Attributes

26 PMIx libraries that support this operation are required to support the following:

27 **PMIX_SETUP_APP_ENVARS** "pmix.setup.env" (bool)
28 Harvest and include relevant environmental variables
29 **PMIX_SETUP_APP_NONENVARS** "pmix.setup.nenv" (bool)
30 Include all relevant data other than environmental variables
31 **PMIX_SETUP_APP_ALL** "pmix.setup.all" (bool)

1 Include all relevant data

2 **PMIX_ALLOC_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

22 **PMIX_ALLOC_FABRIC_TYPE** "pmix.alloc.nettype" (char*)

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

24 **PMIX_ALLOC_FABRIC_PLANE** "pmix.alloc.netplane" (char*)

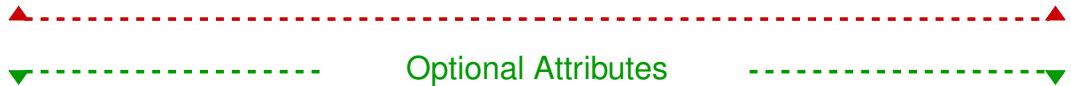
25 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

26 **PMIX_ALLOC_FABRIC_ENDPTS** "pmix.alloc.endpts" (`size_t`)

27 Number of endpoints to allocate per process

28 **PMIX_ALLOC_FABRIC_ENDPTS_NODE** "pmix.alloc.endpts.nd" (`size_t`)

29 Number of endpoints to allocate per node



Optional Attributes

30 PMIx libraries that support this operation may support the following:

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

32 Mbits/sec.

33 **PMIX_ALLOC_FABRIC_QOS** "pmix.alloc.netqos" (char*)

34 Quality of service level.

35 **PMIX_ALLOC_TIME** "pmix.alloc.time" (`uint32_t`)

1 Time in seconds.

2 The following optional attributes may be provided by the host environment to identify the
3 programming model (as specified by the user) being executed within the application. The PMIx
4 server library may utilize this information to harvest/forward model-specific environmental
5 variables, record the programming model associated with the application, etc.

- 6 • **PMIX_PROGRAMMING_MODEL** "pmix.pgm.model" (char*)
7 Programming model being initialized (e.g., "MPI" or "OpenMP")
- 8 • **PMIX_MODEL_LIBRARY_NAME** "pmix.mdl.name" (char*)
9 Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
- 10 • **PMIX_MODEL_LIBRARY_VERSION** "pmix.mld.vrs" (char*)
11 Programming model version string (e.g., "2.1.1")



12 Description

13 Provide a function by which the RM can request application-specific setup data (e.g., environmental
14 variables, fabric configuration and security credentials) from supporting PMIx server library
15 subsystems prior to initiating launch of a job.

▼ Advice to PMIx server hosts ▼

16 Host environments are required to execute this operation prior to launching a job. In addition to
17 supported directives, the *info* array must include a description of the **job** using the
18 **PMIX_NODE_MAP** and **PMIX_PROC_MAP** attributes.



19 This is defined as a non-blocking operation in case contributing subsystems need to perform some
20 potentially time consuming action (e.g., query a remote service) before responding. The returned
21 data must be distributed by the RM and subsequently delivered to the local PMIx server on each
22 node where application processes will execute, prior to initiating execution of those processes.

▼ Advice to PMIx library implementers ▼

23 Support for harvesting of environmental variables and providing of local configuration information
24 by the PMIx implementation is optional.



25 11.2.10 PMIx_Register_attributes

26 Summary

27 Register host environment attribute support for a function.

1
PMIx v4.0

Format

C

```
2 pmix_status_t
3 PMIx_Register_attributes(char *function,
4                          pmix_regattr_t attrs[],
5                          size_t nattrs)
```

C

- 6 **IN function**
String name of function (string)
- 8 **IN attrs**
Array of [pmix_regattr_t](#) describing the supported attributes (handle)
- 10 **IN nattrs**
Number of elements in *attrs* (**size_t**)

12 Returns [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

Description

13 The **PMIx_Register_attributes** function is used by the host environment to register with
 14 its PMIx server library the attributes it supports for each [pmix_server_module_t](#) function.
 15 The *function* is the string name of the server module function (e.g., "register_events",
 16 "validate_credential", or "allocate") whose attributes are being registered. See the
 17 [pmix_regattr_t](#) entry for a description of the *attrs* array elements.
 18
 19 Note that the host environment can also query the library (using the [PMIx_Query_info_nb](#)
 20 API) for its attribute support both at the server, client, and tool levels once the host has executed
 21 [PMIx_server_init](#) since the server will internally register those values.

Advice to PMIx server hosts

22 Host environments are strongly encouraged to register all supported attributes immediately after
23 initializing the library to ensure that user requests are correctly serviced.

Advice to PMIx library implementers

PMIx implementations are *required* to register all internally supported attributes for each API during initialization of the library (i.e., when the process calls their respective PMIx init function). Specifically, the implementation *must not* register supported attributes upon first call to a given API as this would prevent users from discovering supported attributes prior to first use of an API.

It is the implementation's responsibility to associate registered attributes for a given `pmix_server_module_t` function with their corresponding user-facing API. Supported attributes *must* be reported to users in terms of their support for user-facing APIs, broken down by the level (see 14.4.33) at which the attribute is supported.

Note that attributes can/will be registered on an API for each level. It is *required* that the implementation support user queries for supported attributes on a per-level basis. Duplicate registrations at the *same* level for a function *shall* return an error - however, duplicate registrations at *different* levels *shall* be independently tracked.

11.2.11 PMIx_server_setup_local_support

Summary

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application.

Format

PMIx v2.0

C

```
pmix_status_t
PMIx_server_setup_local_support(const pmix_namespace_t nspace,
                               pmix_info_t info[], size_t ninfo,
                               pmix_op_cbfunc_t cbfunc,
                               void *cbdata);
```

C

IN nspace
Namespace (string)

IN info
Array of info structures (array of handles)

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

IN cbfunc
Callback function `pmix_op_cbfunc_t` (function reference)

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

Returns one of the following:

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

Description

Provide a function by which the local PMIx server can perform any application-specific operations prior to spawning local clients of a given application. For example, a fabric library might need to setup the local driver for “instant on” addressing. The data provided in the *info* array is the data returned to the host RM by the callback function executed as a result of a call to

[PMIx_server_setup_application](#) .

Advice to PMIx server hosts

Host environments are required to execute this operation prior to starting any local application processes from the specified namespace.

11.2.12 PMIx_server_IOF_deliver

Summary

Provide a function by which the host environment can pass forwarded IO to the PMIx server library for distribution to its clients.

Format

PMIx v3.0

C

```
pmix_status_t
PMIx_server_IOF_deliver(const pmix_proc_t *source,
                        pmix_iof_channel_t channel,
                        const pmix_byte_object_t *bo,
                        const pmix_info_t info[], size_t ninfo,
                        pmix_op_cbfunc_t cbfunc, void *cbdata);
```

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_iof_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 • **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 library must not invoke the callback
 18 function prior to returning from the API.
- 19 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
 20 returned *success* - the *cbfunc* will not be called
- 21 • a PMIx error constant indicating either an error in the input or that the request was immediately
 22 processed and failed - the *cbfunc* will not be called

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 **11.2.13 PMIx_server_collect_inventory**

29 **Summary**

30 Collect inventory of resources on a node

1
PMIx v3.0

Format

C

```

2  pmix_status_t
3  PMIx_server_collect_inventory(const pmix_info_t directives[],
4                               size_t ndirs,
5                               pmix_info_cbfunc_t cbfunc,
6                               void *cbdata);

```

C

- 7 **IN directives**
Array of [pmix_info_t](#) directing the request (array of handles)
- 8
- 9 **IN ndirs**
Number of elements in the *directives* array ([size_t](#))
- 10
- 11 **IN cbfunc**
Callback function to return collected data ([pmix_info_cbfunc_t](#) function reference)
- 12
- 13 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 14

15 Returns [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant. In the event
16 the function returns an error, the *cbfunc* will not be called.

Description

17 Provide a function by which the host environment can request its PMIx server library collect an
18 inventory of local resources. Supported resources depends upon the PMIx implementation, but may
19 include the local node topology and fabric interfaces.
20

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 11.2.14 PMIx_server_deliver_inventory

28 Summary

29 Pass collected inventory to the PMIx server library for storage

Format

C

```

1  pmix_status_t
2  PMIx_server_deliver_inventory(const pmix_info_t info[],
3                                size_t ninfo,
4                                const pmix_info_t directives[],
5                                size_t ndirs,
6                                pmix_op_cbfunc_t cbfunc,
7                                void *cbdata);
8

```

C

9 **IN info**
 Array of [pmix_info_t](#) containing the inventory (array of handles)

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

11 **IN directives**
 Array of [pmix_info_t](#) directing the request (array of handles)

12 **IN ndirs**
 Number of elements in the *directives* array (**size_t**)

13 **IN cbfunc**
 Callback function [pmix_op_cbfunc_t](#) (function reference)

14 **IN cbdata**
 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 will be returned in the provided *cbfunc*. Note that the library must not invoke the callback function prior to returning from the API.
- 17 • [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- 18 • 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

19 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.

1 11.3 Server Function Pointers

2 PMIx utilizes a "function-shipping" approach to support for implementing the server-side of the
3 protocol. This method allows RMs to implement the server without being burdened with PMIx
4 internal details. When a request is received from the client, the corresponding server function will
5 be called with the information.

6 Any functions not supported by the RM can be indicated by a **NULL** for the function pointer. PMIx
7 implementations are required to return a **PMIX_ERR_NOT_SUPPORTED** status to all calls to
8 functions that require host environment support and are not backed by a corresponding server
9 module entry.

10 The host RM will provide the function pointers in a `pmix_server_module_t` structure passed
11 to `PMIx_server_init`. That module structure and associated function references are defined
12 in this section.

▼ Advice to PMIx server hosts ▼

13 For performance purposes, the host server is required to return as quickly as possible from all
14 functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx
15 server support library to handle multiple client requests as quickly and scalably as possible.

16 All data passed to the host server functions is "owned" by the PMIX server support library and
17 must not be free'd. Data returned by the host server via callback function is owned by the host
18 server, which is free to release it upon return from the callback

19 11.3.1 `pmix_server_module_t` Module

20 Summary

21 List of function pointers that a PMIx server passes to `PMIx_server_init` during startup.

22 Format

```

1 typedef struct pmix_server_module_3_0_0_t
2     /* v1x interfaces */
3     pmix_server_client_connected_fn_t    client_connected;
4     pmix_server_client_finalized_fn_t    client_finalized;
5     pmix_server_abort_fn_t               abort;
6     pmix_server_fence_nb_fn_t            fence_nb;
7     pmix_server_dmodex_req_fn_t          direct_modex;
8     pmix_server_publish_fn_t             publish;
9     pmix_server_lookup_fn_t              lookup;
10    pmix_server_unpublish_fn_t            unpublish;
11    pmix_server_spawn_fn_t                 spawn;
12    pmix_server_connect_fn_t               connect;
13    pmix_server_disconnect_fn_t            disconnect;
14    pmix_server_register_events_fn_t       register_events;
15    pmix_server_deregister_events_fn_t     deregister_events;
16    pmix_server_listener_fn_t              listener;
17    /* v2x interfaces */
18    pmix_server_notify_event_fn_t          notify_event;
19    pmix_server_query_fn_t                 query;
20    pmix_server_tool_connection_fn_t       tool_connected;
21    pmix_server_log_fn_t                   log;
22    pmix_server_alloc_fn_t                 allocate;
23    pmix_server_job_control_fn_t           job_control;
24    pmix_server_monitor_fn_t               monitor;
25    /* v3x interfaces */
26    pmix_server_get_cred_fn_t              get_credential;
27    pmix_server_validate_cred_fn_t         validate_credential;
28    pmix_server_iof_fn_t                   iof_pull;
29    pmix_server_stdin_fn_t                 push_stdin;
30    /* v4x interfaces */
31    pmix_server_grp_fn_t                   group;
32    pmix_server_fabric_fn_t                fabric;
33    pmix_server_module_t;

```

34 11.3.2 pmix_server_client_connected_fn_t

35 Summary

36 Notify the host server that a client connected to this server.

1
PMIx v1.0

Format

C

```
typedef pmix_status_t (*pmix_server_client_connected_fn_t) (  
    const pmix_proc_t *proc,  
    void* server_object,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

C

7 **IN** `proc`
8 `pmix_proc_t` structure (handle)
9 **IN** `server_object`
10 object reference (memory reference)
11 **IN** `cbfunc`
12 Callback function `pmix_op_cbfunc_t` (function reference)
13 **IN** `cbdata`
14 Data to be passed to the callback function (memory reference)

15 Returns one of the following:

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

Description

23 Notify the host environment that a client has called **PMIx_Init**. Note that the client will be in a
24 blocked state until the host server executes the callback function, thus allowing the PMIx server
25 support library to release the client. The `server_object` parameter will be the value of the
26 `server_object` parameter passed to **PMIx_server_register_client** by the host server
27 when registering the connecting client. If provided, an implementation of
28 `pmix_server_client_connected_fn_t` is only required to call the callback function
29 designated. A host server can choose to not be notified when clients connect by setting
30 `pmix_server_client_connected_fn_t` to **NULL**.

31
32 It is possible that only a subset of the clients in a namespace call **PMIx_Init**. The server's
33 `pmix_server_client_connected_fn_t` implementation should not depend on being
34 called once per rank in a namespace or delay calling the callback function until all ranks have
35 connected. However, if a rank makes any PMIx calls, it must first call **PMIx_Init** and therefore
36 the server's `pmix_server_client_connected_fn_t` will be called before any other
37 server functions specific to the rank.

Advice to PMIx server hosts

This operation is an opportunity for a host environment to update the status of the ranks it manages. It is also a convenient and well defined time to perform initialization necessary to support further calls into the server related to that rank.

11.3.3 pmix_server_client_finalized_fn_t

Summary

Notify the host environment that a client called `PMIx_Finalize`.

Format

PMIx v1.0

```
typedef pmix_status_t (*pmix_server_client_finalized_fn_t) (  
    const pmix_proc_t *proc,  
    void* server_object,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

IN `proc`

`pmix_proc_t` structure (handle)

IN `server_object`

object reference (memory reference)

IN `cbfunc`

Callback function `pmix_op_cbfunc_t` (function reference)

IN `cbdata`

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

Returns one of the following:

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

Description

Notify the host environment that a client called `PMIx_Finalize`. Note that the client will be in a blocked state until the host server executes the callback function, thus allowing the PMIx server support library to release the client. The `server_object` parameter will be the value of the `server_object` parameter passed to `PMIx_server_register_client` by the host server when registering the connecting client. If provided, an implementation of `pmix_server_client_finalized_fn_t` is only required to call the callback function designated. A host server can choose to not be notified when clients finalize by setting `pmix_server_client_finalized_fn_t` to `NULL`.

Note that the host server is only being informed that the client has called `PMIx_Finalize`. The client might not have exited. If a client exits without calling `PMIx_Finalize`, the server support library will not call the `pmix_server_client_finalized_fn_t` implementation.

Advice to PMIx server hosts

This operation is an opportunity for a host server to update the status of the tasks it manages. It is also a convenient and well defined time to release resources used to support that client.

11.3.4 `pmix_server_abort_fn_t`

Summary

Notify the host environment that a local client called `PMIx_Abort`.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_abort_fn_t) (  
    const pmix_proc_t *proc,  
    void *server_object,  
    int status,  
    const char msg[],  
    pmix_proc_t procs[],  
    size_t nprocs,  
    pmix_op_cbfnc_t cbfunc,  
    void *cbdata)
```

```

1  IN  proc
2      pmix_proc_t structure identifying the process requesting the abort (handle)
3  IN  server_object
4      object reference (memory reference)
5  IN  status
6      exit status (integer)
7  IN  msg
8      exit status message (string)
9  IN  procs
10     Array of pmix_proc_t structures identifying the processes to be terminated (array of
11     handles)
12  IN  nprocs
13     Number of elements in the procs array (integer)
14  IN  cbfunc
15     Callback function pmix_op_cbfunc_t (function reference)
16  IN  cbdata
17     Data to be passed to the callback function (memory reference)

```

18 Returns one of the following:

- 19 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
20 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
21 prior to returning from the API.
- 22 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
23 returned *success* - the *cbfunc* will not be called
- 24 • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
25 request, even though the function entry was provided in the server module - the *cbfunc* will not
26 be called
- 27 • a PMIx error constant indicating either an error in the input or that the request was immediately
28 processed and failed - the *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 **11.3.5 pmix_server_fence_nb_fn_t**

35 **Summary**

36 At least one client called either **PMIx_Fence** or **PMIx_Fence_nb** .

Format

C

```
2 typedef pmix_status_t (*pmix_server_fence_fn_t) (  
3     const pmix_proc_t procs[],  
4     size_t nprocs,  
5     const pmix_info_t info[],  
6     size_t ninfo,  
7     char *data, size_t ndata,  
8     pmix_modex_cbfunc_t cbfunc,  
9     void *cbdata)
```

C

- 10 **IN** `procs`
11 Array of `pmix_proc_t` structures identifying operation participants(array of handles)
- 12 **IN** `nprocs`
13 Number of elements in the `procs` array (integer)
- 14 **IN** `info`
15 Array of info structures (array of handles)
- 16 **IN** `ninfo`
17 Number of elements in the `info` array (integer)
- 18 **IN** `data`
19 (string)
- 20 **IN** `ndata`
21 (integer)
- 22 **IN** `cbfunc`
23 Callback function `pmix_modex_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

PMIx libraries are required to pass any provided attributes to the host environment for processing.

The following attributes are required to be supported by all host environments:

PMIX_COLLECT_DATA "pmix.collect" (bool)

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

Optional Attributes

The following attributes are optional for host environments:

PMIX_TIMEOUT "pmix.timeout" (int)

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

PMIX_COLLECTIVE_ALGO "pmix.calgo" (char*)

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

PMIX_COLLECTIVE_ALGO_REQD "pmix.calreqd" (bool)

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

Advice to PMIx server hosts

Host environment are required to return **PMIX_ERR_NOT_SUPPORTED** if passed an attributed marked as **PMIX_INFO_REQD** that they do not support, even if support for that attribute is optional.

Description

All local clients in the provided array of *procs* called either `PMIx_Fence` or `PMIx_Fence_nb`. In either case, the host server will be called via a non-blocking function to execute the specified operation once all participating local processes have contributed. All processes in the specified *procs* array are required to participate in the `PMIx_Fence` / `PMIx_Fence_nb` operation. The callback is to be executed once every daemon hosting at least one participant has called the host server's `pmix_server_fence_nb_fn_t` function.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

The host will receive a single call for each collective operation. It is the responsibility of the host to identify the nodes containing participating processes, execute the collective across all participating nodes, and notify the local PMIx server library upon completion of the global collective. Data received from each node must be simply concatenated to form an aggregated unit, as shown in the following example:

```
C
uint8_t *blob1, *blob2, *total;
size_t sz_blob1, sz_blob2, sz_total;

sz_total = sz_blob1 + sz_blob2;
total = (uint8_t*)malloc(sz_total);
memcpy(total, blob1, sz_blob1);
memcpy(&total[sz_blob1], blob2, sz_blob2);
```

Note that the ordering of the data blobs does not matter.

The provided data is to be collectively shared with all PMIx servers involved in the fence operation, and returned in the modex *cbfunc*. A **NULL** data value indicates that the local processes had no data to contribute.

The array of *info* structs is used to pass user-requested options to the server. This can include directives as to the algorithm to be used to execute the fence operation. The directives are optional unless the `PMIX_INFO_REQD` flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

1 11.3.6 pmix_server_dmodex_req_fn_t

2 Summary

3 Used by the PMIx server to request its local host contact the PMIx server on the remote node that
4 hosts the specified proc to obtain and return a direct modex blob for that proc.

5 Format

PMIx v1.0

```
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)
```

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

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Description

Used by the PMIx server to request its local host contact the PMIx server on the remote node that hosts the specified proc to obtain and return any information that process posted via calls to **PMIx_Put** and **PMIx_Commit**.

The array of *info* structs is used to pass user-requested options to the server. This can include a timeout to preclude an indefinite wait for data that may never become available. The directives are optional unless the *mandatory* flag has been set - in such cases, the host RM is required to return an error if the directive cannot be met.

11.3.7 pmix_server_publish_fn_t

Summary

Publish data per the PMIx API specification.

Format

PMIx v1.0

```
typedef pmix_status_t (*pmix_server_publish_fn_t) (  
    const pmix_proc_t *proc,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

IN **proc**
pmix_proc_t structure of the process publishing the data (handle)

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

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

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

1 **IN cldata**

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

3 Returns one of the following:

- 4 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
5 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
6 prior to returning from the API.
- 7 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
8 returned *success* - the *cbfunc* will not be called
- 9 • **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 _____
21 Host environments that implement this entry point are required to support the following attributes:

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

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

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

25 Value for calls to **PMIx_Publish**.

▲----- Optional Attributes -----▲

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

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.

Description

Publish data per the **PMIx_Publish** specification. The callback is to be executed upon completion of the operation. The default data range is left to the host environment, but expected to be **PMIX_RANGE_SESSION**, and the default persistence **PMIX_PERSIST_SESSION** or their equivalent. These values can be specified by including the respective attributed in the *info* array.

The persistence indicates how long the server should retain the data.

Advice to PMIx server hosts

The host environment is not required to guarantee support for any specific range - i.e., the environment does not need to return an error if the data store doesn't support a specified range so long as it is covered by some internally defined range. However, the server must return an error (a) if the key is duplicative within the storage range, and (b) if the server does not allow overwriting of published info by the original publisher - it is left to the discretion of the host environment to allow info-key-based flags to modify this behavior.

The **PMIX_USERID** and **PMIX_GRPID** of the publishing process will be provided to support authorization-based access to published information and must be returned on any subsequent lookup request.

11.3.8 pmix_server_lookup_fn_t

Summary

Lookup published data.

Format

PMIx v1.0

C

```
typedef pmix_status_t (*pmix_server_lookup_fn_t) (
    const pmix_proc_t *proc,
    char **keys,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_lookup_cbfnc_t cbfunc,
    void *cbdata)
```

```

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

```

13 Returns one of the following:

- 14 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
15 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
16 prior to returning from the API.
- 17 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
18 returned *success* - the *cbfunc* will not be called
- 19 • **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the
20 request, even though the function entry was provided in the server module - the *cbfunc* will not
21 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 required to pass any provided attributes to the host environment for processing.
25 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.

30
31 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

8 Lookup published data. The host server will be passed a **NULL**-terminated array of string keys
9 identifying the data being requested.

10
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.

22 11.3.9 pmix_server_unpublish_fn_t

23 Summary

24 Delete data from the data store.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

Format

PMIx v1.0

C

```

typedef pmix_status_t (*pmix_server_unpublish_fn_t) (
    const pmix_proc_t *proc,
    char **keys,
    const pmix_info_t info[],
    size_t ninfo,
    pmix_op_cbfunc_t cbfunc,
    void *cbdata)

```

C

- IN **proc**
[pmix_proc_t](#) structure identifying the process making the request (handle)
- IN **keys**
(array of strings)
- IN **info**
Array of info structures (array of handles)
- IN **ninfo**
Number of elements in the *info* array (integer)
- IN **cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)
- IN **cbdata**
Data to be passed to the callback function (memory reference)

Returns one of the following:

- [PMIX_SUCCESS](#) , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- [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

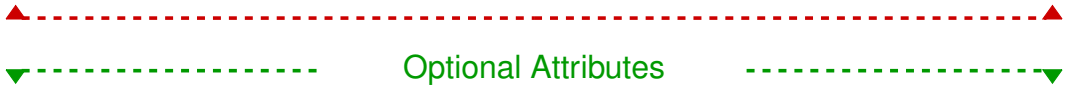
PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

- PMIX_USERID** "pmix.euid" ([uint32_t](#))
Effective user id.

1 **PMIX_GRPID** "pmix.egid" (uint32_t)
2 Effective group id.

3
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.



12 **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_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.



22 **11.3.10 pmix_server_spawn_fn_t**

23 **Summary**

24 Spawn a set of applications/processes as per the **PMIx_Spawn** API.

1
PMIx v1.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_spawn_fn_t) (  
3     const pmix_proc_t *proc,  
4     const pmix_info_t job_info[],  
5     size_t ninfo,  
6     const pmix_app_t apps[],  
7     size_t napps,  
8     pmix_spawn_cbfunc_t cbfunc,  
9     void *cbdata)
```

C

10 **IN** `proc`
11 `pmix_proc_t` structure of the process making the request (handle)
12 **IN** `job_info`
13 Array of info structures (array of handles)
14 **IN** `ninfo`
15 Number of elements in the `jobinfo` array (integer)
16 **IN** `apps`
17 Array of `pmix_app_t` structures (array of handles)
18 **IN** `napps`
19 Number of elements in the `apps` array (integer)
20 **IN** `cbfunc`
21 Callback function `pmix_spawn_cbfunc_t` (function reference)
22 **IN** `cbdata`
23 Data to be passed to the callback function (memory reference)

24 Returns one of the following:

- 25 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
26 will be returned in the provided `cbfunc`. Note that the host must not invoke the callback function
27 prior to returning from the API.
- 28 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
29 returned `success` - the `cbfunc` will not be called
- 30 • **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the
31 request, even though the function entry was provided in the server module - the `cbfunc` will not
32 be called
- 33 • a PMIx error constant indicating either an error in the input or that the request was immediately
34 processed and failed - the `cbfunc` will not be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

Host environments that provide this module entry point are required to pass the **PMIX_SPAWNED** and **PMIX_PARENT_ID** attributes to all PMIx servers launching new child processes so those values can be returned to clients upon connection to the PMIx server. In addition, they are required to support the following attributes when present in either the *job_info* or the *info* array of an element of the *apps* array:

PMIX_WDIR "pmix.wdir" (char*)

Working directory for spawned processes.

PMIX_SET_SESSION_CWD "pmix.ssn cwd" (bool)

Set the application's current working directory to the session working directory assigned by the RM - when accessed using **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the session working directory assigned to the provided namespace

PMIX_PREFIX "pmix.prefix" (char*)

Prefix to use for starting spawned processes.

PMIX_HOST "pmix.host" (char*)

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

PMIX_HOSTFILE "pmix.hostfile" (char*)

Hostfile to use for spawned processes.

Optional Attributes

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

PMIX_ADD_HOSTFILE "pmix.addhostfile" (char*)

Hostfile listing hosts to add to existing allocation.

PMIX_ADD_HOST "pmix.addhost" (char*)

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

PMIX_PRELOAD_BIN "pmix.preloadbin" (bool)

Preload binaries onto nodes.

PMIX_PRELOAD_FILES "pmix.preloadfiles" (char*)

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 Spawned processes will not call **PMIx_Init**.

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

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

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

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

30 **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)

31 Forward **stdout** from spawned processes to this process.

32 **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)

33 Forward **stderr** from spawned processes to this process.

34 **PMIX_DEBUGGER_DAEMONS** "pmix.debugger" (bool)

35 Spawned application consists of debugger daemons.

36 **PMIX_TAG_OUTPUT** "pmix.tagout" (bool)

1 Tag application output with the identity of the source process.

2 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)

3 Timestamp output from applications.

4 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)

5 Merge **stdout** and **stderr** streams from application processes.

6 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)

7 Output application output to the specified file.

8 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)

9 Mark the **argv** with the rank of the process.

10 **PMIX_CPUS_PER_PROC** "pmix.cpusperproc" (uint32_t)

11 Number of cpus to assign to each rank - when accessed using **PMIx_Get** , use the
12 **PMIX_RANK_WILDCARD** value for the rank to discover the cpus/process assigned to the
13 provided namespace

14 **PMIX_NO_PROCS_ON_HEAD** "pmix.nolocal" (bool)

15 Do not place processes on the head node.

16 **PMIX_NO_OVERSUBSCRIBE** "pmix.noover" (bool)

17 Do not oversubscribe the cpus.

18 **PMIX_REPORT_BINDINGS** "pmix.repbinding" (bool)

19 Report bindings of the individual processes.

20 **PMIX_CPU_LIST** "pmix.cpulist" (char*)

21 List of cpus to use for this job - when accessed using **PMIx_Get** , use the
22 **PMIX_RANK_WILDCARD** value for the rank to discover the cpu list used for the provided
23 namespace

24 **PMIX_JOB_RECOVERABLE** "pmix.recover" (bool)

25 Application supports recoverable operations.

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

27 Application is continuous, all failed processes should be immediately restarted.

28 **PMIX_MAX_RESTARTS** "pmix.maxrestarts" (uint32_t)

29 Maximum number of times to restart a job - when accessed using **PMIx_Get** , use the
30 **PMIX_RANK_WILDCARD** value for the rank to discover the max restarts for the provided
31 namespace

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

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



Description

Spawn a set of applications/processes as per the [PMIx_Spawn](#) API. Note that applications are not required to be MPI or any other programming model. Thus, the host server cannot make any assumptions as to their required support. The callback function is to be executed once all processes have been started. An error in starting any application or process in this request shall cause all applications and processes in the request to be terminated, and an error returned to the originating caller.

Note that a timeout can be specified in the `job_info` array to indicate that failure to start the requested job within the given time should result in termination to avoid hangs.

11.3.11 `pmix_server_connect_fn_t`

Summary

Record the specified processes as *connected*.

Format

PMIx v1.0

```
typedef pmix_status_t (*pmix_server_connect_fn_t) (  
    const pmix_proc_t procs[],  
    size_t nprocs,  
    const pmix_info_t info[],  
    size_t ninfo,  
    pmix_op_cbfunc_t cbfunc,  
    void *cbdata)
```

IN `procs`

Array of [pmix_proc_t](#) structures identifying participants (array of handles)

IN `nprocs`

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

IN `info`

Array of info structures (array of handles)

IN `ninfo`

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

IN `cbfunc`

Callback function [pmix_op_cbfunc_t](#) (function reference)

IN `cbdata`

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

Returns one of the following:

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

- 1 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
2 returned *success* - the *cbfunc* will not be called
- 3 • **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the
4 request, even though the function entry was provided in the server module - the *cbfunc* will not
5 be called
- 6 • a PMIx error constant indicating either an error in the input or that the request was immediately
7 processed and failed - the *cbfunc* will not be called

Required Attributes

8 PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

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

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

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

14 **PMIX_COLLECTIVE_ALGO** "pmix.calgo" (char*)

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

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

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

1
2
3
4
5

6
7

8
9
10

11
12
13

Description

Record the processes specified by the *procs* array as *connected* as per the PMIx definition. The callback is to be executed once every daemon hosting at least one participant has called the host server’s `pmix_server_connect_fn_t` function, and the host environment has completed any supporting operations required to meet the terms of the PMIx definition of *connected* processes.

▼ **Advice to PMIx library implementers** ▼

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

▼ **Advice to PMIx server hosts** ▼

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

11.3.12 pmix_server_disconnect_fn_t

Summary

Disconnect a previously connected set of processes.

1
PMIx v1.0

Format

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**
Array of [pmix_proc_t](#) structures identifying participants (array of handles)
- 11 **IN nprocs**
Number of elements in the *procs* array (integer)
- 13 **IN info**
Array of info structures (array of handles)
- 15 **IN ninfo**
Number of elements in the *info* array (integer)
- 17 **IN cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)
- 19 **IN cbdata**
Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22 • [PMIX_SUCCESS](#) , indicating that the request is being processed by the host environment - result
23 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
24 prior to returning from the API.
- 25 • [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and
26 returned *success* - the *cbfunc* will not be called
- 27 • [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

Required Attributes

32 PMIx libraries are required to pass any provided attributes to the host environment for processing.

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Description

Disconnect a previously connected set of processes. The callback is to be executed once every daemon hosting at least one participant has called the host server’s `pmix_server_disconnect_fn_t` function, and the host environment has completed any required supporting operations.

Advice to PMIx library implementers

The PMIx server library is required to aggregate participation by local clients, passing the request to the host environment once all local participants have executed the API.

Advice to PMIx server hosts

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

A **PMIX_ERR_INVALID_OPERATION** error must be returned if the specified set of *procs* was not previously *connected* via a call to the `pmix_server_connect_fn_t` function.

11.3.13 pmix_server_register_events_fn_t

Summary

Register to receive notifications for the specified events.

1
PMIx v1.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_register_events_fn_t) (  
3     pmix_status_t *codes,  
4     size_t ncodes,  
5     const pmix_info_t info[],  
6     size_t ninfo,  
7     pmix_op_cbfunc_t cbfunc,  
8     void *cbdata)
```

C

- 9 **IN codes**
Array of [pmix_status_t](#) values (array of handles)
- 10 **IN ncodes**
Number of elements in the *codes* array (integer)
- 11 **IN info**
Array of info structures (array of handles)
- 12 **IN ninfo**
Number of elements in the *info* array (integer)
- 13 **IN cbfunc**
Callback function [pmix_op_cbfunc_t](#) (function reference)
- 14 **IN cbdata**
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 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- 17 • [PMIX_OPERATION_SUCCEEDED](#), indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- 18 • [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
- 19 • 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

20 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:

21 **PMIX_USERID** "pmix.euid" ([uint32_t](#))
Effective user id.

1 **PMIX_GRPID** "pmix.egid" (uint32_t)
2 Effective group id.

Description

3 Register to receive notifications for the specified status codes. The *info* array included in this API is
4 reserved for possible future directives to further steer notification.
5

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 11.3.14 pmix_server_deregister_events_fn_t

18 Summary

19 Deregister to receive notifications for the specified events.

1
PMIx v1.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_deregister_events_fn_t) (
3     pmix_status_t *codes,
4     size_t ncodes,
5     pmix_op_cbfunc_t cbfunc,
6     void *cbdata)

```

C

- 7 **IN codes**
Array of `pmix_status_t` values (array of handles)
- 8
- 9 **IN ncodes**
Number of elements in the `codes` array (integer)
- 10
- 11 **IN cbfunc**
Callback function `pmix_op_cbfunc_t` (function reference)
- 12
- 13 **IN cbdata**
Data to be passed to the callback function (memory reference)
- 14

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

Description

26 Deregister to receive notifications for the specified events to which the PMIx server has previously
27 registered.
28

Advice to PMIx library implementers

29 The PMIx server library must track all client registrations. This module function shall only be
30 called when:

- 31 • the library is deregistering environmental codes (i.e., a PMIx codes in the range between
32 **PMIX_ERR_SYS_BASE** and **PMIX_ERR_SYS_OTHER**, inclusive) or codes that lies outside
33 the defined PMIx range of constants; and

- no client (including the server library itself) remains registered for notifications on any included code - i.e., a code should be included in this call only when no registered notifications against it remain.

11.3.15 pmix_server_notify_event_fn_t

Summary

Notify the specified processes of an event.

Format

PMIx v2.0

```
typedef pmix_status_t (*pmix_server_notify_event_fn_t) (pmix_status_t code,
                                                       const pmix_proc_t *source,
                                                       pmix_data_range_t range,
                                                       pmix_info_t info[],
                                                       size_t ninfo,
                                                       pmix_op_cbfunc_t cbfunc,
                                                       void *cbdata);
```

IN code

The `pmix_status_t` event code being referenced structure (handle)

IN source

`pmix_proc_t` of process that generated the event (handle)

IN range

`pmix_data_range_t` range over which the event is to be distributed (handle)

IN info

Optional array of `pmix_info_t` structures containing additional information on the event (array of handles)

IN ninfo

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

IN cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

IN cbdata

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

Returns one of the following:

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

- **PMIX_ERR_NOT_SUPPORTED** , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing.

Host environments that provide this module entry point are required to support the following attributes:

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

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

Description

Notify the specified processes (described through a combination of *range* and attributes provided in the *info* array) of an event generated either by the PMIx server itself or by one of its local clients. The process generating the event is provided in the *source* parameter, and any further descriptive information is included in the *info* array.

Advice to PMIx server hosts

The callback function is to be executed once the host environment no longer requires that the PMIx server library maintain the provided data structures. It does not necessarily indicate that the event has been delivered to any process, nor that the event has been distributed for delivery

11.3.16 pmix_server_listener_fn_t

Summary

Register a socket the host server can monitor for connection requests.

1
PMIx v1.0

Format

C

```
2 typedef pmix_status_t (*pmix_server_listener_fn_t) (  
3     int listening_sd,  
4     pmix_connection_cbfunc_t cbfunc,  
5     void *cbdata)
```

C

6 **IN** `incoming_sd`
7 (integer)
8 **IN** `cbfunc`
9 Callback function `pmix_connection_cbfunc_t` (function reference)
10 **IN** `cbdata`
11 (memory reference)

12 Returns `PMIX_SUCCESS` indicating that the request is accepted, or a negative value
13 corresponding to a PMIx error constant indicating that the request has been rejected.

Description

14 Register a socket the host environment can monitor for connection requests, harvest them, and then
15 call the PMIx server library's internal callback function for further processing. A listener thread is
16 essential to efficiently harvesting connection requests from large numbers of local clients such as
17 occur when running on large SMPs. The host server listener is required to call `accept` on the
18 incoming connection request, and then pass the resulting socket to the provided `cbfunc`. A `NULL`
19 for this function will cause the internal PMIx server to spawn its own listener thread.
20

21 11.3.17 `pmix_server_query_fn_t`

22 Summary

23 Query information from the resource manager.

24 Format

PMIx v2.0

C

```
25 typedef pmix_status_t (*pmix_server_query_fn_t) (  
26     pmix_proc_t *proct,  
27     pmix_query_t *queries, size_t nqueries,  
28     pmix_info_cbfunc_t cbfunc,  
29     void *cbdata)
```

C

30 **IN** `proct`
31 `pmix_proc_t` structure of the requesting process (handle)
32 **IN** `queries`
33 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 cbdata**
6 Data to be passed to the callback function (memory reference)

7 Returns one of the following:

- 8 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
9 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
10 prior to returning from the API.
- 11 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
12 returned *success* - the *cbfunc* will not be called
- 13 • **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the
14 request, even though the function entry was provided in the server module - the *cbfunc* will not
15 be called
- 16 • a PMIx error constant indicating either an error in the input or that the request was immediately
17 processed and failed - the *cbfunc* will not be called

▼----- Required Attributes -----▼

18 PMIx libraries are required to pass any provided attributes to the host environment for processing.
19 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 **PMIX_GRPID** "pmix.egid" (uint32_t)
23 Effective group id.



▼----- Optional Attributes -----▼

24 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.

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

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

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

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

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

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

10 **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
 11 Return a comma-delimited list of supported debug attributes.

12 **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)
 13 Return information on memory usage for the processes indicated in the qualifiers.

14 **PMIX_QUERY_LOCAL_ONLY** "pmix.qry.local" (bool)
 15 Constrain the query to local information only.

16 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
 17 Report only average values for sampled information.

18 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)
 19 Report minimum and maximum values.

20 **PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (char*)
 21 String identifier of the allocation whose status is being requested.

22 **PMIX_TIME_REMAINING** "pmix.time.remaining" (char*)
 23 Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.
 24



25 Description

26 Query information from the host environment. The query will include the namespace/rank of the
 27 process that is requesting the info, an array of **pmix_query_t** describing the request, and a
 28 callback function/data for the return.

Advice to PMIx library implementers

29 The PMIx server library should not block in this function as the host environment may, depending
 30 upon the information being requested, require significant time to respond.



1 11.3.18 pmix_server_tool_connection_fn_t

2 Summary

3 Register that a tool has connected to the server.

4 Format

PMIx v2.0

C

```
5 typedef void (*pmix_server_tool_connection_fn_t) (  
6             pmix_info_t info[], size_t ninfo,  
7             pmix_tool_connection_cbfunc_t cbfunc,  
8             void *cbdata)
```

C

- 9 **IN info**
10 Array of [pmix_info_t](#) structures (array of handles)
- 11 **IN ninfo**
12 Number of elements in the *info* array (integer)
- 13 **IN cbfunc**
14 Callback function [pmix_tool_connection_cbfunc_t](#) (function reference)
- 15 **IN cbdata**
16 Data to be passed to the callback function (memory reference)

Required Attributes

17 PMIx libraries are required to pass the following attributes in the *info* array:

18 **PMIX_USERID** "pmix.euid" (uint32_t)
19 Effective user id.

20 **PMIX_GRPID** "pmix.egid" (uint32_t)
21 Effective group id.

Optional Attributes

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

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

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

27 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)
28 Forward this process's **stdin** to the designated process.

Description

Register that a tool has connected to the server, and request that the tool be assigned a namespace/rank identifier for further interactions. The `pmix_info_t` array is used to pass qualifiers for the connection request, including the effective uid and gid of the calling tool for authentication purposes.

Advice to PMIx server hosts

The host environment is solely responsible for authenticating and authorizing the connection, and for authorizing all subsequent tool requests. The host must not execute the callback function prior to returning from the API.

11.3.19 pmix_server_log_fn_t

Summary

Log data on behalf of a client.

Format

PMIx v2.0

```
typedef void (*pmix_server_log_fn_t) (  
    const pmix_proc_t *client,  
    const pmix_info_t data[], size_t ndata,  
    const pmix_info_t directives[], size_t ndirs,  
    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

IN client

`pmix_proc_t` structure (handle)

IN data

Array of info structures (array of handles)

IN ndata

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

IN directives

Array of info structures (array of handles)

IN ndirs

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

IN cbfunc

Callback function `pmix_op_cbfunc_t` (function reference)

IN cbdata

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

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_LOG_STDERR "pmix.log.stderr" (char*)

Log string to **stderr**.

PMIX_LOG_STDOUT "pmix.log.stdout" (char*)

Log string to **stdout**.

PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)

Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available, otherwise to local syslog

Optional Attributes

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

PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)

Message blob to be sent somewhere.

PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)

Log via email based on **pmix_info_t** containing directives.

PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)

Comma-delimited list of email addresses that are to receive the message.

PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)

Subject line for email.

PMIX_LOG_EMAIL_MSG "pmix.log.emmsg" (char*)

Message to be included in email.

Description

Log data on behalf of a client. This function is not intended for output of computational results, but rather for reporting status and error messages. The host must not execute the callback function prior to returning from the API.

11.3.20 pmix_server_alloc_fn_t

Summary

Request allocation operations on behalf of a client.

Format

PMIx v2.0

```
typedef pmix_status_t (*pmix_server_alloc_fn_t) (  
    const pmix_proc_t *client,  
    pmix_alloc_directive_t directive,  
    const pmix_info_t data[], size_t ndata,  
    pmix_info_cbfunc_t cbfunc, void *cbdata)
```

IN client

[pmix_proc_t](#) structure of process making request (handle)

IN directive

Specific action being requested ([pmix_alloc_directive_t](#))

IN data

Array of info structures (array of handles)

IN ndata

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

IN cbfunc

Callback function [pmix_info_cbfunc_t](#) (function reference)

IN cbdata

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

Returns one of the following:

- [PMIX_SUCCESS](#) , indicating that the request is being processed by the host environment - result will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function prior to returning from the API.
- [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- [PMIX_ERR_NOT_SUPPORTED](#) , indicating that the host environment does not support the request, even though the function entry was provided in the server module - the *cbfunc* will not be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will not be called

Required Attributes

PMIx libraries are required to pass any provided attributes to the host environment for processing. In addition, the following attributes are required to be included in the passed *info* array:

PMIX_USERID "pmix.euid" (uint32_t)

Effective user id.

PMIX_GRPID "pmix.egid" (uint32_t)

Effective group id.

Host environments that provide this module entry point are required to support the following attributes:

PMIX_ALLOC_ID "pmix.alloc.id" (char*)

A string identifier (provided by the host environment) for the resulting allocation which can later be used to reference the allocated resources in, for example, a call to **PMIx_Spawn**.

PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)

The number of nodes.

PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)

Number of cpus.

PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)

Time in seconds.

Optional Attributes

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

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

Regular expression of the specific nodes.

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

Regular expression of the number of cpus for each node.

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

Regular expression of the specific cpus indicating the cpus involved.

PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)

Number of Megabytes.

PMIX_ALLOC_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*)

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 Mbits/sec.

16 `PMIX_ALLOC_FABRIC_QOS` "pmix.alloc.netqos" (char*)
17 Quality of service level.



18 Description

19 Request new allocation or modifications to an existing allocation on behalf of a client. Several
20 broad categories are envisioned, including the ability to:

- 21 • Request allocation of additional resources, including memory, bandwidth, and compute for an
22 existing allocation. Any additional allocated resources will be considered as part of the current
23 allocation, and thus will be released at the same time.
- 24 • Request a new allocation of resources. Note that the new allocation will be disjoint from (i.e., not
25 affiliated with) the allocation of the requestor - thus the termination of one allocation will not
26 impact the other.
- 27 • Extend the reservation on currently allocated resources, subject to scheduling availability and
28 priorities.
- 29 • Return no-longer-required resources to the scheduler. This includes the *loan* of resources back to
30 the scheduler with a promise to return them upon subsequent request.

31 The callback function provides a *status* to indicate whether or not the request was granted, and to
32 provide some information as to the reason for any denial in the `pmix_info_cbfunc_t` array of
33 `pmix_info_t` structures.

34 11.3.21 pmix_server_job_control_fn_t

35 Summary

36 Execute a job control action on behalf of a client.

1
PMIx v2.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_job_control_fn_t) (
3     const pmix_proc_t *requestor,
4     const pmix_proc_t targets[], size_t ntargets,
5     const pmix_info_t directives[], size_t ndirs,
6     pmix_info_cbfunc_t cbfunc, void *cbdata)

```

C

- 7 **IN requestor**
- 8 [pmix_proc_t](#) structure of requesting process (handle)
- 9 **IN targets**
- 10 Array of proc structures (array of handles)
- 11 **IN ntargets**
- 12 Number of elements in the *targets* array (integer)
- 13 **IN directives**
- 14 Array of info structures (array of handles)
- 15 **IN ndirs**
- 16 Number of elements in the *info* array (integer)
- 17 **IN cbfunc**
- 18 Callback function [pmix_op_cbfunc_t](#) (function reference)
- 19 **IN cbdata**
- 20 Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22 • [PMIX_SUCCESS](#) , indicating that the request is being processed by the host environment - result
- 23 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
- 24 prior to returning from the API.
- 25 • [PMIX_OPERATION_SUCCEEDED](#) , indicating that the request was immediately processed and
- 26 returned *success* - the *cbfunc* will not be called
- 27 • [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

Required Attributes

32 PMIx libraries are required to pass any attributes provided by the client to the host environment for
33 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_GRPID** "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** -----▼

20 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.pvning" (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

11 Execute a job control action on behalf of a client. The *targets* array identifies the processes to
 12 which the requested job control action is to be applied. A **NULL** value can be used to indicate all
 13 processes in the caller's namespace. The use of **PMIX_RANK_WILDCARD** can also be used to
 14 indicate that all processes in the given namespace are to be included.

15 The directives are provided as pmix_info_t structures in the *directives* array. The callback
 16 function provides a *status* to indicate whether or not the request was granted, and to provide some
 17 information as to the reason for any denial in the pmix_info_cbfunc_t array of
 18 pmix_info_t structures.

20 11.3.22 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 cbdata**
 14 Data to be passed to the callback function (memory reference)

15 Returns one of the following:

- 16 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
 17 will be returned in the provided *cbfunc*. Note that the host must not invoke the callback function
 18 prior to returning from the API.
- 19 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
 20 returned *success* - the *cbfunc* will not be called
- 21 • **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
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

▲-----▲

▼-----▼

Optional Attributes

The following attributes may be implemented by a host environment.

PMIX_MONITOR_ID "pmix.monitor.id" (char*)

Provide a string identifier for this request.

PMIX_MONITOR_CANCEL "pmix.monitor.cancel" (char*)

Identifier to be canceled (NULL means cancel all monitoring for this process).

PMIX_MONITOR_APP_CONTROL "pmix.monitor.appctrl" (bool)

The application desires to control the response to a monitoring event.

PMIX_MONITOR_HEARTBEAT "pmix.monitor.mbeat" (void)

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

PMIX_MONITOR_HEARTBEAT_TIME "pmix.monitor.btime" (uint32_t)

Time in seconds before declaring heartbeat missed.

PMIX_MONITOR_HEARTBEAT_DROPS "pmix.monitor.bdrop" (uint32_t)

Number of heartbeats that can be missed before generating the event.

PMIX_MONITOR_FILE "pmix.monitor.fmon" (char*)

Register to monitor file for signs of life.

PMIX_MONITOR_FILE_SIZE "pmix.monitor.fsize" (bool)

Monitor size of given file is growing to determine if the application is running.

PMIX_MONITOR_FILE_ACCESS "pmix.monitor.faccess" (char*)

Monitor time since last access of given file to determine if the application is running.

PMIX_MONITOR_FILE_MODIFY "pmix.monitor.fmod" (char*)

Monitor time since last modified of given file to determine if the application is running.

PMIX_MONITOR_FILE_CHECK_TIME "pmix.monitor.ftime" (uint32_t)

Time in seconds between checking the file.

PMIX_MONITOR_FILE_DROPS "pmix.monitor.fdrop" (uint32_t)

Number of file checks that can be missed before generating the event.

▲-----▲

Description

Request that a client be monitored for activity.

1 11.3.23 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

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

PMIX_CRED_TYPE "pmix.sec ctype" (char*)

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

Request a credential from the host environment

11.3.24 pmix_server_validate_cred_fn_t

Summary

Request validation of a credential

1
PMIx v3.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_validate_cred_fn_t) (
3     const pmix_proc_t *proc,
4     const pmix_byte_object_t *cred,
5     const pmix_info_t directives[],
6     size_t ndirs,
7     pmix_validation_cbfunc_t cbfunc,
8     void *cbdata);

```

C

- 9 **IN** `proc`
 [pmix_proc_t](#) structure of requesting process (handle)
- 10 **IN** `cred`
 Pointer to [pmix_byte_object_t](#) containing the credential (handle)
- 11 **IN** `directives`
 Array of info structures (array of handles)
- 12 **IN** `ndirs`
 Number of elements in the *info* array (integer)
- 13 **IN** `cbfunc`
 Callback function to return the result ([pmix_validation_cbfunc_t](#) function
 reference)
- 14 **IN** `cbdata`
 Data to be passed to the callback function (memory reference)

22 Returns one of the following:

- 23 • **PMIX_SUCCESS**, indicating that the request is being processed by the host environment - result
24 will be returned in the provided *cbfunc*
- 25 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
26 returned *success* - the *cbfunc* will not be called
- 27 • **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

Required Attributes

32 If the PMIx library does not itself validate the credential, then it is required to pass any attributes
33 provided by the client to the host environment for processing. In addition, it must include the
34 following attributes in the passed *info* array:

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

1 Effective user id.
2 **PMIX_GRPID** "pmix.egid" (uint32_t)
3 Effective group id.

4
5 Host environments are not required to support any specific attributes.



6 **Optional Attributes**

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.



11 **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
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 **11.3.25 pmix_server_iof_fn_t**

21 **Summary**

22 Request the specified IO channels be forwarded from the given array of processes.

1
PMIx v3.0

Format

C

```

2 typedef pmix_status_t (*pmix_server_iof_fn_t) (
3     const pmix_proc_t procs[], size_t nprocs,
4     const pmix_info_t directives[], size_t ndirs,
5     pmix_iof_channel_t channels,
6     pmix_op_cbfunc_t cbfunc, void *cbdata);

```

C

- 7 **IN** `procs`
Array `pmix_proc_t` identifiers whose IO is being requested (handle)
- 8 **IN** `nprocs`
Number of elements in `procs` (`size_t`)
- 9 **IN** `directives`
Array of `pmix_info_t` structures further defining the request (array of handles)
- 10 **IN** `ndirs`
Number of elements in the `info` array (integer)
- 11 **IN** `channels`
Bitmask identifying the channels to be forwarded (`pmix_iof_channel_t`)
- 12 **IN** `cbfunc`
Callback function `pmix_op_cbfunc_t` (function reference)
- 13 **IN** `cbdata`
Data to be passed to the callback function (memory reference)

21 Returns one of the following:

- 22 • **PMIX_SUCCESS** , indicating that the request is being processed by the host environment - result
23 will be returned in the provided `cbfunc`. Note that the 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

Required Attributes

32 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`)

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.csize" (uint32_t)

6 The requested size of the server cache in bytes for each specified channel. By default, the
7 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, drop the oldest bytes to make room in the cache.

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

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

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29


Optional Attributes

The following attributes may be supported by a host environment.

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

16 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
17 IO arrives. The library will execute the callback whenever the specified number of bytes
18 becomes available. Any remaining buffered data will be “flushed” upon 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
22 arrive.


Description

Request the specified IO channels be forwarded from the given array of processes. An error shall be returned in the callback function if the requested service from any of the requested processes cannot be provided.



Advice to PMIx library implementers

The forwarding of stdin is a *push* process - processes cannot request that it be *pulled* from some other source. Requests including the **PMIX_FWD_STDIN_CHANNEL** channel will return a **PMIX_ERR_NOT_SUPPORTED** error.



1 11.3.26 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_GRPID "pmix.egid" (uint32_t)

Effective group id.

Description

Passes stdin to the host environment for transmission to specified recipients. The host environment is responsible for forwarding the data to all locations that host the specified *targets* and delivering the payload to the PMIx server library connected to those clients.

11.3.27 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 group (string)
- IN procs**
Array of [pmix_proc_t](#) identifiers of participants (handle)
- IN nprocs**
Number of elements in the *procs* array (integer)
- IN directives**
Array of info structures (array of handles)

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 Returns one of the following:

- 8 • **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.
- 9 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will not be called
- 10 • **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
- 11 • 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 -----▼

18 The following attributes may be supported by a host environment.

19 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)
 20 Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

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

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

34 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)
 35 Participation is optional - do not return an error if any of the specified processes terminate without having joined. The default is false

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

3 The following attributes may be included in the host's response:

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

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

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

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



12 Description

13 Perform the specified operation across the identified processes, plus any special actions included in
14 the directives. Return the result of any special action requests in the callback function when the
15 operation is completed. Actions may include a request (**PMIX_GROUP_ASSIGN_CONTEXT_ID**
16) that the host assign a unique numerical (size_t) ID to this group - if given, the **PMIX_RANGE**
17 attribute will specify the range across which the ID must be unique (default to
18 **PMIX_RANGE_SESSION**).

19 11.3.28 pmix_server_fabric_fn_t

20 Summary

21 Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

22 Format

PMIx v4.0

C

```
23 typedef pmix_status_t (*pmix_server_fabric_fn_t) (  
24     const pmix_proc_t *requestor,  
25     pmix_fabric_operation_t op,  
26     const pmix_info_t directives[],  
27     size_t ndirs,  
28     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:

- 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 • **PMIX_OPERATION_SUCCEEDED**, indicating that the request was immediately processed and
 19 returned *success* - the *cbfunc* will not be called
- 20 • **PMIX_ERR_NOT_SUPPORTED**, indicating that the host environment does not support the
 21 request, even though the function entry was provided in the server module - the *cbfunc* will not
 22 be called
- 23 • a PMIx error constant indicating either an error in the input or that the request was immediately
 24 processed and failed - the *cbfunc* will not be called

Required Attributes

25 The following directives are required to be supported by all hosts to aid users in identifying the
 26 fabric to whom the operation is to be applied:

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
2
3
4

Description

Perform the specified operation. Return the result of any requests in the callback function when the operation is completed. Operations may, for example, include a request for fabric information. See [pmix_fabric_t](#) for a list of expected information to be included in the response.

CHAPTER 12

Fabric Support Definitions

1 As the drive for performance continues, interest has grown in both scheduling algorithms that take
2 into account network locality of the allocated resources, and in optimizing collective
3 communication patterns by structuring them to follow fabric topology. Several interfaces have been
4 defined that are specifically intended to support WLMs (also known as *schedulers*) by providing
5 access to information of potential use to scheduling algorithms - e.g., information on
6 communication costs between different points on the fabric.

7 In contrast, hierarchical collective operations require each process have global information about
8 both its peers and the fabric. For example, one might aggregate the contribution from all processes
9 on a node, then again across all nodes on a common switch, and finally across all switches. Creating
10 such optimized patterns relies on detailed knowledge of the fabric location of each participant.

11 PMIx supports these efforts by defining datatypes and attributes by which fabric coordinates for
12 processes and devices can be obtained from the host SMS. When used in conjunction with the
13 PMIx *instant on* methods, this results in the ability of a process to obtain the fabric coordinate of all
14 other processes without incurring additional overhead associated with the publish/exchange of that
15 information.

12.1 Fabric Support Constants

17 The following constants are defined for use in fabric-related events.

18 **PMIX_FABRIC_UPDATE_PENDING** The PMIx server library has been alerted to a change in
19 the fabric that requires updating of one or more registered `pmix_fabric_t` objects.

20 **PMIX_FABRIC_UPDATED** The PMIx server library has completed updating the entries of all
21 affected `pmix_fabric_t` objects registered with the library. Access to the entries of those
22 objects may now resume.

23 **PMIX_FABRIC_COORDS_UPDATED** Fabric coordinates have been updated - the affected
24 fabrics/planes are identified in the notification. Coordinates of processes and devices on those
25 affected components should be refreshed prior to next use.

12.2 Fabric Support Datatypes

27 Several datatype definitions have been created to support fabric-related operations and information.

1 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;
```

C

C

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.

1 12.2.2 Fabric Coordinate Support Macros

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

3 12.2.2.1 Initialize the `pmix_coord_t` structure

4 Initialize the `pmix_coord_t` fields

PMIx v4.0  

5 **PMIX_COORD_CONSTRUCT** (m)

6 **IN** m

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

8 12.2.2.2 Destruct the `pmix_coord_t` structure

9 Destruct the `pmix_coord_t` fields

PMIx v4.0  

10 **PMIX_COORD_DESTRUCT** (m)

11 **IN** m

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

13 12.2.2.3 Create a `pmix_coord_t` array

14 Allocate and initialize a `pmix_coord_t` array

PMIx v4.0  

15 **PMIX_COORD_CREATE** (m, n)

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`)

20 12.2.2.4 Release a `pmix_coord_t` array

21 Release an array of `pmix_coord_t` structures

PMIx v4.0  

22 **PMIX_COORD_FREE** (m, n)

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`)

1 12.2.3 Fabric Coordinate Views

PMIx v4.0

```
typedef uint8_t pmix_coord_view_t;
#define PMIX_COORD_VIEW_UNDEF      0x00
#define PMIX_COORD_LOGICAL_VIEW    0x01
#define PMIX_COORD_PHYSICAL_VIEW   0x02
```

Fabric coordinates can be reported based on different *views* according to user preference at the time of request. The following views have been defined:

PMIX_COORD_VIEW_UNDEF The coordinate view has not been defined.

PMIX_COORD_LOGICAL_VIEW The coordinates are provided in a *logical* view, typically given in Cartesian (x,y,z) dimensions, that describes the data flow in the fabric as defined by the arrangement of the hierarchical addressing scheme, fabric segmentation, routing domains, and other similar factors employed by that fabric.

PMIX_COORD_PHYSICAL_VIEW The coordinates are provided in a *physical* view based on the actual wiring diagram of the fabric - i.e., values along each axis reflect the relative position of that interface on the specific fabric cabling.

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.

20 12.2.4 Fabric Link State

The `pmix_link_state_t` is a `uint32_t` type for fabric link states.

PMIx v4.0

```
typedef uint8_t pmix_link_state_t;
```

The following constants can be used to set a variable of the type `pmix_link_state_t`. All definitions were introduced in version 4 of the standard unless otherwise marked. Valid link state values start at zero.

PMIX_LINK_STATE_UNKNOWN The port state is unknown or not applicable.

PMIX_LINK_DOWN The port is inactive.

PMIX_LINK_UP The port is active.

1 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.

20 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.grps" (**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 **PMIX_FABRIC_DEVICE_SPEED** "pmix.fabdev.speed" (**size_t**)
 4 The active link data rate, given in bits per second.

5 **PMIX_FABRIC_DEVICE_STATE** "pmix.fabdev.state" (**pmix_link_state_t**)
 6 The last available physical port state. Possible values are **PMIX_LINK_STATE_UNKNOWN**,
 7 **PMIX_LINK_DOWN**, and **PMIX_LINK_UP**, to indicate if the port state is unknown or not
 8 applicable (unknown), inactive (down), or active (up).

9 **PMIX_FABRIC_DEVICE_TYPE** "pmix.fabdev.type" (**string**)
 10 Specifies the type of fabric interface currently active on the device, such as Ethernet or
 11 InfiniBand.

12 **PMIX_FABRIC_DEVICE_BUS_TYPE** "pmix.fabdev.btyp" (**string**)
 13 The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

14 **PMIX_FABRIC_DEVICE_PCI_DEVID** "pmix.fabdev.pcidevid" (**string**)
 15 A node-level unique identifier for a Peripheral Component Interconnect (PCI) device.
 16 Provided only if the device is located on a PCI bus. The identifier is constructed as a
 17 four-part tuple delimited by colons comprised of the PCI 16-bit domain, 8-bit bus, 8-bit
 18 device, and 8-bit function IDs, each expressed in zero-extended hexadecimal form. Thus, an
 19 example identifier might be "abc1:0f:23:01". The combination of node identifier (**PMIX_HOSTNAME**
 20 or **PMIX_NODEID**) and **PMIX_FABRIC_DEVICE_PCI_DEVID**
 21 shall be unique within the system.



22 12.3 Fabric Support Attributes

23 The following attributes are used by the library supporting the system's WLM to either access or
 24 return fabric-related information (e.g., as part of the **pmix_fabric_t** structure).

25 **PMIX_SERVER_SCHEDULER** "pmix.srv.sched" (**bool**)
 26 Server requests access to WLM-supporting features - passed solely to the
 27 **PMIx_server_init** API to indicate that the library is to be initialized for scheduler
 28 support.

29 **PMIX_FABRIC_COST_MATRIX** "pmix.fab.cm" (**pointer**)
 30 Pointer to a two-dimensional array of point-to-point relative communication costs expressed
 31 as **uint16_t** values

32 **PMIX_FABRIC_GROUPS** "pmix.fab.grps" (**string**)
 33 A string delineating the group membership of nodes in the system, where each fabric group
 34 consists of the group number followed by a colon and a comma-delimited list of nodes in
 35 that group, with the groups delimited by semi-colons (e.g.,
 36 0:node000,node002,node004,node006;1:node001,node003,node005,node007)

1 The following attributes may be returned by calls to the scheduler-related APIs or in response to
2 queries (e.g., `PMIx_Get` or `PMIx_Query_info`) made by processes or tools.

3 **PMIX_FABRIC_VENDOR** "`pmix.fab.vndr`" (`string`)

4 Name of fabric vendor (e.g., Amazon, Mellanox, Cray, Intel)

5 **PMIX_FABRIC_IDENTIFIER** "`pmix.fab.id`" (`string`)

6 An identifier for the fabric (e.g., MgmtEthernet, Slingshot-11, OmniPath-1)

7 **PMIX_FABRIC_INDEX** "`pmix.fab.idx`" (`size_t`)

8 The index of the fabric as returned in `pmix_fabric_t`

9 **PMIX_FABRIC_NUM_VERTICES** "`pmix.fab.nverts`" (`size_t`)

10 Total number of NICs in the system - corresponds to the number of vertices (i.e., rows and
11 columns) in the cost matrix

12 **PMIX_FABRIC_COORDINATE** "`pmix.fab.coord`" (`pmix_data_array_t`)

13 Fabric coordinate(s) of the specified process in the view and/or plane provided by the
14 requester. If only one NIC has been assigned to the specified process, then the array will
15 contain only one address. Otherwise, the array will contain the coordinates of all NICs
16 available to the process in order of least to greatest distance from the process (NICs equally
17 distant from the process will be listed in arbitrary order).

18 **PMIX_FABRIC_VIEW** "`pmix.fab.view`" (`pmix_coord_view_t`)

19 Fabric coordinate view to be used for the requested coordinate - see
20 `pmix_coord_view_t` for the list of accepted values.

21 **PMIX_FABRIC_DIMS** "`pmix.fab.dims`" (`uint32_t`)

22 Number of dimensions in the specified fabric plane/view. If no plane is specified in a
23 request, then the dimensions of all planes in the system will be returned as a
24 `pmix_data_array_t` containing an array of `uint32_t` values. Default is to provide
25 dimensions in *logical* view.

26 **PMIX_FABRIC_PLANE** "`pmix.fab.plane`" (`char*`)

27 ID string of a fabric plane (e.g., CIDR for Ethernet). When used as a modifier in a request
28 for information, specifies the plane whose information is to be returned. When used directly
29 in a request, returns a `pmix_data_array_t` of string identifiers for all fabric planes in
30 the system.

31 **PMIX_FABRIC_ENDPT** "`pmix.fab.endpt`" (`pmix_data_array_t`)

32 Fabric endpoints for a specified process. As multiple endpoints may be assigned to a given
33 process (e.g., in the case where multiple NICs are associated with a package to which the
34 process is bound), the returned values will be provided in a `pmix_data_array_t` - the
35 returned data type of the individual values in the array varies by fabric provider.

36 **PMIX_FABRIC_SHAPE** "`pmix.fab.shape`" (`pmix_data_array_t*`)

37 The size of each dimension in the specified fabric plane/view, returned in a
38 `pmix_data_array_t` containing an array of `uint32_t` values. The size is defined as
39 the number of elements present in that dimension - e.g., the number of NICs in one
40 dimension of a physical view of a fabric plane. If no plane is specified, then the shape of
41 each plane in the system will be returned in an array of fabric shapes. Default is to provide
42 the shape in *logical* view.

43 **PMIX_FABRIC_SHAPE_STRING** "`pmix.fab.shapestr`" (`string`)

1 Network shape expressed as a string (e.g., "10x12x2").

2 The following attributes are used to describe devices (a.k.a., NICs) attached to the fabric.

3 **PMIX_FABRIC_DEVICE** "pmix.fabdev" (**pmix_data_array_t**)

4 An array of **pmix_info_t** describing a particular fabric device (NIC).

5 **PMIX_FABRIC_DEVICE_INDEX** "pmix.fabdev.idx" (**uint32_t**)

6 System-unique index of a particular fabric device (NIC).

7 **PMIX_FABRIC_DEVICE_NAME** "pmix.fabdev.nm" (**string**)

8 The operating system name associated with the device. This may be a logical fabric interface
9 name (e.g. eth0 or eno1) or an absolute filename.

10 **PMIX_FABRIC_DEVICE_VENDOR** "pmix.fabdev.vndr" (**string**)

11 Indicates the name of the vendor that distributes the NIC.

12 **PMIX_FABRIC_DEVICE_BUS_TYPE** "pmix.fabdev.btyp" (**string**)

13 The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").

14 **PMIX_FABRIC_DEVICE_ID** "pmix.fabdev.devid" (**string**)

15 This is a vendor-provided identifier for the device or product.

16 **PMIX_FABRIC_DEVICE_DRIVER** "pmix.fabdev.driver" (**string**)

17 The name of the driver associated with the device

18 **PMIX_FABRIC_DEVICE_FIRMWARE** "pmix.fabdev.fmwr" (**string**)

19 The device's firmware version

20 **PMIX_FABRIC_DEVICE_ADDRESS** "pmix.fabdev.addr" (**string**)

21 The primary link-level address associated with the NIC, such as a MAC address. If multiple
22 addresses are available, only one will be reported.

23 **PMIX_FABRIC_DEVICE_MTU** "pmix.fabdev.mtu" (**size_t**)

24 The maximum transfer unit of link level frames or packets, in bytes.

25 **PMIX_FABRIC_DEVICE_SPEED** "pmix.fabdev.speed" (**size_t**)

26 The active link data rate, given in bits per second.

27 **PMIX_FABRIC_DEVICE_STATE** "pmix.fabdev.state" (**pmix_link_state_t**)

28
29 The last available physical port state. Possible values are **PMIX_LINK_STATE_UNKNOWN** ,
30 **PMIX_LINK_DOWN** , and **PMIX_LINK_UP** , to indicate if the port state is unknown or not
31 applicable (unknown), inactive (down), or active (up).

32 **PMIX_FABRIC_DEVICE_TYPE** "pmix.fabdev.type" (**string**)

33 Specifies the type of fabric interface currently active on the device, such as Ethernet or
34 InfiniBand.

35 **PMIX_FABRIC_DEVICE_PCI_DEVID** "pmix.fabdev.pcidevid" (**string**)

36 A node-level unique identifier for a PCI device. Provided only if the device is located on a
37 PCI bus. The identifier is constructed as a four-part tuple delimited by colons comprised of
38 the PCI 16-bit domain, 8-bit bus, 8-bit device, and 8-bit function IDs, each expressed in
39 zero-extended hexadecimal form. Thus, an example identifier might be "abc1:0f:23:01". The
40 combination of node identifier (**PMIX_HOSTNAME** or **PMIX_NODEID**) and
41 **PMIX_FABRIC_DEVICE_PCI_DEVID** shall be unique within the system.

1 12.4 Fabric Support Functions

2 The following APIs allow the WLM to request specific services from the fabric subsystem via the
3 PMIx library.

Advice to PMIx server hosts

4 Due to their high cost in terms of execution, memory consumption, and interactions with other
5 SMS components (e.g., a fabric manager), it is strongly advised that the underlying implementation
6 of these APIs be restricted to a single PMIx server in a system that is supporting the SMS
7 component responsible for the scheduling of allocations (i.e., the system `scheduler`). The
8 `PMIX_SERVER_SCHEDULER` attribute can be used for this purpose to control the execution path.
9 Clients, tools, and other servers utilizing these functions are advised to have their requests
10 forwarded to the server supporting the scheduler using the `pmix_server_fabric_fn_t`
11 server module function, as needed.

12 12.4.1 PMIx_Fabric_register

13 Summary

14 Register for access to fabric-related information.

15 Format

PMIx v4.0

C

```
16 pmix_status_t  
17 PMIx_Fabric_register(pmix_fabric_t *fabric,  
18                     const pmix_info_t directives[],  
19                     size_t ndirs)
```

C

20 IN fabric

21 address of a `pmix_fabric_t` (backed by storage). User may populate the "name" field at
22 will - PMIx does not utilize this field (handle)

23 IN directives

24 an optional array of values indicating desired behaviors and/or fabric to be accessed. If `NULL`,
25 then the highest priority available fabric will be used (array of handles)

26 IN ndirs

27 Number of elements in the *directives* array (integer)

28 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)

Description

Register for access to fabric-related information, including the communication cost matrix. This call must be made prior to requesting information from a fabric. The caller may request access to a particular fabric using the vendor, type, or identifier, or to a specific `fabric plane` via the `PMIX_FABRIC_PLANE` attribute - otherwise, the default fabric will be returned.

For performance reasons, the PMIx library does not provide thread protection for accessing the information in the `pmix_fabric_t` structure. Instead, the PMIx implementation shall provide two methods for coordinating updates to the provided fabric information:

- Users may periodically poll for updates using the `PMIx_Fabric_update` API
- Users may register for `PMIX_FABRIC_UPDATE_PENDING` events indicating that an update to the cost matrix is pending. When received, users are required to terminate or pause any actions involving access to the cost matrix before returning from the event. Completion of the `PMIX_FABRIC_UPDATE_PENDING` event handler indicates to the PMIx library that the fabric object's entries are available for updating. This may include releasing and re-allocating memory as the number of vertices may have changed (e.g., due to addition or removal of one or more NICs). When the update has been completed, the PMIx library will generate a `PMIX_FABRIC_UPDATED` event indicating that it is safe to begin using the updated fabric object(s).

There is no requirement that the caller exclusively use either one of these options. For example, the user may choose to both register for fabric update events, but poll for an update prior to some critical operation.

1 12.4.2 PMIx_Fabric_update

2 Summary

3 Update fabric-related information.

4 Format

PMIx v4.0

```
▼ _____ C _____ ▼  
pmix_status_t  
PMIx_Fabric_update(pmix_fabric_t *fabric)  
▲ _____ C _____ ▲
```

7 **IN** fabric

8 address of a `pmix_fabric_t` (backed by storage) (handle)

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

10 Description

11 Update fabric-related information. This call can be made at any time to request an update of the
12 fabric information contained in the provided `pmix_fabric_t` object. The caller is not allowed
13 to access the provided `pmix_fabric_t` until the call has returned.

14 12.4.3 PMIx_Fabric_deregister

15 Summary

16 Deregister a fabric object.

17 Format

PMIx v4.0

```
▼ _____ C _____ ▼  
pmix_status_t PMIx_Fabric_deregister(pmix_fabric_t *fabric)  
▲ _____ C _____ ▲
```

19 **IN** input

20 address of a `pmix_fabric_t` (handle)

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

22 Description

23 Deregister a fabric object, providing an opportunity for the PMIx library to cleanup any information
24 (e.g., cost matrix) associated with it. Contents of the provided `pmix_fabric_t` will be
25 invalidated upon function return.

26 12.4.4 PMIx_Fabric_get_vertex_info

27 Summary

28 Given a communication cost matrix index for a specified fabric, return the corresponding vertex
29 info.

1
PMIx v4.0

Format

C

```
2 pmix_status_t
3 PMIx_Fabric_get_vertex_info(pmix_fabric_t *fabric, uint32_t index,
4                             pmix_info_t **info, size_t *ninfo)
```

C

- 5 **IN fabric**
address of a `pmix_fabric_t` (handle)
- 7 **IN index**
vertex index (i.e., communication cost matrix row or column number) (integer)
- 9 **INOUT info**
Address where a pointer to an array of `pmix_info_t` containing the results of the query can be returned (memory reference)
- 12 **INOUT ninfo**
Address where the number of elements in `info` can be returned (handle)

14 Returns one of the following:

- 15 • **PMIX_SUCCESS** , indicating return of a valid value.
- 16 • **PMIX_ERR_BAD_PARAM** , indicating that the provided index is out of bounds.
- 17 • a PMIx error constant indicating either an error in the input or that the request failed.

Description

19 Query information about a specified vertex (fabric device, or NIC) in the system. The returned *status* indicates if requested data was found or not. The returned array of `pmix_info_t` will contain information on the specified vertex - the exact contents will depend on the PMIx implementation and the fabric vendor. At a minimum, it must contain sufficient information to uniquely identify the device within the system (ordering is arbitrary):

Required Attributes

- 24 **PMIX_HOSTNAME** "pmix.hname" (**char***)
25 Name of the host (e.g., where a specified process is running, or a given device is located).
26 The **PMIX_NODEID** may be returned in its place, or in addition to the hostname.
- 27 **PMIX_FABRIC_DEVICE_NAME** "pmix.fabdev.nm" (**string**)
28 The operating system name associated with the device. This may be a logical fabric interface
29 name (e.g. eth0 or eno1) or an absolute filename.
- 30 **PMIX_FABRIC_DEVICE_VENDOR** "pmix.fabdev.vndr" (**string**)
31 Indicates the name of the vendor that distributes the NIC.
- 32 **PMIX_FABRIC_DEVICE_BUS_TYPE** "pmix.fabdev.btyp" (**string**)
33 The type of bus to which the device is attached (e.g., "PCI", "GEN-Z").
- 34 **PMIX_FABRIC_DEVICE_PCI_DEVID** "pmix.fabdev.pcidevid" (**string**)

1 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_DEVICE](#) 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.

1 12.4.5 PMIx_Fabric_get_index

2 Summary

3 Given vertex info, return the corresponding communication cost matrix index.

4 Format

PMIx v4.0

C

5 `pmix_status_t`

```
6 PMIx_Fabric_get_index(pmix_fabric_t *fabric,  
7                       const pmix_info_t vertex[], size_t ninfo,  
8                       uint32_t *index)
```

C

9 **IN fabric**

10 address of a `pmix_fabric_t` (handle)

11 **IN vertex**

12 array of `pmix_info_t` containing info describing the vertex whose index is being queried
13 (handle)

14 **IN ninfo**

15 number of elements in *vertex*

16 **OUT index**

17 pointer to the location where the index is to be returned (memory reference (handle))

18 Returns one of the following:

- 19 • **PMIX_SUCCESS** , indicating return of a valid value.
- 20 • a PMIx error constant indicating either an error in the input or that the request failed.

21 Description

22 Query the index number of a vertex corresponding to the provided description. The description
23 must provide adequate information to uniquely identify the target vertex. At a minimum, this must
24 include identification of the node hosting the device using either the **PMIX_HOSTNAME** or
25 **PMIX_NODEID** , plus a node-level unique identifier for the device (e.g., the
26 **PMIX_FABRIC_DEVICE_PCI_DEVID** for a PCI device).

CHAPTER 13

Process Sets and Groups

PMIx supports two slightly related, but functionally different concepts known as *process sets* and *process groups*. This chapter these two concepts and describes how they are utilized, along with their corresponding APIs.

13.1 Process Sets

A PMIx *Process Set* is a user-provided label associated with a given set of application processes. Definition of a PMIx process set typically occurs at time of application execution - e.g., on a PR RTE command line:

```
$ prun -n 4 --pset ocean myoceanapp : -n 3 --pset ice myiceapp
```

In this example, the processes in the first application will be labeled with a **PMIX_PSET_NAME** attribute of *ocean* while those in the second application will be labeled with an *ice* value. During the execution, application processes could lookup the process set attribute for any other process using **PMIx_Get**. Alternatively, other executing applications could utilize the **PMIx_Query_info_nb** API to obtain the number of declared process sets in the system, a list of their names, and other information about them. In other words, the *process set* identifier provides a label by which an application can derive information about a process and its application - it does *not*, however, confer any operational function.

Thus, *process sets* differ from *process groups* in several key ways:

- *Process sets* have no implied relationship between their members - i.e., a process in a process set has no concept of a “pset rank” as it would in a process *group*
- Processes can only have one process *set* identifier, but can simultaneously belong to multiple process *groups*
- Process *set* identifiers are considered job-level information set at launch. No PMIx API is provided by which a user can change the process *set* value of a process on-the-fly. In contrast, PMIx process *groups* can only be defined dynamically by the application.

- Process *groups* can be used in calls to PMIx operations. Members of process *groups* that are involved in an operation are translated by their PMIx server into their *native* identifier prior to the operation being passed to the host environment. For example, an application can define a process group to consist of ranks 0 and 1 from the host-assigned namespace of 210456, identified by the group id of *foo*. If the application subsequently calls the `PMIx_Fence` API with a process identifier of {foo, PMIX_RANK_WILDCARD}, the PMIx server will replace that identifier with an array consisting of {210456, 0} and {210456, 1} - the host-assigned identifiers of the participating processes - prior to passing the request up to the host environment
- Process *groups* can request that the host environment assign a unique `size_t` Process Group Context Identifier (PGCID) to the group at time of group construction. An MPI library may, for example, use the PGCID as the MPI communicator identifier for the group.

The two concepts do, however, overlap in one specific area. Process *groups* are included in the process *set* information returned by calls to `PMIx_Query_info_nb`. Thus, a *process group* can effectively be considered an extended version of a *process set* that adds dynamic definition and operational context to the *process set* concept.

Advice to PMIx library implementers

PMIx implementations are required to include all active *group* identifiers in the returned list of process *set* names provided in response to the appropriate `PMIx_Query_info_nb` call.

13.2 Process Groups

PMIx *Groups* are defined as a collection of processes desiring a common, unique identifier for purposes such as passing events or participating in PMIx fence operations. As with processes that assemble via `PMIx_Connect`, each member of the group is provided with both the job-level information of any other namespace represented in the group, and the contact information for all group members. However, *groups* differ from `PMIx_Connect` assemblages in the following key areas:

- Relation to the host environment
 - Calls to `PMIx_Connect` are relayed to the host environment. This means that the host RM should treat the failure of any process in the specified assemblage as a reportable event and take appropriate action. However, the environment is not required to define a new identifier for the connected assemblage or any of its member processes, nor does it define a new rank for each process within that assemblage. In addition, the PMIx server does not provide any tracking support for the assemblage. Thus, the caller is responsible for addressing members of the connected assemblage using their RM-provided identifiers.

- 1 – Calls to PMIx Group APIs are first processed within the local PMIx server. When constructed,
2 the server creates a tracker that associates the specified processes with the user-provided group
3 identifier, and assigns a new *group rank* based on their relative position in the array of
4 processes provided in the call to `PMIx_Group_construct`. Members of the group can
5 subsequently utilize the group identifier in PMIx function calls to address the group’s
6 members, using either `PMIX_RANK_WILDCARD` to refer to all of them or the group-level
7 rank of specific members. The PMIx server will translate the specified processes into their
8 RM-assigned identifiers prior to passing the request up to its host. Thus, the host environment
9 has no visibility into the group’s existence or membership.

Advice to users

10 User-provided group identifiers must be distinct from anything provided by the RM so as to
11 avoid collisions between group identifiers and RM-assigned namespaces. This can usually be
12 accomplished through the use of an application-specific prefix – e.g., “myapp-foo”

• Construction procedure

- 13
- 14 – `PMIx_Connect` calls require that every process call the API before completing – i.e., it is
15 modeled upon the bulk synchronous traditional MPI connect/accept methodology. Thus, a
16 given application thread can only be involved in one connect/accept operation at a time, and is
17 blocked in that operation until all specified processes participate. In addition, there is no
18 provision for replacing processes in the assemblage due to failure to participate, nor a
19 mechanism by which a process might decline participation.
- 20 – PMIx Groups are designed to be more flexible in their construction procedure by relaxing
21 these constraints. While a standard blocking form of constructing groups is provided, the event
22 notification system is utilized to provide a designated *group leader* with the ability to replace
23 participants that fail to participate within a given timeout period. This provides a mechanism
24 by which the application can, if desired, replace members on-the-fly or allow the group to
25 proceed with partial membership. In such cases, the final group membership is returned to all
26 participants upon completion of the operation.

27 Additionally, PMIx supports dynamic definition of group membership based on an invite/join
28 model. A process can asynchronously initiate construction of a group of any processes via the
29 `PMIx_Group_invite` function call. Invitations are delivered via a PMIx event (using the
30 `PMIX_GROUP_INVITED` event) to the invited processes which can then either accept or
31 decline the invitation using the `PMIx_Group_join` API. The initiating process tracks
32 responses by registering for the events generated by the call to `PMIx_Group_join`,
33 timeouts, or process terminations, optionally replacing processes that decline the invitation,
34 fail to respond in time, or terminate without responding. Upon completion of the operation,
35 the final list of participants is communicated to each member of the new group.

• Destruct procedure

- 1 – Processes that assemble via **PMIx_Connect** must all depart the assemblage together – i.e.,
- 2 no member can depart the assemblage while leaving the remaining members in it. Even the
- 3 non-blocking form of **PMIx_Disconnect** retains this requirement in that members remain
- 4 a part of the assemblage until all members have called **PMIx_Disconnect_nb**
- 5 – Members of a PMIx Group may depart the group at any time via the **PMIx_Group_leave**
- 6 API. Other members are notified of the departure via the **PMIX_GROUP_LEFT** event to
- 7 distinguish such events from those reporting process termination. This leaves the remaining
- 8 members free to continue group operations. The **PMIx_Group_destruct** operation offers
- 9 a collective method akin to **PMIx_Disconnect** for deconstructing the entire group.

10 Note that applications supporting dynamic group behaviors such as asynchronous departure

11 take responsibility for ensuring global consistency in the group definition prior to executing

12 group collective operations - i.e., it is the application's responsibility to either ensure that

13 knowledge of the current group membership is globally consistent across the participants, or to

14 register for appropriate events to deal with the lack of consistency during the operation.

15 In other words, members of PMIx Groups are *loosely coupled* as opposed to *tightly connected*

16 when constructed via **PMIx_Connect** . The relevant APIs are explained below.

Advice to users

17 The reliance on PMIx events in the PMIx Group concept dictates that processes utilizing these APIs

18 must register for the corresponding events. Failure to do so will likely lead to operational failures.

19 Users are recommended to utilize the **PMIX_TIMEOUT** directive (or retain an internal timer) on

20 calls to PMIx Group APIs (especially the blocking form of those functions) as processes that have

21 not registered for required events will never respond.

22 13.2.1 Group Operation Constants

23 *PMIx v4.0* The **pmix_group_operation_t** structure is an enumerated type for specifying group

24 operations. All values were originally defined in version 4 of the standard unless otherwise marked.

- 25 **PMIX_GROUP_DECLINE** Decline an invitation to join a PMIx group - provided for readability
- 26 of user code
- 27 **PMIX_GROUP_ACCEPT** Accept an invitation to join a PMIx group - provided for readability
- 28 of user code
- 29 **PMIX_GROUP_CONSTRUCT** Construct a group composed of the specified processes - used by
- 30 a PMIx server library to direct host operation
- 31 **PMIX_GROUP_DESTRUCT** Destruct the specified group - used by a PMIx server library to
- 32 direct host operation

1 13.2.2 PMIx_Group_construct

2 Summary

3 Construct a PMIx process group

4 Format

PMIx v4.0

C

5 `pmix_status_t`

```
6 PMIx_Group_construct(const char grp[],  
7                   const pmix_proc_t procs[], size_t nprocs,  
8                   const pmix_info_t directives[], size_t ndirs,  
9                   pmix_info_t **results, size_t *nresults)
```

C

- 10 **IN** `grp`
11 `NULL`-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the
12 group identifier (string)
- 13 **IN** `procs`
14 Array of `pmix_proc_t` structures containing the PMIx identifiers of the member processes
15 (array of handles)
- 16 **IN** `nprocs`
17 Number of elements in the `procs` array (`size_t`)
- 18 **IN** `directives`
19 Array of `pmix_info_t` structures (array of handles)
- 20 **IN** `ndirs`
21 Number of elements in the `directives` array (`size_t`)
- 22 **INOUT** `results`
23 Pointer to a location where the array of `pmix_info_t` describing the results of the
24 operation is to be returned (pointer to handle)
- 25 **INOUT** `nresults`
26 Pointer to a `size_t` location where the number of elements in `results` is to be returned
27 (memory reference)
- 28 Returns one of the following:
- 29 • `PMIX_SUCCESS` , indicating that the request has been successfully completed
 - 30 • `PMIX_ERR_NOT_SUPPORTED` The PMIx library and/or the host RM does not support this
31 operation
 - 32 • a PMIx error constant indicating either an error in the input or that the request failed to be
33 completed

Required Attributes

The following attributes are *required* to be supported by all PMIx libraries that support this operation:

PMIX_GROUP_LEADER "pmix.grp.ldr" (bool)

This process is the leader of the group

PMIX_GROUP_OPTIONAL "pmix.grp.opt" (bool)

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

PMIX_GROUP_LOCAL_ONLY "pmix.grp.lcl" (bool)

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

Host environments that support this operation are *required* to provide the following attributes:

PMIX_GROUP_ASSIGN_CONTEXT_ID "pmix.grp.actxid" (bool)

Requests that the RM assign a new context identifier to the newly created group. The identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range specified in the request. Thus, the value serves as a means of identifying the group within that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

PMIX_GROUP_NOTIFY_TERMINATION "pmix.grp.notterm" (bool)

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

Construct a new group composed of the specified processes and identified with the provided group identifier. The group identifier is a user-defined, `NULL`-terminated character array of length less than or equal to `PMIX_MAX_NSLEN`. Only characters accepted by standard string comparison functions (e.g., `strncmp`) are supported. Processes may engage in multiple simultaneous group construct operations so long as each is provided with a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

If the `PMIX_GROUP_NOTIFY_TERMINATION` attribute is provided and has a value of `true`, then either the construct leader (if `PMIX_GROUP_LEADER` is provided) or all participants who register for the `PMIX_GROUP_MEMBER_FAILED` event will receive events whenever a process fails or terminates prior to calling `PMIx_Group_construct` – i.e. if a *group leader* is declared, *only* that process will receive the event. In the absence of a declared leader, *all* specified group members will receive the event.

The event will contain the identifier of the process that failed to join plus any other information that the host RM provided. This provides an opportunity for the leader or the collective members to react to the event – e.g., to decide to proceed with a smaller group or to abort the operation. The decision is communicated to the PMIx library in the results array at the end of the event handler. This allows PMIx to properly adjust accounting for procedure completion. When construct is complete, the participating PMIx servers will be alerted to any change in participants and each group member will receive an updated group membership (marked with the `PMIX_GROUP_MEMBERSHIP` attribute) as part of the *results* array returned by this API.

Failure of the declared leader at any time will cause a `PMIX_GROUP_LEADER_FAILED` event to be delivered to all participants so they can optionally declare a new leader. A new leader is identified by providing the `PMIX_GROUP_LEADER` attribute in the results array in the return of the event handler. Only one process is allowed to return that attribute, thereby declaring itself as the new leader. Results of the leader selection will be communicated to all participants via a `PMIX_GROUP_LEADER_SELECTED` event identifying the new leader. If no leader was selected, then the `pmix_info_t` provided to that event handler will include that information so the participants can take appropriate action.

Any participant that returns `PMIX_GROUP_CONSTRUCT_ABORT` from either the `PMIX_GROUP_MEMBER_FAILED` or the `PMIX_GROUP_LEADER_FAILED` event handler will

1 cause the construct process to abort, returning from the call with a
2 **PMIX_GROUP_CONSTRUCT_ABORT** status.

3 If the **PMIX_GROUP_NOTIFY_TERMINATION** attribute is not provided or has a value of
4 **false**, then the **PMIx_Group_construct** operation will simply return an error whenever a
5 proposed group member fails or terminates prior to calling **PMIx_Group_construct** .

6 Providing the **PMIX_GROUP_OPTIONAL** attribute with a value of **true** directs the PMIx library
7 to consider participation by any specified group member as non-required - thus, the operation will
8 return **PMIX_SUCCESS** if all members participate, or **PMIX_ERR_PARTIAL_SUCCESS** if
9 some members fail to participate. The *results* array will contain the final group membership in the
10 latter case. Note that this use-case can cause the operation to hang if the **PMIX_TIMEOUT**
11 attribute is not specified and one or more group members fail to call **PMIx_Group_construct**
12 while continuing to execute. Also, note that no leader or member failed events will be generated
13 during the operation.

14 Processes in a group under construction are not allowed to leave the group until group construction
15 is complete. Upon completion of the construct procedure, each group member will have access to
16 the job-level information of all namespaces represented in the group plus any information posted
17 via **PMIx_Put** (subject to the usual scoping directives) for every group member.

Advice to PMIx library implementers

18 At the conclusion of the construct operation, the PMIx library is *required* to ensure that job-related
19 information from each participating namespace plus any information posted by group members via
20 **PMIx_Put** (subject to scoping directives) is available to each member via calls to **PMIx_Get** .

Advice to PMIx server hosts

21 The collective nature of this API generally results in use of a fence-like operation by the backend
22 host environment. Host environments that utilize the array of process participants as a *signature* for
23 such operations may experience potential conflicts should both a **PMIx_Group_construct**
24 and a **PMIx_Fence** operation involving the same participants be simultaneously executed. As
25 PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve
26 any potential conflicts.

27 13.2.3 PMIx_Group_construct_nb

28 Summary

29 Non-blocking form of **PMIx_Group_construct**

1
PMIx v4.0

Format

C

```
2 pmix_status_t
3 PMIx_Group_construct_nb(const char grp[],
4                        const pmix_proc_t procs[], size_t nprocs,
5                        const pmix_info_t directives[], size_t ndirs,
6                        pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

- 7 **IN** `grp`
8 NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the
9 group identifier (string)
- 10 **IN** `procs`
11 Array of `pmix_proc_t` structures containing the PMIx identifiers of the member processes
12 (array of handles)
- 13 **IN** `nprocs`
14 Number of elements in the `procs` array (`size_t`)
- 15 **IN** `directives`
16 Array of `pmix_info_t` structures (array of handles)
- 17 **IN** `ndirs`
18 Number of elements in the `directives` array (`size_t`)
- 19 **IN** `cbfunc`
20 Callback function `pmix_info_cbfunc_t` (function reference)
- 21 **IN** `cbdata`
22 Data to be passed to the callback function (memory reference)

23 Returns one of the following:

- 24 • `PMIX_SUCCESS` indicating that the request has been accepted for processing and the provided
25 callback function will be executed upon completion of the operation. Note that the library *must*
26 *not* invoke the callback function prior to returning from the API.
- 27 • `PMIX_OPERATION_SUCCEEDED`, indicating that the request was immediately processed and
28 returned *success* - the `cbfunc` will *not* be called
- 29 • `PMIX_ERR_NOT_SUPPORTED` The PMIx library does not support this operation - the `cbfunc`
30 will *not* be called
- 31 • a non-zero PMIx error constant indicating a reason for the request to have been rejected - the
32 `cbfunc` will *not* be called

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

- 35 • `PMIX_SUCCESS` The operation succeeded and all specified members participated.

- 1 • **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members
2 participated - the final group membership is included in the callback function
- 3 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
4 does not.
- 5 • a non-zero PMIx error constant indicating a reason for the request's failure

----- Required Attributes -----

6 PMIx libraries that choose not to support this operation *must* return
7 **PMIX_ERR_NOT_SUPPORTED** when the function is called.

8 The following attributes are *required* to be supported by all PMIx libraries that support this
9 operation:

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

11 This process is the leader of the group

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

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

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

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

23 Host environments that support this operation are *required* to provide the following attributes:

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

25 Requests that the RM assign a new context identifier to the newly created group. The
26 identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
27 specified in the request. Thus, the value serves as a means of identifying the group within
28 that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION**.

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

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

----- ▲

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

Non-blocking version of the **PMIx_Group_construct** operation. The callback function will be called once all group members have called either **PMIx_Group_construct** or **PMIx_Group_construct_nb**.

13.2.4 PMIx_Group_destruct

Summary

Destruct a PMIx process group

1
PMIx v4.0

Format

C

```
2 pmix_status_t
3 PMIx_Group_destruct(const char grp[],
4                     const pmix_info_t directives[], size_t ndirs)
```

C

- 5 **IN grp**
- 6 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
- 7 identifier of the group to be destructed (string)
- 8 **IN directives**
- 9 Array of **pmix_info_t** structures (array of handles)
- 10 **IN ndirs**
- 11 Number of elements in the *directives* array (**size_t**)

12 Returns one of the following:

- 13 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 14 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
- 15 operation
- 16 • a PMIx error constant indicating either an error in the input or that the request failed to be
- 17 completed

Required Attributes

18 For implementations and host environments that support the operation, there are no identified
19 required attributes for this API.

Optional Attributes

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

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

Advice to PMIx library implementers

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

Description

Destruct a group identified by the provided group identifier. Processes may engage in multiple simultaneous group destruct operations so long as each involves a unique group ID. The *directives* array can be used to pass user-level directives regarding timeout constraints and other options available from the PMIx server.

The destruct API will return an error if any group process fails or terminates prior to calling `PMIx_Group_destruct` or its non-blocking version unless the `PMIX_GROUP_NOTIFY_TERMINATION` attribute was provided (with a value of `false`) at time of group construction. If notification was requested, then the `PMIX_GROUP_MEMBER_FAILED` event will be delivered for each process that fails to call destruct and the destruct tracker updated to account for the lack of participation. The `PMIx_Group_destruct` operation will subsequently return `PMIX_SUCCESS` when the remaining processes have all called destruct – i.e., the event will serve in place of return of an error.

Advice to PMIx server hosts

The collective nature of this API generally results in use of a fence-like operation by the backend host environment. Host environments that utilize the array of process participants as a *signature* for such operations may experience potential conflicts should both a `PMIx_Group_destruct` and a `PMIx_Fence` operation involving the same participants be simultaneously executed. As PMIx allows for such use-cases, it is therefore the responsibility of the host environment to resolve any potential conflicts.

13.2.5 `PMIx_Group_destruct_nb`

Summary

Non-blocking form of `PMIx_Group_destruct`

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_destruct_nb(const char grp[],
4                       const pmix_info_t directives[], size_t ndirs,
5                       pmix_op_cbfunc_t cbfunc, void *cbdata)

```

C

- 6 **IN grp**
- 7 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
- 8 identifier of the group to be destructed (string)
- 9 **IN directives**
- 10 Array of **pmix_info_t** structures (array of handles)
- 11 **IN ndirs**
- 12 Number of elements in the *directives* array (**size_t**)
- 13 **IN cbfunc**
- 14 Callback function **pmix_op_cbfunc_t** (function reference)
- 15 **IN cbdata**
- 16 Data to be passed to the callback function (memory reference)

17 Returns one of the following:

- 18 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
- 19 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
- 20 from the API.
- 21 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
- 22 returned *success* - the *cbfunc* will *not* be called
- 23 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
- 24 will *not* be called
- 25 • a PMIx error constant indicating either an error in the input or that the request was immediately
- 26 processed and failed - the *cbfunc* will *not* be called

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

28 constants:

- 29 • **PMIX_SUCCESS** The operation was successfully completed
- 30 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
- 31 does not.
- 32 • a non-zero PMIx error constant indicating a reason for the request’s failure

Required Attributes

PMIx libraries that choose not to support this operation *must* return `PMIX_ERR_NOT_SUPPORTED` when the function is called. For implementations and host environments that support the operation, there are no identified required attributes for this API.

Optional Attributes

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

`PMIX_TIMEOUT` "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

Non-blocking version of the `PMIx_Group_destruct` operation. The callback function will be called once all members of the group have executed either `PMIx_Group_destruct` or `PMIx_Group_destruct_nb`.

13.2.6 PMIx_Group_invite

Summary

Asynchronously construct a PMIx process group

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_invite(const char grp[],
4                  const pmix_proc_t procs[], size_t nprocs,
5                  const pmix_info_t directives[], size_t ndirs,
6                  pmix_info_t **results, size_t *nresult)

```

C

- 7 **IN grp**
8 NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the
9 group identifier (string)
- 10 **IN procs**
11 Array of **pmix_proc_t** structures containing the PMIx identifiers of the processes to be
12 invited (array of handles)
- 13 **IN nprocs**
14 Number of elements in the *procs* array (**size_t**)
- 15 **IN directives**
16 Array of **pmix_info_t** structures (array of handles)
- 17 **IN ndirs**
18 Number of elements in the *directives* array (**size_t**)
- 19 **INOUT results**
20 Pointer to a location where the array of **pmix_info_t** describing the results of the
21 operation is to be returned (pointer to handle)
- 22 **INOUT nresults**
23 Pointer to a **size_t** location where the number of elements in *results* is to be returned
24 (memory reference)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this
28 operation
- 29 • a PMIx error constant indicating either an error in the input or that the request failed to be
30 completed

Required Attributes

31 The following attributes are *required* to be supported by all PMIx libraries that support this
32 operation:

- 33 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (**bool**)
34 Participation is optional - do not return an error if any of the specified processes terminate
35 without having joined. The default is false

1 Host environments that support this operation are *required* to provide the following attributes:

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

3 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.

Description

Explicitly invite the specified processes to join a group. The process making the `PMIx_Group_invite` call is automatically declared to be the *group leader*. Each invited process will be notified of the invitation via the `PMIX_GROUP_INVITED` event - the processes being invited must therefore register for the `PMIX_GROUP_INVITED` event in order to be notified of the invitation. Note that the PMIx event notification system caches events - thus, no ordering of invite versus event registration is required.

The invitation event will include the identity of the inviting process plus the name of the group. When ready to respond, each invited process provides a response using either the blocking or non-blocking form of `PMIx_Group_join`. This will notify the inviting process that the invitation was either accepted (via the `PMIX_GROUP_INVITE_ACCEPTED` event) or declined (via the `PMIX_GROUP_INVITE_DECLINED` event). The `PMIX_GROUP_INVITE_ACCEPTED` event is captured by the PMIx client library of the inviting process – i.e., the application itself does not need to register for this event. The library will track the number of accepting processes and alert the inviting process (by returning from the blocking form of `PMIx_Group_invite` or calling the callback function of the non-blocking form) when group construction completes.

The inviting process should, however, register for the `PMIX_GROUP_INVITE_DECLINED` if the application allows invited processes to decline the invitation. This provides an opportunity for the application to either invite a replacement, declare “abort”, or choose to remove the declining process from the final group. The inviting process should also register to receive `PMIX_GROUP_INVITE_FAILED` events whenever a process fails or terminates prior to responding to the invitation. Actions taken by the inviting process in response to these events must be communicated at the end of the event handler by returning the corresponding result so that the PMIx library can adjust accordingly.

Upon completion of the operation, all members of the new group will receive access to the job-level information of each other’s namespaces plus any information posted via `PMIx_Put` by the other members.

The inviting process is automatically considered the leader of the asynchronous group construction procedure and will receive all failure or termination events for invited members prior to completion. The inviting process is required to provide a `PMIX_GROUP_CONSTRUCT_COMPLETE` event once the group has been fully assembled – this event is used by the PMIx library as a trigger to release participants from their call to `PMIx_Group_join` and provides information (e.g., the final group membership) to be returned in the *results* array.

Advice to users

Applications are not allowed to use the group in any operations until group construction is complete. This is required in order to ensure consistent knowledge of group membership across all participants.

1 Failure of the inviting process at any time will cause a `PMIX_GROUP_LEADER_FAILED` event to
2 be delivered to all participants so they can optionally declare a new leader. A new leader is
3 identified by providing the `PMIX_GROUP_LEADER` attribute in the results array in the return of
4 the event handler. Only one process is allowed to return that attribute, declaring itself as the new
5 leader. Results of the leader selection will be communicated to all participants via a
6 `PMIX_GROUP_LEADER_SELECTED` event identifying the new leader. If no leader was selected,
7 then the status code provided in the event handler will provide an error value so the participants can
8 take appropriate action.

9 13.2.7 `PMIx_Group_invite_nb`

10 Summary

11 Non-blocking form of `PMIx_Group_invite`

12 Format

PMIx v4.0

C

```
13 pmix_status_t  
14 PMIx_Group_invite_nb(const char grp[],  
15                     const pmix_proc_t procs[], size_t nprocs,  
16                     const pmix_info_t directives[], size_t ndirs,  
17                     pmix_info_cbfunc_t cbfunc, void *cbdata)
```

C

- 18 **IN** `grp`
19 NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the
20 group identifier (string)
- 21 **IN** `procs`
22 Array of `pmix_proc_t` structures containing the PMIx identifiers of the processes to be
23 invited (array of handles)
- 24 **IN** `nprocs`
25 Number of elements in the `procs` array (`size_t`)
- 26 **IN** `directives`
27 Array of `pmix_info_t` structures (array of handles)
- 28 **IN** `ndirs`
29 Number of elements in the `directives` array (`size_t`)
- 30 **IN** `cbfunc`
31 Callback function `pmix_info_cbfunc_t` (function reference)
- 32 **IN** `cbdata`
33 Data to be passed to the callback function (memory reference)

34 Returns one of the following:

- 1 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
2 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
3 from the API.
- 4 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
5 returned *success* - the *cbfunc* will *not* be called
- 6 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
7 will *not* be called
- 8 • a PMIx error constant indicating either an error in the input or that the request was immediately
9 processed and failed - the *cbfunc* will *not* be called

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

- 12 • **PMIX_SUCCESS** The operation succeeded and all specified members participated.
- 13 • **PMIX_ERR_PARTIAL_SUCCESS** The operation succeeded but not all specified members
14 participated - the final group membership is included in the callback function
- 15 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
16 does not.
- 17 • a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

18 The following attributes are *required* to be supported by all PMIx libraries that support this
19 operation:

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

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

23 Host environments that support this operation are *required* to provide the following attributes:

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

25 Requests that the RM assign a new context identifier to the newly created group. The
26 identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
27 specified in the request. Thus, the value serves as a means of identifying the group within
28 that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

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

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

Optional Attributes

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

PMIX_TIMEOUT "pmix.timeout" (int)

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

Advice to PMIx library implementers

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

Description

Non-blocking version of the **PMIx_Group_invite** operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either **PMIx_Group_join** or **PMIx_Group_join_nb**.

13.2.8 PMIx_Group_join

Summary

Accept an invitation to join a PMIx process group

1
PMIx v4.0

Format

C

```

2 pmix_status_t
3 PMIx_Group_join(const char grp[],
4                 const pmix_proc_t *leader,
5                 pmix_group_operation_t opt,
6                 const pmix_info_t directives[], size_t ndirs,
7                 pmix_info_t **results, size_t *nresult)

```

C

- 8 **IN grp**
NULL-terminated character array of maximum size **PMIX_MAX_NSLEN** containing the group identifier (string)
- 10 **IN leader**
Process that generated the invitation (handle)
- 12 **IN opt**
Accept or decline flag (**pmix_group_operation_t**)
- 14 **IN directives**
Array of **pmix_info_t** structures (array of handles)
- 16 **IN ndirs**
Number of elements in the *directives* array (**size_t**)
- 18 **INOUT results**
Pointer to a location where the array of **pmix_info_t** describing the results of the operation is to be returned (pointer to handle)
- 20 **INOUT nresults**
Pointer to a **size_t** location where the number of elements in *results* is to be returned (memory reference)

25 Returns one of the following:

- 26 • **PMIX_SUCCESS** , indicating that the request has been successfully completed
- 27 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library and/or the host RM does not support this operation
- 28 • a PMIx error constant indicating either an error in the input or that the request failed to be completed

Required Attributes

31 There are no identified required attributes for implementers.

Optional Attributes

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 Respond to an invitation to join a group that is being asynchronously constructed. The process must
13 have registered for the **PMIX_GROUP_INVITED** event in order to be notified of the invitation.
14 When called, the event information will include the **pmix_proc_t** identifier of the process that
15 generated the invitation along with the identifier of the group being constructed. When ready to
16 respond, the process provides a response using either form of **PMIx_Group_join** .
17

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

PMIx v4.0

```
24 pmix_status_t  
25 PMIx_Group_join_nb(const char grp[],  
26                   const pmix_proc_t *leader,  
27                   pmix_group_operation_t opt,  
28                   const pmix_info_t directives[], size_t ndirs,  
29                   pmix_info_cbfunc_t cbfunc, void *cbdata)
```

30 IN `grp`

31 `NULL`-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the
32 group identifier (string)

33 IN `leader`

34 Process that generated the invitation (handle)

1 **IN** **opt**
2 Accept or decline flag (`pmix_group_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 • **PMIX_SUCCESS** , indicating that the request is being processed - result will be returned in the
13 provided *cbfunc*. Note that the library *must not* invoke the callback function prior to returning
14 from the API.
- 15 • **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and
16 returned *success* - the *cbfunc* will *not* be called
- 17 • **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc*
18 will *not* be called
- 19 • a PMIx error constant indicating either an error in the input or that the request was immediately
20 processed and failed - the *cbfunc* will *not* be called

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

- 23 • **PMIX_SUCCESS** The operation succeeded and group membership is in the callback function
24 parameters
- 25 • **PMIX_ERR_NOT_SUPPORTED** While the PMIx server supports this operation, the host RM
26 does not.
- 27 • a non-zero PMIx error constant indicating a reason for the request's failure

▼----- Required Attributes -----▼

28 There are no identified required attributes for implementers.

▲-----

▼----- Optional Attributes -----▼

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

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

Advice to PMIx library implementers

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

Description

Non-blocking version of the `PMIx_Group_join` operation. The callback function will be called once all invited members of the group (or their substitutes) have executed either `PMIx_Group_join` or `PMIx_Group_join_nb`.

13.2.10 PMIx_Group_leave

Summary

Leave a PMIx process group

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_leave(const char grp[],
                 const pmix_info_t directives[], size_t ndirs)
```

C

IN grp

NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

IN directives

Array of `pmix_info_t` structures (array of handles)

IN ndirs

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

Returns one of the following:

- `PMIX_SUCCESS`, indicating that the request has been communicated to the local PMIx server
- `PMIX_ERR_NOT_SUPPORTED` The PMIx library and/or the host RM does not support this operation
- a PMIx error constant indicating either an error in the input or that the request is unsupported

Required Attributes

There are no identified required attributes for implementers.

Description

Leave a PMIx Group. Calls to `PMIx_Group_leave` (or its non-blocking form) will cause a `PMIX_GROUP_LEFT` event to be generated notifying all members of the group of the caller's departure. The function will return (or the non-blocking function will execute the specified callback function) once the event has been locally generated and is not indicative of remote receipt.

Advice to users

The `PMIx_Group_leave` API is intended solely for asynchronous departures of individual processes from a group as it is not a scalable operation – i.e., when a process determines it should no longer be a part of a defined group, but the remainder of the group retains a valid reason to continue in existence. Developers are advised to use `PMIx_Group_destruct` (or its non-blocking form) for all other scenarios as it represents a more scalable operation.

13.2.11 `PMIx_Group_leave_nb`

Summary

Non-blocking form of `PMIx_Group_leave`

Format

PMIx v4.0

C

```
pmix_status_t
PMIx_Group_leave_nb(const char grp[],
                   const pmix_info_t directives[], size_t ndirs,
                   pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

IN `grp`

NULL-terminated character array of maximum size `PMIX_MAX_NSLEN` containing the group identifier (string)

IN `directives`

Array of `pmix_info_t` structures (array of handles)

IN `ndirs`

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

IN `cbfunc`

Callback function `pmix_op_cbfunc_t` (function reference)

IN `cbdata`

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

Returns one of the following:

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

- **PMIX_OPERATION_SUCCEEDED** , indicating that the request was immediately processed and returned *success* - the *cbfunc* will *not* be called
- **PMIX_ERR_NOT_SUPPORTED** The PMIx library does not support this operation - the *cbfunc* will *not* be called
- a PMIx error constant indicating either an error in the input or that the request was immediately processed and failed - the *cbfunc* will *not* be called

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

- **PMIX_SUCCESS** The operation succeeded - i.e., the **PMIX_GROUP_LEFT** event was generated
- **PMIX_ERR_NOT_SUPPORTED** While the PMIx library supports this operation, the host RM does not.
- a non-zero PMIx error constant indicating a reason for the request's failure

Required Attributes

There are no identified required attributes for implementers.

Description

Non-blocking version of the **PMIx_Group_leave** operation. The callback function will be called once the event has been locally generated and is not indicative of remote receipt.

CHAPTER 14

Data Structures and Types

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

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

▼ Advice to PMIx library implementers ▼

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

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



▼ Advice to users ▼

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

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

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

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

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

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

5 14.1 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

1 14.1.1 PMIx Error Constants

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

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

▼ Advice to PMIx library implementers ▼

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



▼ Advice to users ▼

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



13 14.1.1.1 General Error Constants

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

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

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

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

9 14.1.1.2 Operational Error Constants

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

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

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

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

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

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

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

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

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

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

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

27 **PMIX_GROUP_CONTEXT_ID_ASSIGNED** A new PGCID has been assigned by the host
28 environment to a group that includes this process - the group identifier will be included in the
29 notification.

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

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

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

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

40 14.1.1.3 System error constants

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

1 **PMIX_ERR_NODE_DOWN** A node has gone down - the identifier of the affected node will be
 2 included in the notification
 3 **PMIX_ERR_NODE_OFFLINE** A node has been marked as *offline* - the identifier of the affected
 4 node will be included in the notification
 5 **PMIX_ERR_SYS_OTHER** Mark the end of a dedicated range of constants for system event
 6 reporting.

7 **14.1.1.4 Event handler error constants**

8 **PMIX_EVENT_NO_ACTION_TAKEN** Event handler: No action taken
 9 **PMIX_EVENT_PARTIAL_ACTION_TAKEN** Event handler: Partial action taken
 10 **PMIX_EVENT_ACTION_DEFERRED** Event handler: Action deferred
 11 **PMIX_EVENT_ACTION_COMPLETE** Event handler: Action complete

12 **14.1.1.5 User-Defined Error Constants**

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

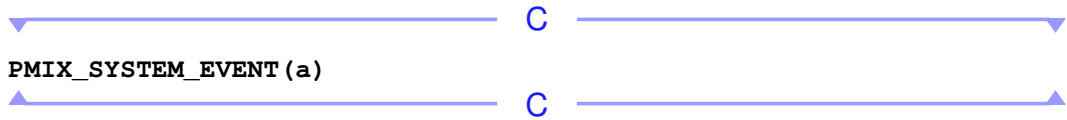
16 **PMIX_EXTERNAL_ERR_BASE** A starting point for user-level defined error constants.
 17 Negative values lower than this are guaranteed not to conflict with PMIx values. Definitions
 18 should always be based on the **PMIX_EXTERNAL_ERR_BASE** constant and not a specific
 19 value as the value of the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

20 **14.1.2 Macros for use with PMIx constants**

21 **14.1.2.1 Detect system event constant**

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

PMIx v2.2



24 **PMIX_SYSTEM_EVENT (a)**
 25 **IN** a
 26 Error constant to be checked (**pmix_status_t**)

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

29 **14.2 Data Types**

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

1 14.2.1 Key Structure

2 The `pmix_key_t` structure is a statically defined character array of length `PMIX_MAX_KEYLEN`
3 +1, thus supporting keys of maximum length `PMIX_MAX_KEYLEN` while preserving space for a
4 mandatory `NULL` terminator.

PMIx v2.0

```
▼ C ▼  
5 typedef char pmix_key_t [PMIX_MAX_KEYLEN+1];  
▲ C ▲
```

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

▼ Advice to users ▼

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

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

▲

15 14.2.1.1 Key support macro

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

PMIx v3.0

```
▼ C ▼  
17 PMIX_CHECK_KEY (a, b)  
▲ C ▲
```

18 **IN** a

19 Pointer to the structure whose key is to be checked (pointer to `pmix_info_t`)

20 **IN** b

21 String value to be compared against (`char*`)

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

1 14.2.3 Rank Structure

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

PMIx v1.0

```
3 typedef uint32_t pmix_rank_t;
```

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

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

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

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

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

17 **PMIX_RANK_INVALID** An invalid rank value.

18 **PMIX_RANK_VALID** Define an upper boundary for valid rank values.

19 14.2.4 Process Structure

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

PMIx v1.0

```
22 typedef struct pmix_proc {  
23     pmix_nspace_t nspace;  
24     pmix_rank_t rank;  
25 } pmix_proc_t;
```

26 14.2.5 Process structure support macros

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

1 14.2.5.1 Initialize the `pmix_proc_t` structure

2 `PMIX_PROC_CONSTRUCT`

3 Initialize the `pmix_proc_t` fields

PMIx v1.0

▼  ▼

4 `PMIX_PROC_CONSTRUCT (m)`

▲  ▲

5 **IN** `m`

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

7 14.2.5.2 Destruct the `pmix_proc_t` structure

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

11 14.2.5.3 Create a `pmix_proc_t` array

12 Allocate and initialize an array of `pmix_proc_t` structures

PMIx v1.0

▼  ▼

13 `PMIX_PROC_CREATE (m, n)`

▲  ▲

14 **INOUT** `m`

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

16 **IN** `n`

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

18 14.2.5.4 Free a `pmix_proc_t` array

19 Release an array of `pmix_proc_t` structures

PMIx v1.0

▼  ▼

20 `PMIX_PROC_FREE (m, n)`

▲  ▲

21 **IN** `m`

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

23 **IN** `n`

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

1 14.2.5.5 Load a `pmix_proc_t` structure

2 Load values into a `pmix_proc_t`

PMIx v2.0



3 `PMIX_PROC_LOAD (m, n, r)`



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

6 **IN** `n`
7 Namespace to be loaded (`pmix_namespace_t`)

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

10 14.2.5.6 Compare identifiers

11 Compare two `pmix_proc_t` identifiers

PMIx v3.0



12 `PMIX_CHECK_PROCID (a, b)`



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

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

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

- 18 • the ranks are the same value
- 19 • one of the ranks is `PMIX_RANK_WILDCARD`

20 14.2.6 Process State Structure

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



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

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

28 14.2.7 Process Information Structure

29 The **pmix_proc_info_t** structure defines a set of information about a specific process
30 including its name, location, and state.

PMIx v2.0

```

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

```

15 **14.2.8 Process Information Structure support macros**

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

17 **14.2.8.1 Initialize the `pmix_proc_info_t` structure**

18 Initialize the `pmix_proc_info_t` fields

PMIx v2.0

```

19  PMIX_PROC_INFO_CONSTRUCT (m)

```

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

22 **14.2.8.2 Destruct the `pmix_proc_info_t` structure**

23 Destruct the `pmix_proc_info_t` fields

PMIx v2.0

```

24  PMIX_PROC_INFO_DESTRUCT (m)

```

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

1 14.2.8.3 Create a `pmix_proc_info_t` array

2 Allocate and initialize a `pmix_proc_info_t` array

PMIx v2.0

C

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

C

4 **INOUT** *m*

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

7 **IN** *n*

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

9 14.2.8.4 Free a `pmix_proc_info_t` array

10 Release an array of `pmix_proc_info_t` structures

PMIx v2.0

C

11 `PMIX_PROC_INFO_FREE` (*m*, *n*)

C

12 **IN** *m*

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

14 **IN** *n*

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

16 14.2.9 Scope of Put Data

17 *PMIx v1.0*

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

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

23 **PMIX_SCOPE_UNDEF** Undefined scope

24 **PMIX_LOCAL** The data is intended only for other application processes on the same node.

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

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

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

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

1 14.2.10 Job State Structure

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

Advice to users

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

8 **PMIX_JOB_STATE_UNDEF** Undefined job state
9 **PMIX_JOB_STATE_PREPPED** Job is ready to be launched
10 **PMIX_JOB_STATE_LAUNCH_UNDERWAY** Job launch is underway
11 **PMIX_JOB_STATE_RUNNING** All processes in the job have been spawned
12 **PMIX_JOB_STATE_SUSPENDED** All processes in the job have been suspended
13 **PMIX_JOB_STATE_CONNECTED** All processes in the job have connected to their PMIx
14 server
15 **PMIX_JOB_STATE_UNTERMINATED** Define a “boundary” between the terminated states
16 and **PMIX_JOB_STATE_TERMINATED** so users can easily and quickly determine if a job
17 is still running or not. Any value less than this constant means that the job has not terminated.
18 **PMIX_JOB_STATE_TERMINATED** All processes in the job have terminated and are no
19 longer running - typically will be accompanied by the job exit status in response to a query
20 **PMIX_JOB_STATE_TERMINATED_WITH_ERROR** Define a boundary so users can easily
21 and quickly determine if a job abnormally terminated - typically will be accompanied by a
22 job-related error code in response to a query Any value above this constant means that the job
23 terminated abnormally.

24 14.2.11 Range of Published Data

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

30 **PMIX_RANGE_UNDEF** Undefined range
31 **PMIX_RANGE_RM** Data is intended for the host resource manager.
32 **PMIX_RANGE_LOCAL** Data is only available to processes on the local node.
33 **PMIX_RANGE_NAMESPACE** Data is only available to processes in the same namespace.
34 **PMIX_RANGE_SESSION** Data is only available to all processes in the session.
35 **PMIX_RANGE_GLOBAL** Data is available to all processes.
36 **PMIX_RANGE_CUSTOM** Range is specified in the `pmix_info_t` associated with this call.

1 **PMIX_RANGE_PROC_LOCAL** Data is only available to this process.
2 **PMIX_RANGE_INVALID** Invalid value

Advice to users

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

5 14.2.12 Data Persistence Structure

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

10 **PMIX_PERSIST_INDEF** Retain data until specifically deleted.
11 **PMIX_PERSIST_FIRST_READ** Retain data until the first access, then the data is deleted.
12 **PMIX_PERSIST_PROC** Retain data until the publishing process terminates.
13 **PMIX_PERSIST_APP** Retain data until the application terminates.
14 **PMIX_PERSIST_SESSION** Retain data until the session/allocation terminates.
15 **PMIX_PERSIST_INVALID** Invalid value

16 14.2.13 Data Array Structure

PMIx v2.0

```
17 typedef struct pmix_data_array  
18     pmix_data_type_t type;  
19     size_t size;  
20     void *array;  
21 pmix_data_array_t;
```

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

24 14.2.14 Data array structure support macros

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

1 14.2.14.1 Initialize the `pmix_data_array_t` structure

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

PMIx v2.2

▼  ▼

3 **PMIX_DATA_ARRAY_CONSTRUCT**(*m*, *n*, *t*)

▲  ▲

4 **IN** *m*

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

6 **IN** *n*

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

8 **IN** *t*

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

10 14.2.14.2 Destruct the `pmix_data_array_t` structure

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

PMIx v2.2

▼  ▼

12 **PMIX_DATA_ARRAY_DESTRUCT**(*m*)

▲  ▲

13 **IN** *m*

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

15 14.2.14.3 Create and initialize a `pmix_data_array_t` object

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

PMIx v2.2

▼  ▼

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

▲  ▲

19 **INOUT** *m*

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

21 **IN** *n*

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

23 **IN** *t*

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

1 14.2.14.4 Free a `pmix_data_array_t` object

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

PMIx v2.2

C

3 `PMIX_DATA_ARRAY_FREE(m)`

C

4 **IN** `m`

5 Pointer to the `pmix_data_array_t` structure (handle)

6 14.2.15 Value Structure

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

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

PMIx v1.0

C


```
13 typedef struct pmix_value {
14     pmix_data_type_t type;
15     union {
16         bool flag;
17         uint8_t byte;
18         char *string;
19         size_t size;
20         pid_t pid;
21         int integer;
22         int8_t int8;
23         int16_t int16;
24         int32_t int32;
25         int64_t int64;
26         unsigned int uint;
27         uint8_t uint8;
28         uint16_t uint16;
29         uint32_t uint32;
30         uint64_t uint64;
31         float fval;
32         double dval;
33         struct timeval tv;
34         time_t time; // version 2.0
35         pmix_status_t status; // version 2.0
```



```

1      pmix_rank_t rank;                // version 2.0
2      pmix_proc_t *proc;              // version 2.0
3      pmix_byte_object_t bo;
4      pmix_persistence_t persist;    // version 2.0
5      pmix_scope_t scope;            // version 2.0
6      pmix_data_range_t range;       // version 2.0
7      pmix_proc_state_t state;       // version 2.0
8      pmix_proc_info_t *pinfo;       // version 2.0
9      pmix_data_array_t *darray;     // version 2.0
10     void *ptr;                      // version 2.0
11     pmix_alloc_directive_t adir;    // version 2.0
12     } data;
13 } pmix_value_t;

```



C

14.2.16 Value structure support macros


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

14.2.16.1 Initialize the `pmix_value_t` structure

Initialize the `pmix_value_t` fields

PMIx v1.0

```
PMIX_VALUE_CONSTRUCT(m)
```



C

IN m


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

14.2.16.2 Destruct the `pmix_value_t` structure

Destruct the `pmix_value_t` fields

PMIx v1.0

```
PMIX_VALUE_DESTRUCT(m)
```



C

IN m

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

1 14.2.16.3 Create a `pmix_value_t` array

2 Allocate and initialize an array of `pmix_value_t` structures

PMIx v1.0

▼ `PMIX_VALUE_CREATE` C

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

▲

4 **INOUT** `m`

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

6 **IN** `n`

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

8 14.2.16.4 Free a `pmix_value_t` array

9 Release an array of `pmix_value_t` structures

PMIx v1.0

▼ `PMIX_VALUE_FREE` C

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

▲

11 **IN** `m`

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

13 **IN** `n`

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

15 14.2.16.5 Load a value structure

16 **Summary**

17 Load data into a `pmix_value_t` structure.

PMIx v2.0

▼ `PMIX_VALUE_LOAD` C

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

▲

19 **IN** `v`

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

21 **IN** `d`

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

23 **IN** `t`

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

Description

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

▼ Advice to users ▼

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

▲

14.2.16.6 Unload a `pmix_value_t` structure

Summary

Unload data from a `pmix_value_t` structure.

PMIx v2.2

▼ C ▼

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

▲ C ▲

OUT `r`

Status code indicating result of the operation `pmix_status_t`

IN `v`

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

INOUT `d`

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

INOUT `t`

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

Description

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

▼ Advice to users ▼

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

▲

1 14.2.16.7 Transfer data between `pmix_value_t` structures

2 Summary

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

PMIx v2.0

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

5 OUT r

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

7 IN d

8 Pointer to the `pmix_value_t` destination (handle)

9 IN s

10 Pointer to the `pmix_value_t` source (handle)

11 Description

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

▼ Advice to users ▼

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

16 14.2.16.8 Retrieve a numerical value from a `pmix_value_t`

17 Retrieve a numerical value from a `pmix_value_t` structure

PMIx v3.0

```
▼ C ▼  
18 PMIX_VALUE_GET_NUMBER(s, m, n, t)  
▲ C ▲
```

19 OUT s

20 Status code for the request (`pmix_status_t`)

21 IN m

22 Pointer to the `pmix_value_t` structure (handle)

23 OUT n

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

25 IN t

26 Type of number expected in *m* (`pmix_data_type_t`)

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

29 `PMIX_ERR_BAD_PARAM` if it doesn't

1 14.2.17 Info Structure

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

PMIx v1.0

```
4 typedef struct pmix_info_t {  
5     pmix_key_t key;  
6     pmix_info_directives_t flags;    // version 2.0  
7     pmix_value_t value;  
8 }
```

9 14.2.18 Info structure support macros

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

11 14.2.18.1 Initialize the `pmix_info_t` structure

12 Initialize the `pmix_info_t` fields

PMIx v1.0

```
13 PMIX_INFO_CONSTRUCT (m)
```

14 **IN** m

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

16 14.2.18.2 Destruct the `pmix_info_t` structure

17 Destruct the `pmix_info_t` fields

PMIx v1.0

```
18 PMIX_INFO_DESTRUCT (m)
```

19 **IN** m

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

21 14.2.18.3 Create a `pmix_info_t` array

22 Allocate and initialize an array of `pmix_info_t` structures

PMIx v1.0

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

24 **INOUT** m

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

26 **IN** n

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

1 14.2.18.4 Free a `pmix_info_t` array

2 Release an array of `pmix_info_t` structures

PMIx v1.0

C

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

C

4 **IN** `m`

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

6 **IN** `n`

Number of structures in the array (`size_t`)

8 14.2.18.5 Load key and value data into a `pmix_info_t`

PMIx v1.0

C

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

C

10 **IN** `v`

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

13 **IN** `k`

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

16 **IN** `d`

Pointer to the data value to be loaded (handle)

18 **IN** `t`

Type of the provided data value (`pmix_data_type_t`)

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

Advice to users

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

1 14.2.18.6 Copy data between `pmix_info_t` structures

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

PMIx v2.0

3 `PMIX_INFO_XFER(d, s);`

4 **IN** `d`

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

6 **IN** `s`

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

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

Advice to users

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

12 14.2.18.7 Test a boolean `pmix_info_t`

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

PMIx v2.0

14 `PMIX_INFO_TRUE(m)`

15 **IN** `m`

16 Pointer to a `pmix_info_t` structure (handle)

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

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

1 14.2.19 Info Type Directives

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

Advice to users

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

Advice to PMIx library implementers

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

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

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

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

Advice to PMIx server hosts

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

25 14.2.20 Info Directive support macros

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

1 14.2.20.1 Mark an info structure as required

2 Summary

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

PMIx v2.0

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

5 **IN** `info`

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

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

8 14.2.20.2 Mark an info structure as optional

9 Summary

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

PMIx v2.0

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

12 **IN** `info`

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

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

15 14.2.20.3 Test an info structure for *required* directive

16 Summary

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

PMIx v2.0

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

19 **IN** `info`

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

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

1 14.2.20.4 Test an info structure for *optional* directive

2 Summary

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

PMIx v2.0

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

5 **IN** `info`

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

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

9 14.2.20.5 Test an info structure for *end of array* directive

10 Summary

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

PMIx v2.2

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

14 **IN** `info`

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

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

17 14.2.21 Job Allocation Directives

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

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

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

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

29 **PMIX_ALLOC_REAQUIRE** Reacquire resources that were previously “lent” back to the
30 scheduler.

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

1 14.2.22 IO Forwarding Channels



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

5 `PMIX_FWD_NO_CHANNELS` Forward no channels
6 `PMIX_FWD_STDIN_CHANNEL` Forward stdin
7 `PMIX_FWD_STDOUT_CHANNEL` Forward stdout
8 `PMIX_FWD_STDERR_CHANNEL` Forward stderr
9 `PMIX_FWD_STDDIAG_CHANNEL` Forward stderr, if available
10 `PMIX_FWD_ALL_CHANNELS` Forward all available channels

11 14.2.23 Environmental Variable Structure

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

```
18 typedef struct  
19     char *envar;  
20     char *value;  
21     char separator;  
22     pmix_envar_t;
```



23 14.2.24 Environmental variable support macros

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

25 14.2.24.1 Initialize the `pmix_envar_t` structure

26 Initialize the `pmix_envar_t` fields

PMIx v3.0

```
27 PMIX_ENVAR_CONSTRUCT(m)
```



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

1 **14.2.24.2 Destruct the `pmix_envar_t` structure**

2 Clear the `pmix_envar_t` fields

PMIx v3.0



3 **PMIX_ENVAR_DESTRUCT (m)**



4 **IN m**

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

6 **14.2.24.3 Create a `pmix_envar_t` array**

7 Allocate and initialize an array of `pmix_envar_t` structures

PMIx v3.0



8 **PMIX_ENVAR_CREATE (m, n)**



9 **INOUT m**

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

11 **IN n**

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

13 **14.2.24.4 Free a `pmix_envar_t` array**

14 Release an array of `pmix_envar_t` structures

PMIx v3.0



15 **PMIX_ENVAR_FREE (m, n)**



16 **IN m**

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

18 **IN n**

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

1 14.2.24.5 Load a `pmix_envvar_t` structure

2 Load values into a `pmix_envvar_t`

PMIx v2.0

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

IN `m`

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

IN `e`

Environmental variable name (`char*`)

IN `v`

Value of variable (`char*`)

IN `s`

Separator character (`char`)

12 14.2.25 Lookup Returned Data Structure

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

PMIx v1.0

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

19 14.2.26 Lookup data structure support macros

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

21 14.2.26.1 Initialize the `pmix_pdata_t` structure

22 Initialize the `pmix_pdata_t` fields

PMIx v1.0

```
PMIX_PDATA_CONSTRUCT(m)
```

IN `m`

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

1 **14.2.26.2 Destruct the `pmix_pdata_t` structure**

2 Destruct the `pmix_pdata_t` fields

PMIx v1.0



3 **PMIX_PDATA_DESTRUCT (m)**



4 **IN m**

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

6 **14.2.26.3 Create a `pmix_pdata_t` array**

7 Allocate and initialize an array of `pmix_pdata_t` structures

PMIx v1.0



8 **PMIX_PDATA_CREATE (m, n)**



9 **INOUT m**

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

11 **IN n**

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

13 **14.2.26.4 Free a `pmix_pdata_t` array**

14 Release an array of `pmix_pdata_t` structures

PMIx v1.0



15 **PMIX_PDATA_FREE (m, n)**



16 **IN m**

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

18 **IN n**

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

1 14.2.26.5 Load a lookup data structure

2 Summary

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

PMIx v1.0

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

5 **IN** `m`

6 Pointer to the `pmix_pdata_t` structure into which the key and data are to be loaded
7 (pointer to `pmix_pdata_t`)

8 **IN** `p`

9 Pointer to the `pmix_proc_t` structure containing the identifier of the process being
10 referenced (pointer to `pmix_proc_t`)

11 **IN** `k`

12 String key to be loaded - must be less than or equal to `PMIX_MAX_KEYLEN` in length
13 (handle)

14 **IN** `d`

15 Pointer to the data value to be loaded (handle)

16 **IN** `t`

17 Type of the provided data value (`pmix_data_type_t`)

18 This macro simplifies the loading of key, process identifier, and data into a `pmix_proc_t` by
19 correctly assigning values to the structure's fields.

Advice to users

20 Key, process identifier, and data will all be copied into the `pmix_pdata_t` - thus, the source
21 information can be modified or free'd without affecting the copied data once the macro has
22 completed.

1 14.2.26.6 Transfer a lookup data structure

2 Summary

3 Transfer key, process identifier, and data value between two `pmix_pdata_t` structures.

PMIx v2.0

C

```
4 PMIX_PDATA_XFER(d, s);
```

C

5 **IN** `d`

6 Pointer to the destination `pmix_pdata_t` (pointer to `pmix_pdata_t`)

7 **IN** `s`

8 Pointer to the source `pmix_pdata_t` (pointer to `pmix_pdata_t`)

9 This macro simplifies the transfer of key and data between two `pmix_pdata_t` structures.

Advice to users

10 Key, process identifier, and data will all be copied into the destination `pmix_pdata_t` - thus, the
11 source `pmix_pdata_t` may free'd without affecting the copied data once the macro has
12 completed.

13 14.2.27 Application Structure

14 The `pmix_app_t` structure describes the application context for the `PMIx_Spawn` and
15 `PMIx_Spawn_nb` operations.

PMIx v1.0

C

```
16 typedef struct pmix_app {  
17     /** Executable */  
18     char *cmd;  
19     /** Argument set, NULL terminated */  
20     char **argv;  
21     /** Environment set, NULL terminated */  
22     char **env;  
23     /** Current working directory */  
24     char *cwd;  
25     /** Maximum processes with this profile */  
26     int maxprocs;  
27     /** Array of info keys describing this application*/  
28     pmix_info_t *info;  
29     /** Number of info keys in 'info' array */  
30     size_t ninfo;  
31 } pmix_app_t;
```

C

1 14.2.28 App structure support macros

2 The following macros are provided to support the `pmix_app_t` structure.

3 14.2.28.1 Initialize the `pmix_app_t` structure

4 Initialize the `pmix_app_t` fields

PMIx v1.0 ▼  C 

5 **PMIX_APP_CONSTRUCT** (m)

▲  C 

6 **IN** m

7 Pointer to the structure to be initialized (pointer to `pmix_app_t`)

8 14.2.28.2 Destruct the `pmix_app_t` structure

9 Destruct the `pmix_app_t` fields

PMIx v1.0 ▼  C 

10 **PMIX_APP_DESTRUCT** (m)

▲  C 

11 **IN** m

12 Pointer to the structure to be destructed (pointer to `pmix_app_t`)

13 14.2.28.3 Create a `pmix_app_t` array

14 Allocate and initialize an array of `pmix_app_t` structures

PMIx v1.0 ▼  C 

15 **PMIX_APP_CREATE** (m, n)

▲  C 

16 **INOUT** m

17 Address where the pointer to the array of `pmix_app_t` structures shall be stored (handle)

18 **IN** n

19 Number of structures to be allocated (`size_t`)

20 14.2.28.4 Free a `pmix_app_t` array

21 Release an array of `pmix_app_t` structures

PMIx v1.0 ▼  C 

22 **PMIX_APP_FREE** (m, n)

▲  C 

23 **IN** m

24 Pointer to the array of `pmix_app_t` structures (handle)

25 **IN** n

26 Number of structures in the array (`size_t`)

1 14.2.28.5 Create the `pmix_info_t` array of application directives

2 Create an array of `pmix_info_t` structures for passing application-level directives, updating the
3 `ninfo` field of the `pmix_app_t` structure.

PMIx v2.2

▼ C _____ ▼

4 `PMIX_APP_INFO_CREATE(m, n)`

▲ C _____ ▲

5 **IN** `m`

6 Pointer to the `pmix_app_t` structure (handle)

7 **IN** `n`

8 Number of directives to be allocated (`size_t`)

9 14.2.29 Query Structure

10 The `pmix_query_t` structure is used by `PMIx_Query_info_nb` to describe a single query
11 operation.

PMIx v2.0

▼ C _____ ▼

```
12 typedef struct pmix_query {  
13     char **keys;  
14     pmix_info_t *qualifiers;  
15     size_t nqual;  
16 } pmix_query_t;
```

▲ C _____ ▲

17 14.2.30 Query structure support macros

18 The following macros are provided to support the `pmix_query_t` structure.

19 14.2.30.1 Initialize the `pmix_query_t` structure

20 Initialize the `pmix_query_t` fields

PMIx v2.0

▼ C _____ ▼

21 `PMIX_QUERY_CONSTRUCT(m)`

▲ C _____ ▲

22 **IN** `m`

23 Pointer to the structure to be initialized (pointer to `pmix_query_t`)

1 14.2.30.2 Destruct the `pmix_query_t` structure

2 Destruct the `pmix_query_t` fields

PMIx v2.0

▼ `PMIX_QUERY_DESTRUCT` C

3 `PMIX_QUERY_DESTRUCT` (m)

▲ C

4 **IN** m

5 Pointer to the structure to be destructed (pointer to `pmix_query_t`)

6 14.2.30.3 Create a `pmix_query_t` array

7 Allocate and initialize an array of `pmix_query_t` structures

PMIx v2.0

▼ `PMIX_QUERY_CREATE` C

8 `PMIX_QUERY_CREATE` (m, n)

▲ C

9 **INOUT** m

10 Address where the pointer to the array of `pmix_query_t` structures shall be stored (handle)

11 **IN** n

12 Number of structures to be allocated (`size_t`)

13 14.2.30.4 Free a `pmix_query_t` array

14 Release an array of `pmix_query_t` structures

PMIx v2.0

▼ `PMIX_QUERY_FREE` C

15 `PMIX_QUERY_FREE` (m, n)

▲ C

16 **IN** m

17 Pointer to the array of `pmix_query_t` structures (handle)

18 **IN** n

19 Number of structures in the array (`size_t`)

20 14.2.30.5 Create the `pmix_info_t` array of query qualifiers

21 Create an array of `pmix_info_t` structures for passing query qualifiers, updating the `nqual` field
22 of the `pmix_query_t` structure.

PMIx v2.2

▼ `PMIX_QUERY_QUALIFIERS_CREATE` C

23 `PMIX_QUERY_QUALIFIERS_CREATE` (m, n)

▲ C

24 **IN** m

25 Pointer to the `pmix_query_t` structure (handle)

26 **IN** n

27 Number of qualifiers to be allocated (`size_t`)

1 14.2.31 Attribute registration structure

2 The `pmix_regattr_t` structure is used to register attribute support for a PMIx function.

PMIx v4.0

```
3 typedef struct pmix_regattr {  
4     char *name;  
5     pmix_key_t *string;  
6     pmix_data_type_t type;  
7     pmix_info_t *info;  
8     size_t ninfo;  
9     char **description;  
10 } pmix_regattr_t;
```

11 Note that in this structure:

- 12 • the *name* is the actual name of the attribute - e.g., "PMIX_MAX_PROCS"; and
- 13 • the *string* is the literal string value of the attribute - e.g., "pmix.max.size" for the `PMIX_MAX_PROCS` attribute
- 14 • *type* must be a PMIx data type identifying the type of data associated with this attribute.
- 15 • the *info* array contains machine-usable information regarding the range of accepted values. This may include entries for `PMIX_MIN_VALUE`, `PMIX_MAX_VALUE`, `PMIX_ENUM_VALUE`, or a combination of them. For example, an attribute that supports all positive integers might delineate it by including a `pmix_info_t` with a key of `PMIX_MIN_VALUE`, type of `PMIX_INT`, and value of zero. The lack of an entry for `PMIX_MAX_VALUE` indicates that there is no ceiling to the range of accepted values.
- 16 • *ninfo* indicates the number of elements in the *info* array
- 17 • The *description* field consists of a **NULL**-terminated array of strings describing the attribute, optionally including a human-readable description of the range of accepted values - e.g., "ALL POSITIVE INTEGERS", or a comma-delimited list of enum value names. No correlation between the number of entries in the *description* and the number of elements in the *info* array is implied or required.

18 The attribute *name* and *string* fields must be **NULL**-terminated strings composed of standard alphanumeric values supported by common utilities such as *strcmp*.

Advice to PMIx library implementers

29 Although not strictly required, PMIx library implementers are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes.

Advice to PMIx server hosts

Although not strictly required, host environments are strongly encouraged to provide both human-readable and machine-parsable descriptions of supported attributes when registering them.

14.2.32 Attribute registration structure support macros

The following macros are provided to support the `pmix_regattr_t` structure.

14.2.32.1 Initialize the `pmix_regattr_t` structure

Initialize the `pmix_regattr_t` fields

PMIx v4.0

PMIX_REGATTR_CONSTRUCT (m)

IN m

Pointer to the structure to be initialized (pointer to `pmix_regattr_t`)

14.2.32.2 Destruct the `pmix_regattr_t` structure

Destruct the `pmix_regattr_t` fields, releasing all strings.

PMIx v4.0

PMIX_REGATTR_DESTRUCT (m)

IN m

Pointer to the structure to be destructed (pointer to `pmix_regattr_t`)

14.2.32.3 Create a `pmix_regattr_t` array

Allocate and initialize an array of `pmix_regattr_t` structures

PMIx v4.0

PMIX_REGATTR_CREATE (m, n)

INOUT m

Address where the pointer to the array of `pmix_regattr_t` structures shall be stored (handle)

IN n

Number of structures to be allocated (`size_t`)

1 14.2.32.4 Free a `pmix_regattr_t` array

2 Release an array of `pmix_regattr_t` structures

PMIx v4.0

C

3 **PMIX_REGATTR_FREE**(*m*, *n*)

C

4 **INOUT** *m*

5 Pointer to the array of `pmix_regattr_t` structures (handle)

6 **IN** *n*

7 Number of structures in the array (**size_t**)

8 14.2.32.5 Load a `pmix_regattr_t` structure

9 Load values into a `pmix_regattr_t` structure. The macro can be called multiple times to add
10 as many strings as desired to the same structure by passing the same address and a **NULL** key to the
11 macro. Note that the *t* type value must be given each time.

PMIx v4.0

C

12 **PMIX_REGATTR_LOAD**(*a*, *n*, *k*, *t*, *ni*, *v*)

C

13 **IN** *a*

14 Pointer to the structure to be loaded (pointer to `pmix_proc_t`)

15 **IN** *n*

16 String name of the attribute (string)

17 **IN** *k*

18 Key value to be loaded (`pmix_key_t`)

19 **IN** *t*

20 Type of data associated with the provided key (`pmix_data_type_t`)

21 **IN** *ni*

22 Number of `pmix_info_t` elements to be allocated in *info* (**size_t**)

23 **IN** *v*

24 One-line description to be loaded (more can be added separately) (string)

25 14.2.32.6 Transfer a `pmix_regattr_t` to another `pmix_regattr_t`

26

27 Non-destructively transfer the contents of a `pmix_regattr_t` structure to another one.

PMIx v4.0

C

28 **PMIX_REGATTR_XFER**(*m*, *n*)

C

29 **INOUT** *m*

30 Pointer to the destination `pmix_regattr_t` structure (handle)

31 **IN** *n*

32 Pointer to the source `pmix_regattr_t` structure (handle)

1 14.2.33 PMIx Group Directives

2 *PMIx v4.0* The `pmix_group_opt_t` type is an enumerated type used with the `PMIx_Group_join` API
3 to indicate *accept* or *decline* of the invitation - these are provided for readability of user code:

4 `PMIX_GROUP_DECLINE` Decline the invitation
5 `PMIX_GROUP_ACCEPT` Accept the invitation.

6 14.2.34 Byte Object Type

7 The `pmix_byte_object_t` structure describes a raw byte sequence.

PMIx v1.0

```
▼ C _____  
8 typedef struct pmix_byte_object {  
9     char *bytes;  
10    size_t size;  
11 } pmix_byte_object_t;  
▲ C _____
```

12 14.2.35 Byte object support macros

13 The following macros support the `pmix_byte_object_t` structure.

14 14.2.35.1 Initialize the `pmix_byte_object_t` structure

15 Initialize the `pmix_byte_object_t` fields

PMIx v2.0

```
▼ C _____  
16 PMIX_BYTE_OBJECT_CONSTRUCT (m)  
▲ C _____
```

IN m

Pointer to the structure to be initialized (pointer to `pmix_byte_object_t`)

19 14.2.35.2 Destruct the `pmix_byte_object_t` structure

20 Clear the `pmix_byte_object_t` fields

PMIx v2.0

```
▼ C _____  
21 PMIX_BYTE_OBJECT_DESTRUCT (m)  
▲ C _____
```

IN m

Pointer to the structure to be destructed (pointer to `pmix_byte_object_t`)

1 14.2.35.3 Create a `pmix_byte_object_t` structure

2 Allocate and initialize an array of `pmix_byte_object_t` structures

PMIx v2.0

▼ `C` _____ ▼

3 **PMIX_BYTE_OBJECT_CREATE**(*m*, *n*)

▲ _____ `C` _____ ▲

4 **INOUT** *m*

5 Address where the pointer to the array of `pmix_byte_object_t` structures shall be
6 stored (handle)

7 **IN** *n*

8 Number of structures to be allocated (**size_t**)

9 14.2.35.4 Free a `pmix_byte_object_t` array

10 Release an array of `pmix_byte_object_t` structures

PMIx v2.0

▼ `C` _____ ▼

11 **PMIX_BYTE_OBJECT_FREE**(*m*, *n*)

▲ _____ `C` _____ ▲

12 **IN** *m*

13 Pointer to the array of `pmix_byte_object_t` structures (handle)

14 **IN** *n*

15 Number of structures in the array (**size_t**)

16 14.2.35.5 Load a `pmix_byte_object_t` structure

17 Load values into a `pmix_byte_object_t`

PMIx v2.0

▼ `C` _____ ▼

18 **PMIX_BYTE_OBJECT_LOAD**(*b*, *d*, *s*)

▲ _____ `C` _____ ▲

19 **IN** *b*

20 Pointer to the structure to be loaded (pointer to `pmix_byte_object_t`)

21 **IN** *d*

22 Pointer to the data to be loaded (**char***)

23 **IN** *s*

24 Number of bytes in the data array (**size_t**)

1 14.2.36 Data Array Structure

2 The `pmix_data_array_t` structure defines an array data structure.

PMIx v2.0

```
3 typedef struct pmix_data_array {  
4     pmix_data_type_t type;  
5     size_t size;  
6     void *array;  
7 } pmix_data_array_t;
```

8 14.2.37 Data array support macros

9 The following macros support the `pmix_data_array_t` structure.

10 14.2.37.1 Initialize a `pmix_data_array_t` structure

11 Initialize the `pmix_data_array_t` fields, allocating memory for the array of the indicated type.

PMIx v2.2

```
12 PMIX_DATA_ARRAY_CONSTRUCT(m, n, t)
```

13 **IN** m

14 Pointer to the structure to be initialized (pointer to `pmix_data_array_t`)

15 **IN** n

16 Number of elements in the array (`size_t`)

17 **IN** t

18 PMIx data type of the array elements (`pmix_data_type_t`)

19 14.2.37.2 Destruct a `pmix_data_array_t` structure

20 Destruct the `pmix_data_array_t`, releasing the memory in the array.

PMIx v2.2

```
21 PMIX_DATA_ARRAY_CONSTRUCT(m)
```

22 **IN** m

23 Pointer to the structure to be destructed (pointer to `pmix_data_array_t`)

1 14.2.37.3 Create a `pmix_data_array_t` structure

2 Allocate memory for the `pmix_data_array_t` object itself, and then allocate memory for the
3 array of the indicated type.

PMIx v2.2

```
▼ _____ C _____ ▼  
4 PMIX_DATA_ARRAY_CREATE(m, n, t)  
▲ _____ C _____ ▲
```

5 **INOUT** m

6 Variable to be set to the address of the structure (pointer to `pmix_data_array_t`)

7 **IN** n

8 Number of elements in the array (`size_t`)

9 **IN** t

10 PMIx data type of the array elements (`pmix_data_type_t`)

11 14.2.37.4 Free a `pmix_data_array_t` structure

12 Release the memory in the array, and then release the `pmix_data_array_t` object itself.

PMIx v2.2

```
▼ _____ C _____ ▼  
13 PMIX_DATA_ARRAY_FREE(m)  
▲ _____ C _____ ▲
```

14 **IN** m

15 Pointer to the structure to be released (pointer to `pmix_data_array_t`)

16 14.2.38 Argument Array Macros

17 The following macros support the construction and release of **NULL**-terminated argv arrays of
18 strings.

19 14.2.38.1 Argument array extension

20 **Summary**

21 Append a string to a **NULL**-terminated, argv-style array of strings.

```
▼ _____ C _____ ▼  
22 PMIX_ARGV_APPEND(r, a, b);  
▲ _____ C _____ ▲
```

23 **OUT** r

24 Status code indicating success or failure of the operation (`pmix_status_t`)

25 **INOUT** a

26 Argument list (pointer to **NULL**-terminated array of strings)

27 **IN** b

28 Argument to append to the list (string)

1
2
3
4

5
6

7
8
9
10

11

12
13
14
15
16
17

18
19
20
21

22
23

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.

14.2.38.2 Argument array extension - unique

Summary

Append a string to a NULL-terminated, argv-style array of strings, but only if the provided argument doesn't already exist somewhere in the array.

C

```
PMIX_ARGV_APPEND_UNIQUE(r, a, b);
```

C

OUT `r`

Status code indicating success or failure of the operation (`pmix_status_t`)

INOUT `a`

Argument list (pointer to NULL-terminated array of strings)

IN `b`

Argument to append to the list (string)

Description

This function helps the caller build the **argv** portion of `pmix_app_t` structure, arrays of keys for querying, or other places where argv-style string arrays are required in the way that the PRI expects it to be constructed.

Advice to users

The provided argument is copied into the destination array - thus, the source string can be free'd without affecting the array once the macro has completed.

1 14.2.38.3 Argument array release

2 Summary

3 Free an argv-style array and all of the strings that it contains

▼  C 

4 `PMIX_ARGV_FREE(a);`

▲  C 

5 **IN** a

6 Argument list (pointer to NULL-terminated array of strings)

7 Description

8 This function releases the array and all of the strings it contains.

9 14.2.38.4 Argument array split

10 Summary

11 Split a string into a NULL-terminated argv array.

▼  C 

12 `PMIX_ARGV_SPLIT(a, b, c);`

▲  C 

13 **OUT** a

14 Resulting argv-style array (**char****)

15 **IN** b

16 String to be split (**char***)

17 **IN** c

18 Delimiter character (**char**)

19 Description

20 Split an input string into a NULL-terminated argv array. Do not include empty strings in the
21 resulting array.

▼  Advice to users 

22 All strings are inserted into the argv array by value; the newly-allocated array makes no references
23 to the src_string argument (i.e., it can be freed after calling this function without invalidating the
24 output argv array)

▲ 

1 14.2.38.5 Argument array join

2 Summary

3 Join all the elements of an argv array into a single newly-allocated string.

▼  ▼

4 `PMIX_ARGV_JOIN(a, b, c);`

▲  ▲

5 **OUT** a

6 Resulting string (**char***)

7 **IN** b

8 Argv-style array to be joined (**char****)

9 **IN** c

10 Delimiter character (**char**)

11 Description

12 Join all the elements of an argv array into a single newly-allocated string.

13 14.2.38.6 Argument array count

14 Summary

15 Return the length of a NULL-terminated argv array.

▼  ▼

16 `PMIX_ARGV_COUNT(r, a);`

▲  ▲

17 **OUT** r

18 Number of strings in the array (integer)

19 **IN** a

20 Argv-style array (**char****)

21 Description

22 Count the number of elements in an argv array

23 14.2.38.7 Argument array copy

24 Summary

25 Copy an argv array, including copying all off its strings.

▼  ▼

26 `PMIX_ARGV_COPY(a, b);`

▲  ▲

27 **OUT** a

28 New argv-style array (**char****)

29 **IN** b

30 Argv-style array (**char****)

1 **Description**
2 Copy an argv array, including copying all off its strings.

3 14.2.39 Set Environment Variable

4 **Summary**

5 Set an environment variable in a **NULL**-terminated, env-style array

▼ **C** ▼
6 **PMIX_SETENV**(*r*, *name*, *value*, *env*);
▲ **C** ▲

7 **OUT** *r*

8 Status code indicating success or failure of the operation ([pmix_status_t](#))

9 **IN** *name*

10 Argument name (string)

11 **IN** *value*

12 Argument value (string)

13 **INOUT** *env*

14 Environment array to update (pointer to array of strings)

15 **Description**

16 Similar to **setenv** from the C API, this allows the caller to set an environment variable in the
17 specified **env** array, which could then be passed to the [pmix_app_t](#) structure or any other
18 destination.

▼ **Advice to users** ▼
19 The provided name and value are copied into the destination environment array - thus, the source
20 strings can be free'd without affecting the array once the macro has completed.
▲

1 14.3 Generalized Data Types Used for Packing/Unpacking

2 The `pmix_data_type_t` structure is a `uint16_t` type for identifying the data type for
3 packing/unpacking purposes. New data type values introduced in this version of the Standard are
4 shown in magenta.

Advice to PMIx library implementers

5 The following constants can be used to set a variable of the type `pmix_data_type_t`. Data
6 types in the PMIx Standard are defined in terms of the C-programming language. Implementers
7 wishing to support other languages should provide the equivalent definitions in a
8 language-appropriate manner. Additionally, a PMIx implementation may choose to add additional
9 types.

10	<code>PMIX_UNDEF</code>	Undefined
11	<code>PMIX_BOOL</code>	Boolean (converted to/from native <code>true/false</code>) (<code>bool</code>)
12	<code>PMIX_BYTE</code>	A byte of data (<code>uint8_t</code>)
13	<code>PMIX_STRING</code>	<code>NULL</code> terminated string (<code>char*</code>)
14	<code>PMIX_SIZE</code>	Size <code>size_t</code>
15	<code>PMIX_PID</code>	Operating PID (<code>pid_t</code>)
16	<code>PMIX_INT</code>	Integer (<code>int</code>)
17	<code>PMIX_INT8</code>	8-byte integer (<code>int8_t</code>)
18	<code>PMIX_INT16</code>	16-byte integer (<code>int16_t</code>)
19	<code>PMIX_INT32</code>	32-byte integer (<code>int32_t</code>)
20	<code>PMIX_INT64</code>	64-byte integer (<code>int64_t</code>)
21	<code>PMIX_UINT</code>	Unsigned integer (<code>unsigned int</code>)
22	<code>PMIX_UINT8</code>	Unsigned 8-byte integer (<code>uint8_t</code>)
23	<code>PMIX_UINT16</code>	Unsigned 16-byte integer (<code>uint16_t</code>)
24	<code>PMIX_UINT32</code>	Unsigned 32-byte integer (<code>uint32_t</code>)
25	<code>PMIX_UINT64</code>	Unsigned 64-byte integer (<code>uint64_t</code>)
26	<code>PMIX_FLOAT</code>	Float (<code>float</code>)
27	<code>PMIX_DOUBLE</code>	Double (<code>double</code>)
28	<code>PMIX_TIMEVAL</code>	Time value (<code>struct timeval</code>)
29	<code>PMIX_TIME</code>	Time (<code>time_t</code>)
30	<code>PMIX_STATUS</code>	Status code <code>pmix_status_t</code>
31	<code>PMIX_VALUE</code>	Value (<code>pmix_value_t</code>)
32	<code>PMIX_PROC</code>	Process (<code>pmix_proc_t</code>)
33	<code>PMIX_APP</code>	Application context
34	<code>PMIX_INFO</code>	Info object
35	<code>PMIX_PDATA</code>	Pointer to data
36	<code>PMIX_BUFFER</code>	Buffer
37	<code>PMIX_BYTE_OBJECT</code>	Byte object (<code>pmix_byte_object_t</code>)
38	<code>PMIX_KVAL</code>	Key/value pair

```

1  PMIX_PERSIST Persistence ( pmix\_persistence\_t )
2  PMIX_POINTER Pointer to an object (void*)
3  PMIX_SCOPE Scope ( pmix\_scope\_t )
4  PMIX_DATA_RANGE Range for data ( pmix\_data\_range\_t )
5  PMIX_COMMAND PMIx command code (used internally)
6  PMIX_INFO_DIRECTIVES Directives flag for pmix\_info\_t (
7     pmix\_info\_directives\_t )
8  PMIX_DATA_TYPE Data type code ( pmix\_data\_type\_t )
9  PMIX_PROC_STATE Process state ( pmix\_proc\_state\_t )
10 PMIX_PROC_INFO Process information ( pmix\_proc\_info\_t )
11 PMIX_DATA_ARRAY Data array ( pmix\_data\_array\_t )
12 PMIX_PROC_RANK Process rank ( pmix\_rank\_t )
13 PMIX_QUERY Query structure ( pmix\_query\_t )
14 PMIX_COMPRESSED_STRING String compressed with zlib (char*)
15 PMIX_ALLOC_DIRECTIVE Allocation directive ( pmix\_alloc\_directive\_t )
16 PMIX_IOF_CHANNEL Input/output forwarding channel ( pmix\_iof\_channel\_t )
17 PMIX_ENVAR Environmental variable structure ( pmix\_envar\_t )
18 PMIX_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 14.4 Reserved attributes

23 The PMIx standard defines a relatively small set of APIs and the caller may customize the behavior
24 of the API by passing one or more attributes to that API. Additionally, attributes may be keys
25 passed to [PMIx_Get](#) calls to access the specified values from the system.

26 Each attribute is represented by a *key* string, and a type for the associated *value*. This section
27 defines a set of **reserved** keys which are prefixed with **pmix_.** to designate them as PMIx standard
28 reserved keys. All definitions were introduced in version 1 of the standard unless otherwise marked.

29 Applications or associated libraries (e.g., MPI) may choose to define additional attributes. The
30 attributes defined in this section are of the system and job as opposed to the attributes that the
31 application (or associated libraries) might choose to expose. Due to this extensibility the
32 [PMIx_Get](#) API will return [PMIX_ERR_NOT_FOUND](#) if the provided *key* cannot be found.

33 Attributes added in this version of the standard are shown in *magenta* to distinguish them from
34 those defined in prior versions, which are shown in *black*. Deprecated attributes are shown in *green*
35 and will be removed in future versions of the standard.

36 **PMIX_ATTR_UNDEF NULL (NULL)**
37 Constant representing an undefined attribute.

1 14.4.1 Initialization attributes

2 These attributes are defined to assist the caller with initialization by passing them into the
3 appropriate initialization API - thus, they are not typically accessed via the `PMIx_Get` API.

4 **PMIX_EVENT_BASE** "pmix.evbase" (struct event_base *)
5 Pointer to libevent¹ `event_base` to use in place of the internal progress thread.
6 **PMIX_SERVER_TOOL_SUPPORT** "pmix.srvr.tool" (bool)
7 The host RM wants to declare itself as willing to accept tool connection requests.
8 **PMIX_SERVER_REMOTE_CONNECTIONS** "pmix.srvr.remote" (bool)
9 Allow connections from remote tools. Forces the PMIx server to not exclusively use
10 loopback device.
11 **PMIX_SERVER_SYSTEM_SUPPORT** "pmix.srvr.sys" (bool)
12 The host RM wants to declare itself as being the local system server for PMIx connection
13 requests.
14 **PMIX_SERVER_TMPDIR** "pmix.srvr.tmpdir" (char*)
15 Top-level temporary directory for all client processes connected to this server, and where the
16 PMIx server will place its tool rendezvous point and contact information.
17 **PMIX_SYSTEM_TMPDIR** "pmix.sys.tmpdir" (char*)
18 Temporary directory for this system, and where a PMIx server that declares itself to be a
19 system-level server will place a tool rendezvous point and contact information.
20 **PMIX_SERVER_ENABLE_MONITORING** "pmix.srv.monitor" (bool)
21 Enable PMIx internal monitoring by the PMIx server.
22 **PMIX_SERVER_NAMESPACE** "pmix.srv.namespace" (char*)
23 Name of the namespace to use for this PMIx server.
24 **PMIX_SERVER_RANK** "pmix.srv.rank" (pmix_rank_t)
25 Rank of this PMIx server
26 **PMIX_SERVER_GATEWAY** "pmix.srv.gway" (bool)
27 Server is acting as a gateway for PMIx requests that cannot be serviced on backend nodes
28 (e.g., logging to email)

29 14.4.2 Tool-related attributes

30 These attributes are defined to assist PMIx-enabled tools to connect with the PMIx server by
31 passing them into the `PMIx_tool_init` API - thus, they are not typically accessed via the
32 `PMIx_Get` API.

33 **PMIX_TOOL_NAMESPACE** "pmix.tool.namespace" (char*)
34 Name of the namespace to use for this tool.
35 **PMIX_TOOL_RANK** "pmix.tool.rank" (uint32_t)
36 Rank of this tool.
37 **PMIX_SERVER_PIDINFO** "pmix.srvr.pidinfo" (pid_t)
38 PID of the target PMIx server for a tool.
39 **PMIX_CONNECT_TO_SYSTEM** "pmix.cnct.sys" (bool)

¹<http://libevent.org/>

1 The requestor requires that a connection be made only to a local, system-level PMIx server.
2 **PMIX_CONNECT_SYSTEM_FIRST** "pmix.cnct.sys.first" (bool)
3 Preferentially, look for a system-level PMIx server first.
4 **PMIX_SERVER_URI** "pmix.srvr.uri" (char*)
5 URI of the PMIx server to be contacted.
6 **PMIX_SERVER_HOSTNAME** "pmix.srvr.host" (char*)
7 Host where target PMIx server is located.
8 **PMIX_CONNECT_MAX_RETRIES** "pmix.tool.mretries" (uint32_t)
9 Maximum number of times to try to connect to PMIx server.
10 **PMIX_CONNECT_RETRY_DELAY** "pmix.tool.retry" (uint32_t)
11 Time in seconds between connection attempts to a PMIx server.
12 **PMIX_TOOL_DO_NOT_CONNECT** "pmix.tool.nocon" (bool)
13 The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.
14 **PMIX_RECONNECT_SERVER** "pmix.tool.recon" (bool)
15 Tool is requesting to change server connections
16 **PMIX_LAUNCHER** "pmix.tool.launcher" (bool)
17 Tool is a launcher and needs rendezvous files created

18 14.4.3 Identification attributes

19 These attributes are defined to identify a process and it's associated PMIx-enabled library. They are
20 not typically accessed via the **PMIx_Get** API, and thus are not associated with a particular rank.

21 **PMIX_USERID** "pmix.euid" (uint32_t)
22 Effective user id.
23 **PMIX_GRPID** "pmix.egid" (uint32_t)
24 Effective group id.
25 **PMIX_DSTPATH** "pmix.dstpath" (char*)
26 Path to shared memory data storage (dstore) files.
27 **PMIX_VERSION_INFO** "pmix.version" (char*)
28 PMIx version of contractor.
29 **PMIX_REQUESTOR_IS_TOOL** "pmix.req.tool" (bool)
30 The requesting process is a PMIx tool.
31 **PMIX_REQUESTOR_IS_CLIENT** "pmix.req.client" (bool)
32 The requesting process is a PMIx client.
33 **PMIX_PSET_NAME** "pmix.pset.nm" (char*)
34 User-assigned name for the process set containing the given process.

1 14.4.4 Programming model attributes

2 These attributes are associated with programming models.

3 **PMIX_PROGRAMMING_MODEL** "pmix.pgm.model" (char*)
4 Programming model being initialized (e.g., "MPI" or "OpenMP")
5 **PMIX_MODEL_LIBRARY_NAME** "pmix.mdl.name" (char*)
6 Programming model implementation ID (e.g., "OpenMPI" or "MPICH")
7 **PMIX_MODEL_LIBRARY_VERSION** "pmix.mdl.vrs" (char*)
8 Programming model version string (e.g., "2.1.1")
9 **PMIX_THREADING_MODEL** "pmix.threads" (char*)
10 Threading model used (e.g., "pthreads")
11 **PMIX_MODEL_NUM_THREADS** "pmix.mdl.nthrds" (uint64_t)
12 Number of active threads being used by the model
13 **PMIX_MODEL_NUM_CPUS** "pmix.mdl.ncpu" (uint64_t)
14 Number of cpus being used by the model
15 **PMIX_MODEL_CPU_TYPE** "pmix.mdl.cputype" (char*)
16 Granularity - "hwthread", "core", etc.
17 **PMIX_MODEL_PHASE_NAME** "pmix.mdl.phase" (char*)
18 User-assigned name for a phase in the application execution (e.g., "cfd reduction")
19 **PMIX_MODEL_PHASE_TYPE** "pmix.mdl.ptype" (char*)
20 Type of phase being executed (e.g., "matrix multiply")
21 **PMIX_MODEL_AFFINITY_POLICY** "pmix.mdl.tap" (char*)
22 Thread affinity policy - e.g.: "master" (thread co-located with master thread), "close" (thread
23 located on cpu close to master thread), "spread" (threads load-balanced across available cpus)

24 14.4.5 UNIX socket rendezvous socket attributes

25 These attributes are used to describe a UNIX socket for rendezvous with the local RM by passing
26 them into the relevant initialization API - thus, they are not typically accessed via the [PMIx_Get](#)
27 API.

28 **PMIX_USOCK_DISABLE** "pmix.usock.disable" (bool)
29 Disable legacy UNIX socket (usock) support
30 **PMIX_SOCKET_MODE** "pmix.sockmode" (uint32_t)
31 POSIX *mode_t* (9 bits valid)
32 **PMIX_SINGLE_LISTENER** "pmix.sing.listnr" (bool)
33 Use only one rendezvous socket, letting priorities and/or environment parameters select the
34 active transport.

1 14.4.6 TCP connection attributes

2 These attributes are used to describe a TCP socket for rendezvous with the local RM by passing
3 them into the relevant initialization API - thus, they are not typically accessed via the `PMIx_Get`
4 API.

5 **PMIX_TCP_REPORT_URI** "pmix.tcp.repuri" (char*)

6 If provided, directs that the TCP URI be reported and indicates the desired method of
7 reporting: '-' for stdout, '+' for stderr, or filename.

8 **PMIX_TCP_URI** "pmix.tcp.uri" (char*)

9 The URI of the PMIx server to connect to, or a file name containing it in the form of
10 `file:<name of file containing it>`.

11 **PMIX_TCP_IF_INCLUDE** "pmix.tcp.ifinclude" (char*)

12 Comma-delimited list of devices and/or CIDR notation to include when establishing the
13 TCP connection.

14 **PMIX_TCP_IF_EXCLUDE** "pmix.tcp.ifexclude" (char*)

15 Comma-delimited list of devices and/or CIDR notation to exclude when establishing the
16 TCP connection.

17 **PMIX_TCP_IPV4_PORT** "pmix.tcp.ipv4" (int)

18 The IPv4 port to be used.

19 **PMIX_TCP_IPV6_PORT** "pmix.tcp.ipv6" (int)

20 The IPv6 port to be used.

21 **PMIX_TCP_DISABLE_IPV4** "pmix.tcp.disipv4" (bool)

22 Set to `true` to disable IPv4 family of addresses.

23 **PMIX_TCP_DISABLE_IPV6** "pmix.tcp.disipv6" (bool)

24 Set to `true` to disable IPv6 family of addresses.

25 14.4.7 Global Data Storage (GDS) attributes

26 These attributes are used to define the behavior of the GDS used to manage key/value pairs by
27 passing them into the relevant initialization API - thus, they are not typically accessed via the
28 `PMIx_Get` API.

29 **PMIX_GDS_MODULE** "pmix.gds.mod" (char*)

30 Comma-delimited string of desired modules.

31 14.4.8 General process-level attributes

32 These attributes are used to define process attributes and are referenced by their process rank.

33 **PMIX_CPUSSET** "pmix.cpuset" (char*)

34 hwloc² bitmap to be applied to the process upon launch.

35 **PMIX_CREDENTIAL** "pmix.cred" (char*)

36 Security credential assigned to the process.

37 **PMIX_SPAWNED** "pmix.spawned" (bool)

²<https://www.open-mpi.org/projects/hwloc/>

1 true if this process resulted from a call to [PMIx_Spawn](#).
2 **PMIX_ARCH** "pmix.arch" (uint32_t)
3 Architecture flag.

4 **14.4.9 Scratch directory attributes**

5 These attributes are used to define an application scratch directory and are referenced using the
6 [PMIX_RANK_WILDCARD](#) rank.

7 **PMIX_TMPDIR** "pmix.tmpdir" (char*)
8 Full path to the top-level temporary directory assigned to the session.
9 **PMIX_NSDIR** "pmix.nsdire" (char*)
10 Full path to the temporary directory assigned to the namespace, under [PMIX_TMPDIR](#).
11 **PMIX_PROCDIR** "pmix.pdire" (char*)
12 Full path to the subdirectory under [PMIX_NSDIR](#) assigned to the process.
13 **PMIX_TDIR_RMCLEAN** "pmix.tdire.rmclean" (bool)
14 Resource Manager will clean session directories

15 **14.4.10 Relative Rank Descriptive Attributes**

16 These attributes are used to describe information about relative ranks as assigned by the RM, and
17 thus are referenced using the process rank except where noted.

18 **PMIX_CLUSTER_ID** "pmix.clid" (char*)
19 A string name for the cluster this proc is executing on
20 **PMIX_PROCID** "pmix.procid" (pmix_proc_t)
21 Process identifier
22 **PMIX_NAMESPACE** "pmix.namespace" (char*)
23 Namespace of the job.
24 **PMIX_JOBID** "pmix.jobid" (char*)
25 Job identifier assigned by the scheduler.
26 **PMIX_APPNUM** "pmix.appnum" (uint32_t)
27 Application number within the job.
28 **PMIX_RANK** "pmix.rank" (pmix_rank_t)
29 Process rank within the job.
30 **PMIX_GLOBAL_RANK** "pmix.grank" (pmix_rank_t)
31 Process rank spanning across all jobs in this session.
32 **PMIX_APP_RANK** "pmix.apprank" (pmix_rank_t)
33 Process rank within this application.
34 **PMIX_NPROC_OFFSET** "pmix.offset" (pmix_rank_t)
35 Starting global rank of this job - referenced using [PMIX_RANK_WILDCARD](#).
36 **PMIX_LOCAL_RANK** "pmix.lrank" (uint16_t)
37 Local rank on this node within this job.
38 **PMIX_NODE_RANK** "pmix.nrank" (uint16_t)
39 Process rank on this node spanning all jobs.

1 **PMIX_LOCALLDR** "pmix.lldr" (**pmix_rank_t**)
2 Lowest rank on this node within this job - referenced using [PMIX_RANK_WILDCARD](#) .

3 **PMIX_APPLDR** "pmix.aldr" (**pmix_rank_t**)
4 Lowest rank in this application within this job - referenced using [PMIX_RANK_WILDCARD](#) .

5 **PMIX_PROC_PID** "pmix.ppid" (**pid_t**)
6 PID of specified process.

7 **PMIX_SESSION_ID** "pmix.session.id" (**uint32_t**)
8 Session identifier - referenced using [PMIX_RANK_WILDCARD](#) .

9 **PMIX_NODE_LIST** "pmix.nlist" (**char***)
10 Comma-delimited list of nodes running processes for the specified namespace - referenced
11 using [PMIX_RANK_WILDCARD](#) .

12 **PMIX_ALLOCATED_NODELIST** "pmix.alist" (**char***)
13 Comma-delimited list of all nodes in this allocation regardless of whether or not they
14 currently host processes - referenced using [PMIX_RANK_WILDCARD](#) .

15 **PMIX_HOSTNAME** "pmix.hname" (**char***)
16 Name of the host (e.g., where a specified process is running, or a given device is located).

17 **PMIX_NODEID** "pmix.nodeid" (**uint32_t**)
18 Node identifier expressed as the node's index (beginning at zero) in an array of nodes within
19 the active session. The value must be unique and directly correlate to the [PMIX_HOSTNAME](#)
20 of the node - i.e., users can interchangeably reference the same location using either the
21 [PMIX_HOSTNAME](#) or corresponding [PMIX_NODEID](#) .

22 **PMIX_LOCAL_PEERS** "pmix.lpeers" (**char***)
23 Comma-delimited list of ranks on this node within the specified namespace - referenced
24 using [PMIX_RANK_WILDCARD](#) .

25 **PMIX_LOCAL_PROCS** "pmix.lprocs" (**pmix_proc_t array**)
26 Array of [pmix_proc_t](#) of all processes on the specified node - referenced using
27 [PMIX_RANK_WILDCARD](#) .

28 **PMIX_LOCAL_CPUSSETS** "pmix.lcpus" (**char***)
29 Colon-delimited cpusets of local peers within the specified namespace - referenced using
30 [PMIX_RANK_WILDCARD](#) .

31 **PMIX_PROC_URI** "pmix.puri" (**char***)
32 URI containing contact information for a given process.

33 **PMIX_LOCALITY** "pmix.loc" (**uint16_t**)
34 Relative locality of the specified process to the requestor.

35 **PMIX_PARENT_ID** "pmix.parent" (**pmix_proc_t**)
36 Process identifier of the parent process of the calling process.

37 **PMIX_EXIT_CODE** "pmix.exit.code" (**int**)
38 Exit code returned when process terminated

1 14.4.11 Information retrieval attributes

2 The following attributes are used to specify the level of information (e.g., [session](#) , [job](#) , or
3 [application](#)) being requested where ambiguity may exist - see [5.1.5](#) for examples of their use.

4 **PMIX_SESSION_INFO** "pmix.ssn.info" (bool)

5 Return information about the specified session. If information about a session other than the
6 one containing the requesting process is desired, then the attribute array must contain a
7 [PMIX_SESSION_ID](#) attribute identifying the desired target.

8 **PMIX_JOB_INFO** "pmix.job.info" (bool)

9 Return information about the specified job or namespace. If information about a job or
10 namespace other than the one containing the requesting process is desired, then the attribute
11 array must contain a [PMIX_JOBID](#) or [PMIX_NAMESPACE](#) attribute identifying the desired
12 target. Similarly, if information is requested about a job or namespace in a session other than
13 the one containing the requesting process, then an attribute identifying the target session
14 must be provided.

15 **PMIX_APP_INFO** "pmix.app.info" (bool)

16 Return information about the specified application. If information about an application other
17 than the one containing the requesting process is desired, then the attribute array must
18 contain a [PMIX_APPNUM](#) attribute identifying the desired target. Similarly, if information
19 is requested about an application in a job or session other than the one containing the
20 requesting process, then attributes identifying the target job and/or session must be provided.

21 **PMIX_NODE_INFO** "pmix.node.info" (bool)

22 Return information about the specified node. If information about a node other than the one
23 containing the requesting process is desired, then the attribute array must contain either the
24 [PMIX_NODEID](#) or [PMIX_HOSTNAME](#) attribute identifying the desired target.

25 14.4.12 Information storage attributes

26 The following attributes are used to assemble information by its level (e.g., [session](#) , [job](#) , or
27 [application](#)) for storage where ambiguity may exist - see [11.2.3.1](#) for examples of their use.

28 **PMIX_SESSION_INFO_ARRAY** "pmix.ssn.arr" ([pmix_data_array_t](#))

29 Provide an array of [pmix_info_t](#) containing session-level information. The
30 [PMIX_SESSION_ID](#) attribute is required to be included in the array.

31 **PMIX_JOB_INFO_ARRAY** "pmix.job.arr" ([pmix_data_array_t](#))

32 Provide an array of [pmix_info_t](#) containing job-level information. The
33 [PMIX_SESSION_ID](#) attribute of the [session](#) containing the [job](#) is required to be
34 included in the array whenever the PMIx server library may host multiple sessions (e.g.,
35 when executing with a host RM daemon). As information is registered one job (aka
36 namespace) at a time via the [PMIx_server_register_namespace](#) API, there is no
37 requirement that the array contain either the [PMIX_NAMESPACE](#) or [PMIX_JOBID](#) attributes
38 when used in that context (though either or both of them may be included). At least one of
39 the job identifiers must be provided in all other contexts where the job being referenced is
40 ambiguous.

1 **PMIX_APP_INFO_ARRAY** "pmix.app.arr" (pmix_data_array_t)

2 Provide an array of **pmix_info_t** containing app-level information. The **PMIX_NAMESPACE**
3 or **PMIX_JOBID** attributes of the **job** containing the application, plus its **PMIX_APPNUM**
4 attribute, are must to be included in the array when the array is *not* included as part of a call
5 to **PMIx_server_register_namespace** - i.e., when the job containing the application is
6 ambiguous. The job identification is otherwise optional.

7 **PMIX_NODE_INFO_ARRAY** "pmix.node.arr" (pmix_data_array_t)

8 Provide an array of **pmix_info_t** containing node-level information. At a minimum,
9 either the **PMIX_NODEID** or **PMIX_HOSTNAME** attribute is required to be included in the
10 array, though both may be included.

11 Note that these assemblages can be used hierarchically:

- 12 • a **PMIX_JOB_INFO_ARRAY** might contain multiple **PMIX_APP_INFO_ARRAY** elements,
13 each describing values for a specific application within the job
- 14 • a **PMIX_JOB_INFO_ARRAY** could contain a **PMIX_NODE_INFO_ARRAY** for each node
15 hosting processes from that job, each array describing job-level values for that node
- 16 • a **PMIX_SESSION_INFO_ARRAY** might contain multiple **PMIX_JOB_INFO_ARRAY**
17 elements, each describing a job executing within the session. Each job array could, in turn,
18 contain both application and node arrays, thus providing a complete picture of the active
19 operations within the allocation

▼ Advice to PMIx library implementers ▼

20 PMIx implementations must be capable of properly parsing and storing any hierarchical depth of
21 information arrays. The resulting stored values are must to be accessible via both **PMIx_Get** and
22 **PMIx_Query_info_nb** APIs, assuming appropriate directives are provided by the caller.

23 14.4.13 Size information attributes

24 These attributes are used to describe the size of various dimensions of the PMIx universe - all are
25 referenced using **PMIX_RANK_WILDCARD** .

26 **PMIX_UNIV_SIZE** "pmix.univ.size" (uint32_t)

27 Number of allocated slots in a session - each slot may or may not be occupied by an
28 executing process. Note that this attribute is the equivalent to the combination of
29 **PMIX_SESSION_INFO_ARRAY** with the **PMIX_MAX_PROCS** entry in the array - it is
30 included in the Standard for historical reasons.

31 **PMIX_JOB_SIZE** "pmix.job.size" (uint32_t)

32 Total number of processes in this job across all contained applications. Note that this value
33 can be different from **PMIX_MAX_PROCS** . For example, users may choose to subdivide an
34 allocation (running several jobs in parallel within it), and dynamic programming models may
35 support adding and removing processes from a running **job** on-the-fly. In the latter case,
36 PMIx events must be used to notify processes within the job that the job size has changed.

1 **PMIX_JOB_NUM_APPS** "pmix.job.napps" (uint32_t)
 2 Number of applications in this job.
 3 **PMIX_APP_SIZE** "pmix.app.size" (uint32_t)
 4 Number of processes in this application.
 5 **PMIX_LOCAL_SIZE** "pmix.local.size" (uint32_t)
 6 Number of processes in this job or application on this node.
 7 **PMIX_NODE_SIZE** "pmix.node.size" (uint32_t)
 8 Number of processes across all jobs on this node.
 9 **PMIX_MAX_PROCS** "pmix.max.size" (uint32_t)
 10 Maximum number of processes that can be executed in this context (session, namespace,
 11 application, or node). Typically, this is a constraint imposed by a scheduler or by user
 12 settings in a hostfile or other resource description.
 13 **PMIX_NUM_SLOTS** "pmix.num.slots" (uint32_t)
 14 Number of slots allocated in this context (session, namespace, application, or node). Note
 15 that this attribute is the equivalent to **PMIX_MAX_PROCS** used in the corresponding
 16 context - it is included in the Standard for historical reasons.
 17 **PMIX_NUM_NODES** "pmix.num.nodes" (uint32_t)
 18 Number of nodes in this session, or that are currently executing processes from the
 19 associated namespace or application.

20 14.4.14 Memory information attributes

21 These attributes are used to describe memory available and used in the system - all are referenced
 22 using **PMIX_RANK_WILDCARD**.

23 **PMIX_AVAIL_PHYS_MEMORY** "pmix.pmem" (uint64_t)
 24 Total available physical memory on this node.
 25 **PMIX_DAEMON_MEMORY** "pmix.dmn.mem" (float)
 26 Megabytes of memory currently used by the RM daemon.
 27 **PMIX_CLIENT_AVG_MEMORY** "pmix.cl.mem.avg" (float)
 28 Average Megabytes of memory used by client processes.

29 14.4.15 Topology information attributes

30 These attributes are used to describe topology information in the PMIx universe - all are referenced
 31 using **PMIX_RANK_WILDCARD** except where noted.

32 **PMIX_LOCAL_TOPO** "pmix.ltopo" (char*)
 33 XML representation of local node topology.
 34 **PMIX_TOPOLOGY** "pmix.topo" (hwloc_topology_t)
 35 Pointer to the PMIx client's internal hwloc topology object.
 36 **PMIX_TOPOLOGY_XML** "pmix.topo.xml" (char*)
 37 XML-based description of topology
 38 **PMIX_TOPOLOGY_FILE** "pmix.topo.file" (char*)
 39 Full path to file containing XML topology description

1 **PMIX_TOPOLOGY_SIGNATURE** "pmix.toposig" (char*)
 2 Topology signature string.

3 **PMIX_LOCALITY_STRING** "pmix.locstr" (char*)
 4 String describing a process's bound location - referenced using the process's rank. The string
 5 is of the form:
 6 **NM%s:SK%s:L3%s:L2%s:L1%s:CR%s:HT%s**
 7 Where the %s is replaced with an integer index or inclusive range for hwloc. **NM** identifies
 8 the numa node(s). **SK** identifies the socket(s). **L3** identifies the L3 cache(s). **L2** identifies the
 9 L2 cache(s). **L1** identifies the L1 cache(s). **CR** identifies the cores(s). **HT** identifies the
 10 hardware thread(s). If your architecture does not have the specified hardware designation
 11 then it can be omitted from the signature.
 12 For example: **NM0:SK0:L30-4:L20-4:L10-4:CR0-4:HT0-39**.
 13 This means numa node 0, socket 0, L3 caches 0, 1, 2, 3, 4, L2 caches 0-4, L1 caches
 14 0-4, cores 0, 1, 2, 3, 4, and hardware threads 0-39.

15 **PMIX_HWLOC_SHMEM_ADDR** "pmix.hwlocaddr" (size_t)
 16 Address of the HWLOC shared memory segment.

17 **PMIX_HWLOC_SHMEM_SIZE** "pmix.hwlocsize" (size_t)
 18 Size of the HWLOC shared memory segment.

19 **PMIX_HWLOC_SHMEM_FILE** "pmix.hwlocfile" (char*)
 20 Path to the HWLOC shared memory file.

21 **PMIX_HWLOC_XML_V1** "pmix.hwlocxml1" (char*)
 22 XML representation of local topology using HWLOC's v1.x format.

23 **PMIX_HWLOC_XML_V2** "pmix.hwlocxml2" (char*)
 24 XML representation of local topology using HWLOC's v2.x format.

25 **PMIX_HWLOC_SHARE_TOPO** "pmix.hwlocsh" (bool)
 26 Share the HWLOC topology via shared memory

27 **PMIX_HWLOC_HOLE_KIND** "pmix.hwlocholek" (char*)
 28 Kind of VM "hole" HWLOC should use for shared memory

29 14.4.16 Request-related attributes

30 These attributes are used to influence the behavior of various PMIx operations - they do not
 31 represent values accessed using the [PMIx_Get](#) API.

32 **PMIX_COLLECT_DATA** "pmix.collect" (bool)
 33 Collect data and return it at the end of the operation.

34 **PMIX_TIMEOUT** "pmix.timeout" (int)
 35 Time in seconds before the specified operation should time out (0 indicating infinite) in
 36 error. The timeout parameter can help avoid "hangs" due to programming errors that prevent
 37 the target process from ever exposing its data.

38 **PMIX_IMMEDIATE** "pmix.immediate" (bool)
 39 Specified operation should immediately return an error from the PMIx server if the requested
 40 data cannot be found - do not request it from the host RM.

41 **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).

3 **PMIX_COLLECTIVE_ALGO** "pmix.calgo" (char*)

4 Comma-delimited list of algorithms to use for the collective operation. PMIx does not
5 impose any requirements on a host environment's collective algorithms. Thus, the
6 acceptable values for this attribute will be environment-dependent - users are encouraged to
7 check their host environment for supported values.

8 **PMIX_COLLECTIVE_ALGO_REQD** "pmix.calreqd" (bool)

9 If true, indicates that the requested choice of algorithm is mandatory.

10 **PMIX_NOTIFY_COMPLETION** "pmix.notecomp" (bool)

11 Notify the parent process upon termination of child job.

12 **PMIX_RANGE** "pmix.range" (pmix_data_range_t)

13 Value for calls to publish/lookup/unpublish or for monitoring event notifications.

14 **PMIX_PERSISTENCE** "pmix.persist" (pmix_persistence_t)

15 Value for calls to [PMIx_Publish](#).

16 **PMIX_DATA_SCOPE** "pmix.scope" (pmix_scope_t)

17 Scope of the data to be found in a [PMIx_Get](#) call.

18 **PMIX_OPTIONAL** "pmix.optional" (bool)

19 Look only in the client's local data store for the requested value - do not request data from
20 the PMIx server if not found.

21 **PMIX_GET_STATIC_VALUES** "pmix.get.static" (bool)

22 Request that any pointers in the returned value point directly to values in the key-value store

23 **PMIX_EMBED_BARRIER** "pmix.embed.barrier" (bool)

24 Execute a blocking fence operation before executing the specified operation. For example,
25 [PMIx_Finalize](#) does not include an internal barrier operation by default. This attribute
26 would direct [PMIx_Finalize](#) to execute a barrier as part of the finalize operation.

27 **PMIX_JOB_TERM_STATUS** "pmix.job.term.status" (pmix_status_t)

28 Status returned by job upon its termination. The status will be communicated as part of a
29 PMIx event payload provided by the host environment upon termination of a job. Note that
30 generation of the [PMIX_ERR_JOB_TERMINATED](#) event is optional and host environments
31 may choose to provide it only upon request.

32 **PMIX_PROC_STATE_STATUS** "pmix.proc.state" (pmix_proc_state_t)

33 State of the specified process as of the last report - may not be the actual current state based
34 on update rate.

35 **PMIX_PROC_TERM_STATUS** "pmix.proc.term.status" (pmix_status_t)

36 Status returned by a process upon its termination. The status will be communicated as part
37 of a PMIx event payload provided by the host environment upon termination of a process.
38 Note that generation of the [PMIX_PROC_TERMINATED](#) event is optional and host
39 environments may choose to provide it only upon request.

1 14.4.17 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 Array of process data. Starts with rank, then contains more data.
- 10 **PMIX_NODE_MAP** "pmix.nmap" (char*)
11 Regular expression of nodes - see [11.2.3.1](#) for an explanation of its generation.
- 12 **PMIX_PROC_MAP** "pmix.pmap" (char*)
13 Regular expression describing processes on each node - see [11.2.3.1](#) 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 14.4.18 Server-to-Client attributes

22 Attributes used internally to communicate data from the PMIx server to the PMIx client - they do
23 not represent values accessed using the [PMIx_Get](#) API.

- 24 **PMIX_PROC_BLOB** "pmix.pblob" (pmix_byte_object_t)
25 Packed blob of process data.
- 26 **PMIX_MAP_BLOB** "pmix.mblob" (pmix_byte_object_t)
27 Packed blob of process location.

28 14.4.19 Event handler registration and notification 29 attributes

30 Attributes to support event registration and notification - they are values passed to the event
31 registration and notification APIs and therefore are not accessed using the [PMIx_Get](#) API.

- 32 **PMIX_EVENT_HDLR_NAME** "pmix.evname" (char*)
33 String name identifying this handler.
- 34 **PMIX_EVENT_HDLR_FIRST** "pmix.evfirst" (bool)
35 Invoke this event handler before any other handlers.
- 36 **PMIX_EVENT_HDLR_LAST** "pmix.evlast" (bool)
37 Invoke this event handler after all other handlers have been called.
- 38 **PMIX_EVENT_HDLR_FIRST_IN_CATEGORY** "pmix.evfirstcat" (bool)

1 Invoke this event handler before any other handlers in this category.

2 **PMIX_EVENT_HDLR_LAST_IN_CATEGORY** "pmix.evlastcat" (bool)

3 Invoke this event handler after all other handlers in this category have been called.

4 **PMIX_EVENT_HDLR_BEFORE** "pmix.evbefore" (char*)

5 Put this event handler immediately before the one specified in the (char*) value.

6 **PMIX_EVENT_HDLR_AFTER** "pmix.evafter" (char*)

7 Put this event handler immediately after the one specified in the (char*) value.

8 **PMIX_EVENT_HDLR_PREPEND** "pmix.evprepend" (bool)

9 Prepend this handler to the precedence list within its category.

10 **PMIX_EVENT_HDLR_APPEND** "pmix.evappend" (bool)

11 Append this handler to the precedence list within its category.

12 **PMIX_EVENT_CUSTOM_RANGE** "pmix.evrange" (pmix_data_array_t*)

13 Array of [pmix_proc_t](#) defining range of event notification.

14 **PMIX_EVENT_AFFECTED_PROC** "pmix.evproc" (pmix_proc_t)

15 The single process that was affected.

16 **PMIX_EVENT_AFFECTED_PROCS** "pmix.evaffected" (pmix_data_array_t*)

17 Array of [pmix_proc_t](#) defining affected processes.

18 **PMIX_EVENT_NON_DEFAULT** "pmix.evnondef" (bool)

19 Event is not to be delivered to default event handlers.

20 **PMIX_EVENT_RETURN_OBJECT** "pmix.evobject" (void *)

21 Object to be returned whenever the registered callback function **cbfunc** is invoked. The

22 object will only be returned to the process that registered it.

23 **PMIX_EVENT_DO_NOT_CACHE** "pmix.evnocache" (bool)

24 Instruct the PMIx server not to cache the event.

25 **PMIX_EVENT_SILENT_TERMINATION** "pmix.evsilentterm" (bool)

26 Do not generate an event when this job normally terminates.

27 **PMIX_EVENT_PROXY** "pmix.evproxy" (pmix_proc_t*)

28 PMIx server that sourced the event

29 **PMIX_EVENT_TEXT_MESSAGE** "pmix.evtext" (char*)

30 Text message suitable for output by recipient - e.g., describing the cause of the event

31 14.4.20 Fault tolerance attributes

32 Attributes to support fault tolerance behaviors - they are values passed to the event notification API

33 and therefore are not accessed using the [PMIx_Get](#) API.

34 **PMIX_EVENT_TERMINATE_SESSION** "pmix.evterm.sess" (bool)

35 The RM intends to terminate this session.

36 **PMIX_EVENT_TERMINATE_JOB** "pmix.evterm.job" (bool)

37 The RM intends to terminate this job.

38 **PMIX_EVENT_TERMINATE_NODE** "pmix.evterm.node" (bool)

39 The RM intends to terminate all processes on this node.

40 **PMIX_EVENT_TERMINATE_PROC** "pmix.evterm.proc" (bool)

41 The RM intends to terminate just this process.

1 **PMIX_EVENT_ACTION_TIMEOUT** "pmix.evtimeout" (int)
 2 The time in seconds before the RM will execute error response.
 3 **PMIX_EVENT_NO_TERMINATION** "pmix.evnoterm" (bool)
 4 Indicates that the handler has satisfactorily handled the event and believes termination of the
 5 application is not required.
 6 **PMIX_EVENT_WANT_TERMINATION** "pmix.evterm" (bool)
 7 Indicates that the handler has determined that the application should be terminated

8 14.4.21 Spawn attributes

9 Attributes used to describe **PMIx_Spawn** behavior - they are values passed to the **PMIx_Spawn**
 10 API and therefore are not accessed using the **PMIx_Get** API when used in that context. However,
 11 some of the attributes defined in this section can be provided by the host environment for other
 12 purposes - e.g., the environment might provide the **PMIX_MAPPER** attribute in the job-related
 13 information so that an application can use **PMIx_Get** to discover the layout algorithm used for
 14 determining process locations. Multi-use attributes and their respective access reference rank are
 15 denoted below.

16 **PMIX_PERSONALITY** "pmix.pers" (char*)
 17 Name of personality to use.
 18 **PMIX_HOST** "pmix.host" (char*)
 19 Comma-delimited list of hosts to use for spawned processes.
 20 **PMIX_HOSTFILE** "pmix.hostfile" (char*)
 21 Hostfile to use for spawned processes.
 22 **PMIX_ADD_HOST** "pmix.addhost" (char*)
 23 Comma-delimited list of hosts to add to the allocation.
 24 **PMIX_ADD_HOSTFILE** "pmix.addhostfile" (char*)
 25 Hostfile listing hosts to add to existing allocation.
 26 **PMIX_PREFIX** "pmix.prefix" (char*)
 27 Prefix to use for starting spawned processes.
 28 **PMIX_WDIR** "pmix.wdir" (char*)
 29 Working directory for spawned processes.
 30 **PMIX_MAPPER** "pmix.mapper" (char*)
 31 Mapping mechanism to use for placing spawned processes - when accessed using
 32 **PMIx_Get**, use the **PMIX_RANK_WILDCARD** value for the rank to discover the mapping
 33 mechanism used for the provided namespace.
 34 **PMIX_DISPLAY_MAP** "pmix.dispmap" (bool)
 35 Display process mapping upon spawn.
 36 **PMIX_PPR** "pmix.ppr" (char*)
 37 Number of processes to spawn on each identified resource.
 38 **PMIX_MAPBY** "pmix.mapby" (char*)
 39 Process mapping policy - when accessed using **PMIx_Get**, use the
 40 **PMIX_RANK_WILDCARD** value for the rank to discover the mapping policy used for the
 41 provided namespace

1 **PMIX_RANKBY** "pmix.rankby" (char*)
2 Process ranking policy - when accessed using **PMIx_Get** , use the
3 **PMIX_RANK_WILDCARD** value for the rank to discover the ranking algorithm used for the
4 provided namespace

5 **PMIX_BINDTO** "pmix.bindto" (char*)
6 Process binding policy - when accessed using **PMIx_Get** , use the
7 **PMIX_RANK_WILDCARD** value for the rank to discover the binding policy used for the
8 provided namespace

9 **PMIX_PRELOAD_BIN** "pmix.preloadbin" (bool)
10 Preload binaries onto nodes.

11 **PMIX_PRELOAD_FILES** "pmix.preloadfiles" (char*)
12 Comma-delimited list of files to pre-position on nodes.

13 **PMIX_NON_PMI** "pmix.nonpmi" (bool)
14 Spawned processes will not call **PMIx_Init** .

15 **PMIX_STDIN_TGT** "pmix.stdin" (uint32_t)
16 Spawned process rank that is to receive **stdin**.

17 **PMIX_FWD_STDIN** "pmix.fwd.stdin" (bool)
18 Forward this process's **stdin** to the designated process.

19 **PMIX_FWD_STDOUT** "pmix.fwd.stdout" (bool)
20 Forward **stdout** from spawned processes to this process.

21 **PMIX_FWD_STDERR** "pmix.fwd.stderr" (bool)
22 Forward **stderr** from spawned processes to this process.

23 **PMIX_FWD_STDDIAG** "pmix.fwd.stddiag" (bool)
24 If a diagnostic channel exists, forward any output on it from the spawned processes to this
25 process (typically used by a tool)

26 **PMIX_DEBUGGER_DAEMONS** "pmix.debugger" (bool)
27 Spawned application consists of debugger daemons.

28 **PMIX_COSPAWN_APP** "pmix.cospawn" (bool)
29 Designated application is to be spawned as a disconnected job. Meaning that it is not part of
30 the "comm_world" of the parent process.

31 **PMIX_SET_SESSION_CWD** "pmix.ssn cwd" (bool)
32 Set the application's current working directory to the session working directory assigned by
33 the RM - when accessed using **PMIx_Get** , use the **PMIX_RANK_WILDCARD** value for
34 the rank to discover the session working directory assigned to the provided namespace

35 **PMIX_TAG_OUTPUT** "pmix.tagout" (bool)
36 Tag application output with the identity of the source process.

37 **PMIX_TIMESTAMP_OUTPUT** "pmix.tsout" (bool)
38 Timestamp output from applications.

39 **PMIX_MERGE_STDERR_STDOUT** "pmix.mergeerrout" (bool)
40 Merge **stdout** and **stderr** streams from application processes.

41 **PMIX_OUTPUT_TO_FILE** "pmix.outfile" (char*)
42 Output application output to the specified file.

43 **PMIX_INDEX_ARGV** "pmix.indxargv" (bool)

1 Mark the `argv` with the rank of the process.

2 **PMIX_CPUS_PER_PROC** "`pmix.cpusperproc`" (`uint32_t`)
3 Number of cpus to assign to each rank - when accessed using `PMIx_Get`, use the
4 `PMIX_RANK_WILDCARD` value for the rank to discover the cpus/process assigned to the
5 provided namespace
6 **PMIX_NO_PROCS_ON_HEAD** "`pmix.nolocal`" (`bool`)
7 Do not place processes on the head node.
8 **PMIX_NO_OVERSUBSCRIBE** "`pmix.noover`" (`bool`)
9 Do not oversubscribe the cpus.
10 **PMIX_REPORT_BINDINGS** "`pmix.repbinding`" (`bool`)
11 Report bindings of the individual processes.
12 **PMIX_CPU_LIST** "`pmix.cpulist`" (`char*`)
13 List of cpus to use for this job - when accessed using `PMIx_Get`, use the
14 `PMIX_RANK_WILDCARD` value for the rank to discover the cpu list used for the provided
15 namespace
16 **PMIX_JOB_RECOVERABLE** "`pmix.recover`" (`bool`)
17 Application supports recoverable operations.
18 **PMIX_JOB_CONTINUOUS** "`pmix.continuous`" (`bool`)
19 Application is continuous, all failed processes should be immediately restarted.
20 **PMIX_MAX_RESTARTS** "`pmix.maxrestarts`" (`uint32_t`)
21 Maximum number of times to restart a job - when accessed using `PMIx_Get`, use the
22 `PMIX_RANK_WILDCARD` value for the rank to discover the max restarts for the provided
23 namespace
24 **PMIX_SPAWN_TOOL** "`pmix.spwn.tool`" (`bool`)
25 Indicate that the job being spawned is a tool

26 14.4.22 Query attributes

27 Attributes used to describe `PMIx_Query_info_nb` behavior - these are values passed to the
28 `PMIx_Query_info_nb` API and therefore are not passed to the `PMIx_Get` API.

29 **PMIX_QUERY_REFRESH_CACHE** "`pmix.qry.rfsh`" (`bool`)
30 Retrieve updated information from server.
31 **PMIX_QUERY_NAMESPACES** "`pmix.qry.ns`" (`char*`)
32 Request a comma-delimited list of active namespaces.
33 **PMIX_QUERY_JOB_STATUS** "`pmix.qry.jst`" (`pmix_status_t`)
34 Status of a specified, currently executing job.
35 **PMIX_QUERY_QUEUE_LIST** "`pmix.qry.qlst`" (`char*`)
36 Request a comma-delimited list of scheduler queues.
37 **PMIX_QUERY_QUEUE_STATUS** "`pmix.qry.qst`" (TBD)
38 Status of a specified scheduler queue.
39 **PMIX_QUERY_PROC_TABLE** "`pmix.qry.ptable`" (`char*`)
40 Input namespace of the job whose information is being requested returns (
41 `pmix_data_array_t`) an array of `pmix_proc_info_t`.

1 **PMIX_QUERY_LOCAL_PROC_TABLE** "pmix.qry.lptable" (char*)
2 Input namespace of the job whose information is being requested returns (
3 **pmix_data_array_t**) an array of **pmix_proc_info_t** for processes in job on same
4 node.

5 **PMIX_QUERY_AUTHORIZATIONS** "pmix.qry.auths" (bool)
6 Return operations the PMIx tool is authorized to perform.

7 **PMIX_QUERY_SPAWN_SUPPORT** "pmix.qry.spawn" (bool)
8 Return a comma-delimited list of supported spawn attributes.

9 **PMIX_QUERY_DEBUG_SUPPORT** "pmix.qry.debug" (bool)
10 Return a comma-delimited list of supported debug attributes.

11 **PMIX_QUERY_MEMORY_USAGE** "pmix.qry.mem" (bool)
12 Return information on memory usage for the processes indicated in the qualifiers.

13 **PMIX_QUERY_LOCAL_ONLY** "pmix.qry.local" (bool)
14 Constrain the query to local information only.

15 **PMIX_QUERY_REPORT_AVG** "pmix.qry.avg" (bool)
16 Report only average values for sampled information.

17 **PMIX_QUERY_REPORT_MINMAX** "pmix.qry.minmax" (bool)
18 Report minimum and maximum values.

19 **PMIX_QUERY_ALLOC_STATUS** "pmix.query.alloc" (char*)
20 String identifier of the allocation whose status is being requested.

21 **PMIX_TIME_REMAINING** "pmix.time.remaining" (char*)
22 Query number of seconds (**uint32_t**) remaining in allocation for the specified namespace.

23 **PMIX_QUERY_ATTRIBUTE_SUPPORT** "pmix.qry.attrs" (bool)
24 Query list of supported attributes for specified APIs

25 **PMIX_QUERY_NUM_PSETS** "pmix.qry.psetnum" (**size_t**)
26 Return the number of psets defined in the specified range (defaults to session).

27 **PMIX_QUERY_PSET_NAMES** "pmix.qry.psets" (char*)
28 Return a comma-delimited list of the names of the psets defined in the specified range
29 (defaults to session).

30 14.4.23 Log attributes

31 Attributes used to describe **PMIx_Log_nb** behavior - these are values passed to the
32 **PMIx_Log_nb** API and therefore are not accessed using the **PMIx_Get** API.

33 **PMIX_LOG_SOURCE** "pmix.log.source" (**pmix_proc_t***)
34 ID of source of the log request

35 **PMIX_LOG_STDERR** "pmix.log.stderr" (char*)
36 Log string to **stderr**.

37 **PMIX_LOG_STDOUT** "pmix.log.stdout" (char*)
38 Log string to **stdout**.

39 **PMIX_LOG_SYSLOG** "pmix.log.syslog" (char*)
40 Log data to syslog. Defaults to **ERROR** priority. Will log to global syslog if available,
41 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_TIMESTAMP "pmix.log.tstmp" (time_t)
8      Timestamp for log report
9  PMIX_LOG_GENERATE_TIMESTAMP "pmix.log.gtstmp" (bool)
10     Generate timestamp for log
11 PMIX_LOG_TAG_OUTPUT "pmix.log.tag" (bool)
12     Label the output stream with the channel name (e.g., "stdout")
13 PMIX_LOG_TIMESTAMP_OUTPUT "pmix.log.tsout" (bool)
14     Print timestamp in output string
15 PMIX_LOG_XML_OUTPUT "pmix.log.xml" (bool)
16     Print the output stream in XML format
17 PMIX_LOG_ONCE "pmix.log.once" (bool)
18     Only log this once with whichever channel can first support it, taking the channels in priority
19     order
20 PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
21     Message blob to be sent somewhere.
22 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
23     Log via email based on pmix\_info\_t containing directives.
24 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
25     Comma-delimited list of email addresses that are to receive the message.
26 PMIX_LOG_EMAIL_SENDER_ADDR "pmix.log.emfaddr" (char*)
27     Return email address of sender
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_EMAIL_SERVER "pmix.log.esrvr" (char*)
33     Hostname (or IP address) of estmp server
34 PMIX_LOG_EMAIL_SRVR_PORT "pmix.log.esrvrprt" (int32_t)
35     Port the email server is listening to
36 PMIX_LOG_GLOBAL_DATASTORE "pmix.log.gstore" (bool)
37     Store the log data in a global data store (e.g., database)
38 PMIX_LOG_JOB_RECORD "pmix.log.jrec" (bool)
39     Log the provided information to the host environment's job record

```

1 14.4.24 Debugger attributes

2 Attributes used to assist debuggers - these are values that can be passed to the `PMIx_Spawn` or
3 `PMIx_Init` APIs. Some may be accessed using the `PMIx_Get` API with the
4 `PMIX_RANK_WILDCARD` rank.

5 `PMIX_DEBUG_STOP_ON_EXEC` "pmix.dbg.exec" (bool)

6 Passed to `PMIx_Spawn` to indicate that the specified application is being spawned under
7 debugger, and that the launcher is to pause the resulting application processes on first
8 instruction for debugger attach.

9 `PMIX_DEBUG_STOP_IN_INIT` "pmix.dbg.init" (bool)

10 Passed to `PMIx_Spawn` to indicate that the specified application is being spawned under
11 debugger, and that the PMIx client library is to pause the resulting application processes
12 during `PMIx_Init` until debugger attach and release.

13 `PMIX_DEBUG_WAIT_FOR_NOTIFY` "pmix.dbg.notify" (bool)

14 Passed to `PMIx_Spawn` to indicate that the specified application is being spawned under
15 debugger, and that the resulting application processes are to pause at some
16 application-determined location until debugger attach and release.

17 `PMIX_DEBUG_JOB` "pmix.dbg.job" (char*)

18 Namespace of the job to be debugged - provided to the debugger upon launch.

19 `PMIX_DEBUG_WAITING_FOR_NOTIFY` "pmix.dbg.waiting" (bool)

20 Job to be debugged is waiting for a release - this is not a value accessed using the
21 `PMIx_Get` API.

22 `PMIX_DEBUG_JOB_DIRECTIVES` "pmix.dbg.jdirs" (pmix_data_array_t*)

23 Array of job-level directives

24 `PMIX_DEBUG_APP_DIRECTIVES` "pmix.dbg.adirs" (pmix_data_array_t*)

25 Array of app-level directives

26 14.4.25 Resource manager attributes

27 Attributes used to describe the RM - these are values assigned by the host environment and accessed
28 using the `PMIx_Get` API. The value of the provided namespace is unimportant but should be
29 given as the namespace of the requesting process and a rank of `PMIX_RANK_WILDCARD` used to
30 indicate that the information will be found with the job-level information.

31 `PMIX_RM_NAME` "pmix.rm.name" (char*)

32 String name of the RM.

33 `PMIX_RM_VERSION` "pmix.rm.version" (char*)

34 RM version string.

1 14.4.26 Environment variable attributes

2 Attributes used to adjust environment variables - these are values passed to the `PMIx_Spawn` API
3 and are not accessed using the `PMIx_Get` API.

4 `PMIX_SET_ENVAR "pmix.envar.set" (pmix_envar_t*)`

5 Set the envar to the given value, overwriting any pre-existing one

6 `PMIX_UNSET_ENVAR "pmix.envar.unset" (char*)`

7 Unset the environment variable specified in the string.

8 `PMIX_ADD_ENVAR "pmix.envar.add" (pmix_envar_t*)`

9 Add the environment variable, but do not overwrite any pre-existing one

10 `PMIX_PREPEND_ENVAR "pmix.envar.prepend" (pmix_envar_t*)`

11 Prepend the given value to the specified environmental value using the given separator
12 character, creating the variable if it doesn't already exist

13 `PMIX_APPEND_ENVAR "pmix.envar.append" (pmix_envar_t*)`

14 Append the given value to the specified environmental value using the given separator
15 character, creating the variable if it doesn't already exist

16 14.4.27 Job Allocation attributes

17 Attributes used to describe the job allocation - these are values passed to and/or returned by the
18 `PMIx_Allocation_request_nb` and `PMIx_Allocation_request` APIs and are not
19 accessed using the `PMIx_Get` API

20 `PMIX_ALLOC_REQ_ID "pmix.alloc.reqid" (char*)`

21 User-provided string identifier for this allocation request which can later be used to query
22 status of the request.

23 `PMIX_ALLOC_ID "pmix.alloc.id" (char*)`

24 A string identifier (provided by the host environment) for the resulting allocation which can
25 later be used to reference the allocated resources in, for example, a call to `PMIx_Spawn`.

26 `PMIX_ALLOC_NUM_NODES "pmix.alloc.nnodes" (uint64_t)`

27 The number of nodes.

28 `PMIX_ALLOC_NODE_LIST "pmix.alloc.nlist" (char*)`

29 Regular expression of the specific nodes.

30 `PMIX_ALLOC_NUM_CPUS "pmix.alloc.ncpus" (uint64_t)`

31 Number of cpus.

32 `PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)`

33 Regular expression of the number of cpus for each node.

34 `PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)`

35 Regular expression of the specific cpus indicating the cpus involved.

36 `PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)`

37 Number of Megabytes.

38 `PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)`

39 Changed to `PMIX_ALLOC_FABRIC`

40 `PMIX_ALLOC_FABRIC "pmix.alloc.net" (array)`

1 Array of `pmix_info_t` describing requested fabric resources. This must include at least:
2 **PMIX_ALLOC_FABRIC_ID**, **PMIX_ALLOC_FABRIC_TYPE**, and
3 **PMIX_ALLOC_FABRIC_ENDPTS**, plus whatever other descriptors are desired.

4 **PMIX_ALLOC_NETWORK_ID** "pmix.alloc.netid" (char*)
5 Changed to **PMIX_ALLOC_FABRIC_ID**

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_BANDWIDTH** "pmix.alloc.bw" (float)
21 Mbits/sec.

22 **PMIX_ALLOC_NETWORK_QOS** "pmix.alloc.netqos" (char*)
23 Changed to **PMIX_ALLOC_FABRIC_QOS**

24 **PMIX_ALLOC_FABRIC_QOS** "pmix.alloc.netqos" (char*)
25 Quality of service level.

26 **PMIX_ALLOC_TIME** "pmix.alloc.time" (uint32_t)
27 Time in seconds.

28 **PMIX_ALLOC_NETWORK_TYPE** "pmix.alloc.nettype" (char*)
29 Changed to **PMIX_ALLOC_FABRIC_TYPE**

30 **PMIX_ALLOC_FABRIC_TYPE** "pmix.alloc.nettype" (char*)
31 Type of desired transport (e.g., "tcp", "udp")

32 **PMIX_ALLOC_NETWORK_PLANE** "pmix.alloc.netplane" (char*)
33 Changed to **PMIX_ALLOC_FABRIC_PLANE**

34 **PMIX_ALLOC_FABRIC_PLANE** "pmix.alloc.netplane" (char*)
35 ID string for the NIC (aka *plane*) to be used for this allocation (e.g., CIDR for Ethernet)

36 **PMIX_ALLOC_NETWORK_ENDPTS** "pmix.alloc.endpts" (size_t)
37 Changed to **PMIX_ALLOC_FABRIC_ENDPTS**

38 **PMIX_ALLOC_FABRIC_ENDPTS** "pmix.alloc.endpts" (size_t)
39 Number of endpoints to allocate per process

40 **PMIX_ALLOC_NETWORK_ENDPTS_NODE** "pmix.alloc.endpts.nd" (size_t)
41 Changed to **PMIX_ALLOC_FABRIC_ENDPTS_NODE**

42 **PMIX_ALLOC_FABRIC_ENDPTS_NODE** "pmix.alloc.endpts.nd" (size_t)
43 Number of endpoints to allocate per node

```

1 PMIX_ALLOC_NETWORK_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
2     Changed to PMIX_ALLOC_FABRIC_SEC_KEY
3 PMIX_ALLOC_FABRIC_SEC_KEY "pmix.alloc.nsec" (pmix_byte_object_t)
4     Fabric security key

```

5 14.4.28 Job control attributes

6 Attributes used to request control operations on an executing application - these are values passed
7 to the **PMIx_Job_control_nb** API and are not accessed using the **PMIx_Get** API.

```

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_CANCEL "pmix.jctrl.cancel" (char*)
17     Cancel the specified request - the provided request ID must match the
18     PMIX_JOB_CTRL_ID provided to a previous call to PMIx_Job_control . An ID of
19     NULL implies cancel all requests from this requestor.
20 PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
21     Forcibly terminate the specified processes and cleanup.
22 PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
23     Restart the specified processes using the given checkpoint ID.
24 PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
25     Checkpoint the specified processes and assign the given ID to it.
26 PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
27     Use event notification to trigger a process checkpoint.
28 PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
29     Use the given signal to trigger a process checkpoint.
30 PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
31     Time in seconds to wait for a checkpoint to complete.
32 PMIX_JOB_CTRL_CHECKPOINT_METHOD
33     "pmix.jctrl.ckmethod" (pmix_data_array_t)
34     Array of pmix_info_t declaring each method and value supported by this application.
35 PMIX_JOB_CTRL_SIGNAL "pmix.jctrl.sig" (int)
36     Send given signal to specified processes.
37 PMIX_JOB_CTRL_PROVISION "pmix.jctrl.pvn" (char*)
38     Regular expression identifying nodes that are to be provisioned.
39 PMIX_JOB_CTRL_PROVISION_IMAGE "pmix.jctrl.pvnmng" (char*)
40     Name of the image that is to be provisioned.
41 PMIX_JOB_CTRL_PREEMPTIBLE "pmix.jctrl.preempt" (bool)

```

1 Indicate that the job can be pre-empted.

2 **PMIX_JOB_CTRL_TERMINATE** "pmix.jctrl.term" (bool)

3 Politely terminate the specified processes.

4 **PMIX_REGISTER_CLEANUP** "pmix.reg.cleanup" (char*)

5 Comma-delimited list of files to be removed upon process termination

6 **PMIX_REGISTER_CLEANUP_DIR** "pmix.reg.cleanupdir" (char*)

7 Comma-delimited list of directories to be removed upon process termination

8 **PMIX_CLEANUP_RECURSIVE** "pmix.clnup.recurse" (bool)

9 Recursively cleanup all subdirectories under the specified one(s)

10 **PMIX_CLEANUP_EMPTY** "pmix.clnup.empty" (bool)

11 Only remove empty subdirectories

12 **PMIX_CLEANUP_IGNORE** "pmix.clnup.ignore" (char*)

13 Comma-delimited list of filenames that are not to be removed

14 **PMIX_CLEANUP_LEAVE_TOPDIR** "pmix.clnup.lvtop" (bool)

15 When recursively cleaning subdirectories, do not remove the top-level directory (the one
16 given in the cleanup request)

17 14.4.29 Monitoring attributes

18 Attributes used to control monitoring of an executing application- these are values passed to the
19 [PMIx_Process_monitor_nb](#) API and are not accessed using the [PMIx_Get](#) API.

20 **PMIX_MONITOR_ID** "pmix.monitor.id" (char*)

21 Provide a string identifier for this request.

22 **PMIX_MONITOR_CANCEL** "pmix.monitor.cancel" (char*)

23 Identifier to be canceled (NULL means cancel all monitoring for this process).

24 **PMIX_MONITOR_APP_CONTROL** "pmix.monitor.appctrl" (bool)

25 The application desires to control the response to a monitoring event.

26 **PMIX_MONITOR_HEARTBEAT** "pmix.monitor.mbeat" (void)

27 Register to have the PMIx server monitor the requestor for heartbeats.

28 **PMIX_SEND_HEARTBEAT** "pmix.monitor.beat" (void)

29 Send heartbeat to local PMIx server.

30 **PMIX_MONITOR_HEARTBEAT_TIME** "pmix.monitor.btime" (uint32_t)

31 Time in seconds before declaring heartbeat missed.

32 **PMIX_MONITOR_HEARTBEAT_DROPS** "pmix.monitor.bdrop" (uint32_t)

33 Number of heartbeats that can be missed before generating the event.

34 **PMIX_MONITOR_FILE** "pmix.monitor.fmon" (char*)

35 Register to monitor file for signs of life.

36 **PMIX_MONITOR_FILE_SIZE** "pmix.monitor.fsize" (bool)

37 Monitor size of given file is growing to determine if the application is running.

38 **PMIX_MONITOR_FILE_ACCESS** "pmix.monitor.faccess" (char*)

39 Monitor time since last access of given file to determine if the application is running.

40 **PMIX_MONITOR_FILE_MODIFY** "pmix.monitor.fmod" (char*)

41 Monitor time since last modified of given file to determine if the application is running.

1 **PMIX_MONITOR_FILE_CHECK_TIME** "pmix.monitor.ftime" (uint32_t)
2 Time in seconds between checking the file.
3 **PMIX_MONITOR_FILE_DROPS** "pmix.monitor.fdrop" (uint32_t)
4 Number of file checks that can be missed before generating the event.

5 **14.4.30 Security attributes**

6 *PMIx v3.0* Attributes for managing security credentials

7 **PMIX_CRED_TYPE** "pmix.sec.ctype" (char*)
8 When passed in **PMIx_Get_credential**, a prioritized, comma-delimited list of desired
9 credential types for use in environments where multiple authentication mechanisms may be
10 available. When returned in a callback function, a string identifier of the credential type.
11 **PMIX_CRYPTO_KEY** "pmix.sec.key" (pmix_byte_object_t)
12 Blob containing crypto key

13 **14.4.31 IO Forwarding attributes**

14 *PMIx v3.0* Attributes used to control IO forwarding behavior

15 **PMIX_IOF_CACHE_SIZE** "pmix.iof.csize" (uint32_t)
16 The requested size of the server cache in bytes for each specified channel. By default, the
17 server is allowed (but not required) to drop all bytes received beyond the max size.
18 **PMIX_IOF_DROP_OLDEST** "pmix.iof.old" (bool)
19 In an overflow situation, drop the oldest bytes to make room in the cache.
20 **PMIX_IOF_DROP_NEWEST** "pmix.iof.new" (bool)
21 In an overflow situation, drop any new bytes received until room becomes available in the
22 cache (default).
23 **PMIX_IOF_BUFFERING_SIZE** "pmix.iof.bsize" (uint32_t)
24 Controls grouping of IO on the specified channel(s) to avoid being called every time a bit of
25 IO arrives. The library will execute the callback whenever the specified number of bytes
26 becomes available. Any remaining buffered data will be “flushed” upon call to deregister the
27 respective channel.
28 **PMIX_IOF_BUFFERING_TIME** "pmix.iof.btime" (uint32_t)
29 Max time in seconds to buffer IO before delivering it. Used in conjunction with buffering
30 size, this prevents IO from being held indefinitely while waiting for another payload to arrive.
31 **PMIX_IOF_COMPLETE** "pmix.iof.cmp" (bool)
32 Indicates whether or not the specified IO channel has been closed by the source.
33 **PMIX_IOF_TAG_OUTPUT** "pmix.iof.tag" (bool)
34 Tag output with the channel it comes from.
35 **PMIX_IOF_TIMESTAMP_OUTPUT** "pmix.iof.ts" (bool)
36 Timestamp output
37 **PMIX_IOF_XML_OUTPUT** "pmix.iof.xml" (bool)
38 Format output in XML

1 14.4.32 Application setup attributes

2 *PMIx v3.0* Attributes for controlling contents of application setup data

- 3 **PMIX_SETUP_APP_ENVARS** "pmix.setup.env" (bool)
- 4 Harvest and include relevant environmental variables
- 5 **PMIX_SETUP_APP_NONENVARS** "pmix.setup.nenv" (bool)
- 6 Include all relevant data other than environmental variables
- 7 **PMIX_SETUP_APP_ALL** "pmix.setup.all" (bool)
- 8 Include all relevant data

9 14.4.33 Attribute support level attributes

- 10 **PMIX_CLIENT_FUNCTIONS** "pmix.client.fns" (bool)
- 11 Request a list of functions supported by the PMIx client library
- 12 **PMIX_CLIENT_ATTRIBUTES** "pmix.client.attrs" (bool)
- 13 Request attributes supported by the PMIx client library
- 14 **PMIX_SERVER_FUNCTIONS** "pmix.srvr.fns" (bool)
- 15 Request a list of functions supported by the PMIx server library
- 16 **PMIX_SERVER_ATTRIBUTES** "pmix.srvr.attrs" (bool)
- 17 Request attributes supported by the PMIx server library
- 18 **PMIX_HOST_FUNCTIONS** "pmix.srvr.fns" (bool)
- 19 Request a list of functions supported by the host environment
- 20 **PMIX_HOST_ATTRIBUTES** "pmix.host.attrs" (bool)
- 21 Request attributes supported by the host environment
- 22 **PMIX_TOOL_FUNCTIONS** "pmix.tool.fns" (bool)
- 23 Request a list of functions supported by the PMIx tool library
- 24 **PMIX_TOOL_ATTRIBUTES** "pmix.setup.env" (bool)
- 25 Request attributes supported by the PMIx tool library functions

26 14.4.34 Descriptive attributes

- 27 **PMIX_MAX_VALUE** "pmix.descr.maxval" (varies)
- 28 Used in `pmix_regattr_t` to describe the maximum valid value for the associated
- 29 attribute.
- 30 **PMIX_MIN_VALUE** "pmix.descr.minval" (varies)
- 31 Used in `pmix_regattr_t` to describe the minimum valid value for the associated
- 32 attribute.
- 33 **PMIX_ENUM_VALUE** "pmix.descr.enum" (char*)
- 34 Used in `pmix_regattr_t` to describe accepted values for the associated attribute.
- 35 Numerical values shall be presented in a form convertible to the attribute's declared data
- 36 type. Named values (i.e., values defined by constant names via a typical C-language enum
- 37 declaration) must be provided as their numerical equivalent.

1 14.4.35 Process group attributes

2 *PMIx v4.0* Attributes for controlling the PMIx Group APIs

3 **PMIX_GROUP_ID** "pmix.grp.id" (char*)

4 User-provided group identifier

5 **PMIX_GROUP_LEADER** "pmix.grp.ldr" (bool)

6 This process is the leader of the group

7 **PMIX_GROUP_OPTIONAL** "pmix.grp.opt" (bool)

8 Participation is optional - do not return an error if any of the specified processes terminate
9 without having joined. The default is false

10 **PMIX_GROUP_NOTIFY_TERMINATION** "pmix.grp.notterm" (bool)

11 Notify remaining members when another member terminates without first leaving the group.
12 The default is false

13 **PMIX_GROUP_INVITE_DECLINE** "pmix.grp.decline" (bool)

14 Notify the inviting process that this process does not wish to participate in the proposed
15 group The default is true

16 **PMIX_GROUP_MEMBERSHIP** "pmix.grp.mbrs" (pmix_data_array_t*)

17 Array of group member ID's

18 **PMIX_GROUP_ASSIGN_CONTEXT_ID** "pmix.grp.actxid" (bool)

19 Requests that the RM assign a new context identifier to the newly created group. The
20 identifier is an unsigned, **size_t** value that the RM guarantees to be unique across the range
21 specified in the request. Thus, the value serves as a means of identifying the group within
22 that range. If no range is specified, then the request defaults to **PMIX_RANGE_SESSION** .

23 **PMIX_GROUP_CONTEXT_ID** "pmix.grp.ctxid" (size_t)

24 Context identifier assigned to the group by the host RM.

25 **PMIX_GROUP_LOCAL_ONLY** "pmix.grp.lcl" (bool)

26 Group operation only involves local processes. PMIx implementations are *required* to
27 automatically scan an array of group members for local vs remote processes - if only local
28 processes are detected, the implementation need not execute a global collective for the
29 operation unless a context ID has been requested from the host environment. This can result
30 in significant time savings. This attribute can be used to optimize the operation by indicating
31 whether or not only local processes are represented, thus allowing the implementation to
32 bypass the scan. The default is false

33 **PMIX_GROUP_ENDPT_DATA** "pmix.grp.endpt" (pmix_byte_object_t)

34 Data collected to be shared during group construction

35 14.5 Callback Functions

36 PMIx provides blocking and nonblocking versions of most APIs. In the nonblocking versions, a
37 callback is activated upon completion of the the operation. This section describes many of those
38 callbacks.

1 14.5.1 Release Callback Function

2 Summary

3 The `pmix_release_cbfunc_t` is used by the `pmix_modex_cbfunc_t` and
4 `pmix_info_cbfunc_t` operations to indicate that the callback data may be reclaimed/freed by
5 the caller.

6 Format

PMIx v1.0

C

```
7 typedef void (*pmix_release_cbfunc_t)
8             (void *cbdata)
```

C

9 INOUT `cbdata`

10 Callback data passed to original API call (memory reference)

11 Description

12 Since the data is “owned” by the host server, provide a callback function to notify the host server
13 that we are done with the data so it can be released.

14 14.5.2 Modex Callback Function

15 Summary

16 The `pmix_modex_cbfunc_t` is used by the `pmix_server_fencefn_t` and
17 `pmix_server_dmodex_req_fn_t` PMIx server operations to return modex business card
18 exchange (BCX) data.

PMIx v1.0

C

```
19 typedef void (*pmix_modex_cbfunc_t)
20             (pmix_status_t status,
21              const char *data, size_t ndata,
22              void *cbdata,
23              pmix_release_cbfunc_t release_fn,
24              void *release_cbdata)
```

C

25 IN `status`

26 Status associated with the operation (handle)

27 IN `data`

28 Data to be passed (pointer)

29 IN `ndata`

30 size of the data (`size_t`)

31 IN `cbdata`

32 Callback data passed to original API call (memory reference)

1 **IN** `release_fn`
2 Callback for releasing *data* (function pointer)
3 **IN** `release_cbdata`
4 Pointer to be passed to *release_fn* (memory reference)

5 **Description**

6 A callback function that is solely used by PMIx servers, and not clients, to return modex BCX data
7 in response to “fence” and “get” operations. The returned blob contains the data collected from
8 each server participating in the operation.

9 **14.5.3 Spawn Callback Function**

10 **Summary**

11 The `pmix_spawn_cbfunc_t` is used on the PMIx client side by `PMIx_Spawn_nb` and on
12 the PMIx server side by `pmix_server_spawn_fn_t`.

PMIx v1.0

```
▼ C ▼  
13 typedef void (*pmix_spawn_cbfunc_t)  
14     (pmix_status_t status,  
15      pmix_namespace_t nspace, void *cbdata);  
▲ C ▲
```

16 **IN** `status`
17 Status associated with the operation (handle)
18 **IN** `namespace`
19 Namespace string (`pmix_namespace_t`)
20 **IN** `cbdata`
21 Callback data passed to original API call (memory reference)

22 **Description**

23 The callback will be executed upon launch of the specified applications in `PMIx_Spawn_nb` , or
24 upon failure to launch any of them.

25 The *status* of the callback will indicate whether or not the spawn succeeded. The *namespace* of the
26 spawned processes will be returned, along with any provided callback data. Note that the returned
27 *namespace* value will not be protected by the PRI upon return from the callback function, so the
28 receiver must copy it if it needs to be retained.

1 14.5.4 Op Callback Function

2 Summary

3 The [pmix_op_cbfunc_t](#) is used by operations that simply return a status.

PMIx v1.0

```
▼ C ▶  
4 typedef void (*pmix_op_cbfunc_t)  
5     (pmix_status_t status, void *cbdata);  
▲ C ▶
```

6 **IN status**

7 Status associated with the operation (handle)

8 **IN cbdata**

9 Callback data passed to original API call (memory reference)

10 Description

11 Used by a wide range of PMIx API's including [PMIx_Fence_nb](#),
12 [pmix_server_client_connected_fn_t](#), [PMIx_server_register_namespace](#). This
13 callback function is used to return a status to an often nonblocking operation.

14 14.5.5 Lookup Callback Function

15 Summary

16 The [pmix_lookup_cbfunc_t](#) is used by [PMIx_Lookup_nb](#) to return data.

PMIx v1.0

```
▼ C ▶  
17 typedef void (*pmix_lookup_cbfunc_t)  
18     (pmix_status_t status,  
19      pmix_pdata_t data[], size_t ndata,  
20      void *cbdata);  
▲ C ▶
```

21 **IN status**

22 Status associated with the operation (handle)

23 **IN data**

24 Array of data returned ([pmix_pdata_t](#))

25 **IN ndata**

26 Number of elements in the *data* array (**size_t**)

27 **IN cbdata**

28 Callback data passed to original API call (memory reference)

Description

A callback function for calls to `PMIx_Lookup_nb`. The function will be called upon completion of the command with the *status* indicating the success or failure of the request. Any retrieved data will be returned in an array of `pmix_pdata_t` structs. The namespace and rank of the process that provided each data element is also returned.

Note that these structures will be released upon return from the callback function, so the receiver must copy/protect the data prior to returning if it needs to be retained.

14.5.6 Value Callback Function

Summary

The `pmix_value_cbfunc_t` is used by `PMIx_Get_nb` to return data.

PMIx v1.0

```
typedef void (*pmix_value_cbfunc_t)
    (pmix_status_t status,
     pmix_value_t *kv, void *cbdata);
```

IN status

Status associated with the operation (handle)

IN kv

Key/value pair representing the data (`pmix_value_t`)

IN cbdata

Callback data passed to original API call (memory reference)

Description

A callback function for calls to `PMIx_Get_nb`. The *status* indicates if the requested data was found or not. A pointer to the `pmix_value_t` structure containing the found data is returned. The pointer will be `NULL` if the requested data was not found.

14.5.7 Info Callback Function

Summary

The `pmix_info_cbfunc_t` is a general information callback used by various APIs.

PMIx v2.0

```
typedef void (*pmix_info_cbfunc_t)
    (pmix_status_t status,
     pmix_info_t info[], size_t ninfo,
     void *cbdata,
     pmix_release_cbfunc_t release_fn,
     void *release_cbdata);
```

C

1 **IN status**
2 Status associated with the operation (`pmix_status_t`)
3 **IN info**
4 Array of `pmix_info_t` returned by the operation (pointer)
5 **IN ninfo**
6 Number of elements in the *info* array (`size_t`)
7 **IN cbdata**
8 Callback data passed to original API call (memory reference)
9 **IN release_fn**
10 Function to be called when done with the *info* data (function pointer)
11 **IN release_cbdata**
12 Callback data to be passed to *release_fn* (memory reference)

Description

13
14 The *status* indicates if requested data was found or not. An array of `pmix_info_t` will contain
15 the key/value pairs.

16 14.5.8 Event Handler Registration Callback Function

17 The `pmix_evhdlr_reg_cbfunc_t` callback function.

Advice to users

18 The PMIx *ad hoc* v1.0 Standard defined an error handler registration callback function with a
19 compatible signature, but with a different type definition function name
20 (`pmix_errhandler_reg_cbfunc_t`). It was removed from the v2.0 Standard and is not included in this
21 document to avoid confusion.

PMIx v2.0

C

```
22 typedef void (*pmix_evhdlr_reg_cbfunc_t)  
23     (pmix_status_t status,  
24      size_t evhdlr_ref,  
25      void *cbdata)
```

C

26 **IN status**
27 Status indicates if the request was successful or not (`pmix_status_t`)
28 **IN evhdlr_ref**
29 Reference assigned to the event handler by PMIx — this reference * must be used to
30 deregister the err handler (`size_t`)
31 **IN cbdata**
32 Callback data passed to original API call (memory reference)

1 **Description**
2 Define a callback function for calls to [PMIx_Register_event_handler](#)

3 14.5.9 Notification Handler Completion Callback Function

4 **Summary**

5 The [pmix_event_notification_cbfnc_fn_t](#) is called by event handlers to indicate
6 completion of their operations.

PMIx v2.0

C

```
7 typedef void (*pmix_event_notification_cbfnc_fn_t)
8     (pmix_status_t status,
9      pmix_info_t *results, size_t nresults,
10     pmix_op_cbfnc_t cbfunc, void *thiscbdata,
11     void *notification_cbdata);
```

C

- 12 **IN status**
13 Status returned by the event handler's operation ([pmix_status_t](#))
- 14 **IN results**
15 Results from this event handler's operation on the event ([pmix_info_t](#))
- 16 **IN nresults**
17 Number of elements in the results array ([size_t](#))
- 18 **IN cbfunc**
19 [pmix_op_cbfnc_t](#) function to be executed when PMIx completes processing the
20 callback (function reference)
- 21 **IN thiscbdata**
22 Callback data that was passed in to the handler (memory reference)
- 23 **IN cbdata**
24 Callback data to be returned when PMIx executes cbfunc (memory reference)

25 **Description**

26 Define a callback by which an event handler can notify the PMIx library that it has completed its
27 response to the notification. The handler is *required* to execute this callback so the library can
28 determine if additional handlers need to be called. The handler shall return
29 [PMIX_EVENT_ACTION_COMPLETE](#) if no further action is required. The return status of each
30 event handler and any returned [pmix_info_t](#) structures will be added to the *results* array of
31 [pmix_info_t](#) passed to any subsequent event handlers to help guide their operation.

32 If non-NULL, the provided callback function will be called to allow the event handler to release the
33 provided info array and execute any other required cleanup operations.

1 14.5.10 Notification Function

2 Summary

3 The `pmix_notification_fn_t` is called by PMIx to deliver notification of an event.

Advice to users

4 The PMIx *ad hoc* v1.0 Standard defined an error notification function with an identical name, but
5 different signature than the v2.0 Standard described below. The *ad hoc* v1.0 version was removed
6 from the v2.0 Standard is not included in this document to avoid confusion.

PMIx v2.0

```
7 typedef void (*pmix_notification_fn_t)
8     (size_t evhdlr_registration_id,
9      pmix_status_t status,
10     const pmix_proc_t *source,
11     pmix_info_t info[], size_t ninfo,
12     pmix_info_t results[], size_t nresults,
13     pmix_event_notification_cbfnc_fn_t cbfunc,
14     void *cbdata);
```

15 **IN** `evhdlr_registration_id`

Registration number of the handler being called (`size_t`)

17 **IN** `status`

Status associated with the operation (`pmix_status_t`)

19 **IN** `source`

Identifier of the process that generated the event (`pmix_proc_t`). If the source is the SMS, then the nspace will be empty and the rank will be `PMIX_RANK_UNDEF`

22 **IN** `info`

Information describing the event (`pmix_info_t`). This argument will be `NULL` if no additional information was provided by the event generator.

25 **IN** `ninfo`

Number of elements in the info array (`size_t`)

27 **IN** `results`

Aggregated results from prior event handlers servicing this event (`pmix_info_t`). This argument will be `NULL` if this is the first handler servicing the event, or if no prior handlers provided results.

31 **IN** `nresults`

Number of elements in the results array (`size_t`)

33 **IN** `cbfunc`

`pmix_event_notification_cbfnc_fn_t` callback function to be executed upon completion of the handler's operation and prior to handler return (function reference).

IN `cbdata`

Callback data to be passed to `cbfunc` (memory reference)

Description

Note that different RMs may provide differing levels of support for event notification to application processes. Thus, the *info* array may be **NULL** or may contain detailed information of the event. It is the responsibility of the application to parse any provided info array for defined key-values if it so desires.

Advice to users

Possible uses of the *info* array include:

- for the host RM to alert the process as to planned actions, such as aborting the session, in response to the reported event
- provide a timeout for alternative action to occur, such as for the application to request an alternate response to the event

For example, the RM might alert the application to the failure of a node that resulted in termination of several processes, and indicate that the overall session will be aborted unless the application requests an alternative behavior in the next 5 seconds. The application then has time to respond with a checkpoint request, or a request to recover from the failure by obtaining replacement nodes and restarting from some earlier checkpoint.

Support for these options is left to the discretion of the host RM. Info keys are included in the common definitions above but may be augmented by environment vendors.

Advice to PMIx server hosts

On the server side, the notification function is used to inform the PMIx server library's host of a detected event in the PMIx server library. Events generated by PMIx clients are communicated to the PMIx server library, but will be relayed to the host via the `pmix_server_notify_event_fn_t` function pointer, if provided.

14.5.11 Server Setup Application Callback Function

The `PMIx_server_setup_application` callback function.

Summary

Provide a function by which the resource manager can receive application-specific environmental variables and other setup data prior to launch of an application.

1
PMIx v2.0

Format

C

```
2 typedef void (*pmix_setup_application_cbfunc_t) (  
3         pmix_status_t status,  
4         pmix_info_t info[], size_t ninfo,  
5         void *provided_cbdata,  
6         pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

- 7 **IN status**
8 returned status of the request ([pmix_status_t](#))
- 9 **IN info**
10 Array of info structures (array of handles)
- 11 **IN ninfo**
12 Number of elements in the *info* array (integer)
- 13 **IN provided_cbdata**
14 Data originally passed to call to [PMIx_server_setup_application](#) (memory
15 reference)
- 16 **IN cbfunc**
17 [pmix_op_cbfunc_t](#) function to be called when processing completed (function reference)
- 18 **IN cbdata**
19 Data to be passed to the *cbfunc* callback function (memory reference)

Description

21 Define a function to be called by the PMIx server library for return of application-specific setup
22 data in response to a request from the host RM. The returned *info* array is owned by the PMIx
23 server library and will be free'd when the provided *cbfunc* is called.

24 14.5.12 Server Direct Modex Response Callback Function

25 The [PMIx_server_dmodex_request](#) callback function.

26 Summary

27 Provide a function by which the local PMIx server library can return connection and other data
28 posted by local application processes to the host resource manager.

29 Format

C

PMIx v1.0

```
30 typedef void (*pmix_dmodex_response_fn_t) (pmix_status_t status,  
31         char *data, size_t sz,  
32         void *cbdata);
```

C

1 **IN status**
2 Returned status of the request (`pmix_status_t`)
3 **IN data**
4 Pointer to a data "blob" containing the requested information (handle)
5 **IN sz**
6 Number of bytes in the *data* blob (integer)
7 **IN cbdata**
8 Data passed into the initial call to `PMIx_server_dmodex_request` (memory reference)

Description

9
10 Define a function to be called by the PMIx server library for return of information posted by a local
11 application process (via `PMIx_Put` with subsequent `PMIx_Commit`) in response to a request
12 from the host RM. The returned *data* blob is owned by the PMIx server library and will be free'd
13 upon return from the function.

14 14.5.13 PMIx Client Connection Callback Function

15 Summary

16 Callback function for incoming connection request from a local client

17 Format

PMIx v1.0

C

```
18 typedef void (*pmix_connection_cbfunc_t) (  
19                                     int incoming_sd, void *cbdata)
```

C

20 **IN incoming_sd**
21 (integer)
22 **IN cbdata**
23 (memory reference)

24 Description

25 Callback function for incoming connection requests from local clients - only used by host
26 environments that wish to directly handle socket connection requests.

27 14.5.14 PMIx Tool Connection Callback Function

28 Summary

29 Callback function for incoming tool connections.

1
PMIx v2.0

Format

C

```
2 typedef void (*pmix_tool_connection_cbfunc_t) (  
3             pmix_status_t status,  
4             pmix_proc_t *proc, void *cbdata)
```

C

- 5 **IN** `status`
- 6 `pmix_status_t` value (handle)
- 7 **IN** `proc`
- 8 `pmix_proc_t` structure containing the identifier assigned to the tool (handle)
- 9 **IN** `cbdata`
- 10 Data to be passed (memory reference)

Description

11 Callback function for incoming tool connections. The host environment shall provide a
12 namespace/rank identifier for the connecting tool.
13

Advice to PMIx server hosts

14 It is assumed that `rank=0` will be the normal assignment, but allow for the future possibility of a
15 parallel set of tools connecting, and thus each process requiring a unique rank.

16 14.5.15 Credential callback function

17 Summary

18 Callback function to return a requested security credential

1
PMIx v3.0

Format

C

```

typedef void (*pmix_credential_cbfunc_t) (
    pmix_status_t status,
    pmix_byte_object_t *credential,
    pmix_info_t info[], size_t ninfo,
    void *cbdata)

```

C

- 7 **IN status**
- 8 `pmix_status_t` value (handle)
- 9 **IN credential**
- 10 `pmix_byte_object_t` structure containing the security credential (handle)
- 11 **IN info**
- 12 Array of provided by the system to pass any additional information about the credential - e.g.,
- 13 the identity of the issuing agent. (handle)
- 14 **IN ninfo**
- 15 Number of elements in *info* (`size_t`)
- 16 **IN cbdata**
- 17 Object passed in original request (memory reference)

Description

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.

28 14.5.16 Credential validation callback function

29 Summary

30 Callback function for security credential validation

1
PMIx v3.0

Format

C

```
2 typedef void (*pmix_validation_cbfunc_t) (  
3         pmix_status_t status,  
4         pmix_info_t info[], size_t ninfo,  
5         void *cbdata);
```

C

- 6 **IN status**
7 `pmix_status_t` value (handle)
- 8 **IN info**
9 Array of `pmix_info_t` provided by the system to pass any additional information about
10 the authentication - e.g., the effective userid and group id of the certificate holder, and any
11 related authorizations (handle)
- 12 **IN ninfo**
13 Number of elements in *info* (`size_t`)
- 14 **IN cbdata**
15 Object passed in original request (memory reference)

Description

16 Define a validation callback function to indicate if a provided credential is valid, and any
17 corresponding information regarding authorizations and other security matters.
18

Advice to users

19 The precise contents of the array will depend on the host environment and its associated security
20 system. At the minimum, it is expected (but not required) that the array will contain entries for the
21 `PMIX_USERID` and `PMIX_GRPID` of the client described in the credential. The *info* array is
22 owned by the PMIx library and is not to be released or altered by the receiving party.

23 14.5.17 IOF delivery function

24 Summary

25 Callback function for delivering forwarded IO to a process

1
PMIx v3.0

Format

C

```
2 typedef void (*pmix_iof_cbfunc_t) (  
3     size_t iofhdlr, pmix_iof_channel_t channel,  
4     pmix_proc_t *source, char *payload,  
5     pmix_info_t info[], size_t ninfo);
```

C

- 6 **IN** `iofhdlr`
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_IOF_COMPLETE` (handle)
- 18 **IN** `ninfo`
19 Number of elements in `info` (`size_t`)

Description

Define a callback function for delivering forwarded IO to a process. This function will be called whenever data becomes available, or a specified buffering size and/or time has been met.

Advice to users


23 Multiple strings may be included in a given *payload*, and the *payload* may *not* be **NULL** terminated.
24 The user is responsible for releasing the *payload* memory. The *info* array is owned by the PMIx
25 library and is not to be released or altered by the receiving party.

26 14.5.18 IOF and Event registration function


27 Summary

28 Callback function for calls to register handlers, e.g., event notification and IOF requests.

1 **Format**

PMIx v3.0 

```
2 typedef void (*pmix_hdlr_reg_cbfunc_t) (pmix_status_t status,
3                                         size_t refid,
4                                         void *cbdata);
```



5 **IN status**

6 **PMIX_SUCCESS** or an appropriate error constant (**pmix_status_t**)

7 **IN refid**

8 reference identifier assigned to the handler by PMIx, used to deregister the handler (**size_t**)

9 **IN cbdata**

10 object provided to the registration call (pointer)

11 **Description**


12 Callback function for calls to register handlers, e.g., event notification and IOF requests.

13 14.6 Constant String Functions


14 Provide a string representation for several types of values. Note that the provided string is statically
 15 defined and must NOT be **free**'d.

16 **Summary**

17 String representation of a **pmix_status_t**.


PMIx v1.0 

```
18 const char*
19 PMIx_Error_string (pmix_status_t status);
```




20 **Summary**

21 String representation of a **pmix_proc_state_t**.

PMIx v2.0 

```
22 const char*
23 PMIx_Proc_state_string (pmix_proc_state_t state);
```



1 **Summary**
2 String representation of a `pmix_scope_t` .

PMIx v2.0 ▼ C ▼

3 `const char*`
4 `PMIx_Scope_string(pmix_scope_t scope);`

▲ C ▲

5 **Summary**
6 String representation of a `pmix_persistence_t` .

PMIx v2.0 ▼ C ▼

7 `const char*`
8 `PMIx_Persistence_string(pmix_persistence_t persist);`

▲ C ▲

9 **Summary**
10 String representation of a `pmix_data_range_t` .

PMIx v2.0 ▼ C ▼

11 `const char*`
12 `PMIx_Data_range_string(pmix_data_range_t range);`

▲ C ▲

13 **Summary**
14 String representation of a `pmix_info_directives_t` .

PMIx v2.0 ▼ C ▼

15 `const char*`
16 `PMIx_Info_directives_string(pmix_info_directives_t directives);`

▲ C ▲

17 **Summary**
18 String representation of a `pmix_data_type_t` .

PMIx v2.0 ▼ C ▼

19 `const char*`
20 `PMIx_Data_type_string(pmix_data_type_t type);`

▲ C ▲

1 **Summary**
2 String representation of a `pmix_alloc_directive_t`.

PMIx v2.0 ▼  ▼

3 `const char*`
4 `PMIx_Alloc_directive_string(pmix_alloc_directive_t directive);`

▲  ▲

5 **Summary**
6 String representation of a `pmix_iof_channel_t`.

PMIx v3.0 ▼  ▼

7 `const char*`
8 `PMIx_IOF_channel_string(pmix_iof_channel_t channel);`

▲  ▲

9 **Summary**
10 String representation of a `pmix_job_state_t`.

PMIx v4.0 ▼  ▼

11 `const char*`
12 `PMIx_Job_state_string(pmix_job_state_t state);`

▲  ▲

13 **Summary**
14 String representation of a PMIx attribute

PMIx v4.0 ▼  ▼

15 `const char*`
16 `PMIx_Get_attribute_string(char *attributename);`

▲  ▲

17 **Summary**
18 Return the PMIx attribute name corresponding to the given attribute string

PMIx v4.0 ▼  ▼

19 `const char*`
20 `PMIx_Get_attribute_name(char *attributestring);`

▲  ▲

1 **Summary**
2 String representation of a `pmix_link_state_t`

PMIx v4.0



C

3 `const char*`
4 `PMIx_Link_state_string(pmix_link_state_t state);`



C

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.

11 A.1 Design Considerations

12 Several issues arose during design of the Python bindings:

13 A.1.1 Error Codes vs Python Exceptions

14 The C programming language reports errors through the return of the corresponding integer status
15 codes. PMIx has defined a range of negative values for this purpose. However, Python has the
16 option of raising *exceptions* that effectively operate as interrupts that can be trapped if the program
17 appropriately tests for them. The PMIx Python bindings opted to follow the C-based standard and
18 return PMIx status codes in lieu of raising exceptions as this method was considered more
19 consistent for those working in both domains.

20 A.1.2 Representation of Structured Data

21 PMIx utilizes a number of C-language structures to efficiently bundle related information. For
22 example, the PMIx process identifier is represented as a struct containing a character array for the
23 namespace and a 32-bit unsigned integer for the process rank. There are several options for
24 translating such objects to Python – e.g., the PMIx process identifier could be represented as a
25 two-element tuple (nspace, rank) or as a dictionary ‘nspace’: name, ‘rank’: 0. Exploration found no
26 discernible benefit to either representation, nor was any clearly identifiable rationale developed that
27 would lead a user to expect one versus the other for a given PMIx data type. Consistency in the
28 translation (i.e., exclusively using tuple or dictionary) appeared to be the most important criterion.
29 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 14 on page 287 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
<code>bool</code>	PMIX_BOOL	boolean	
<code>byte</code>	PMIX_BYTE	A single element byte array (i.e., a byte array of length one)	
<code>char*</code>	PMIX_STRING	string	
<code>size_t</code>	PMIX_SIZE	integer	
<code>pid_t</code>	PMIX_PID	integer	value shall be limited to the <code>uint32_t</code> range
<code>int, int8_t, int16_t, int32_t, int64_t</code>	PMIX_INT, PMIX_INT8, PMIX_INT16, PMIX_INT32, PMIX_INT64	integer	value shall be limited to its corresponding range
<code>uint, uint8_t, uint16_t, uint32_t, uint64_t</code>	PMIX_UINT, PMIX_UINT8, PMIX_UINT16, PMIX_UINT32, PMIX_UINT64	integer	value shall be limited to its corresponding range
<code>float, double</code>	PMIX_FLOAT, PMIX_DOUBLE	float	value shall be limited to its corresponding range
<code>struct timeval</code>	PMIX_TIMEVAL	{'sec': sec, 'usec': microsec}	each field is an integer value
<code>time_t</code>	PMIX_TIME	integer	limited to positive values
<code>pmix_data_type_t</code>	PMIX_DATA_TYPE	integer	value shall be limited to the <code>uint16_t</code> range
<code>pmix_status_t</code>	PMIX_STATUS	integer	
<code>pmix_key_t</code>	N/A	string	The string's length shall be limited to one less than the size of the <code>pmix_key_t</code> array (to reserve space for the terminating NULL)
<code>pmix_nspace_t</code>	N/A	string	The string's length shall be limited to one less than the size of the <code>pmix_nspace_t</code> array (to reserve space for the terminating NULL)

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_BYTE_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_envvar_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': {'nspace': 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 <i>lists</i> containing Python strings; <i>maxprocs</i> is an integer; and <i>info</i> is a Python <i>list</i> of info values
<code>pmix_query_t</code>	PMIX_QUERY	{'keys': [keys], 'qualifiers': [info]}	<i>keys</i> is a Python <i>list</i> of Python strings, and <i>qualifiers</i> is a Python <i>list</i> of info 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 <i>list</i> of info 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 uint8_t 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':PMIX_STRING},  
27         {'key':PMIX_MODEL_LIBRARY_NAME,  
28         'value':'PMIX', 'val_type':PMIX_STRING}]  
29 (rc,myproc) = myclient.init(info)  
30
```

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
   PMIx v4.0
2 def iofcbfunc(iofhdlr:integer, channel:integer,
3                 source:dict, payload:dict, info:list)
   Python
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
   PMIx v4.0
21 def evhandler(evhdlr:integer, status:integer,
22                 source:dict, info:list, results:list)
   Python
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:

```

- 1 • *rc* - Status returned by the event handler's operation (integer)
- 2 • *results* - List of Python **info** containing results from this event handler's operation on the event
- 3 (list)
- 4 See [pmix_notification_fn_t](#) for details

5 A.3.3 Server Module Functions

6 The following definitions represent functions that may be provided to the PMIx server library at
 7 time of initialization for servicing of client requests. Module functions that are not provided default
 8 to returning "not supported" to the caller.

9 A.3.3.1 Client Connected

10 Summary

11 Notify the host server that a client connected to this server.

12 Format

PMIx v4.0

```

Python
def clientconnected(proc:dict is not None)
Python
  
```

14 IN *proc*

15 Python **proc** identifying the namespace/rank of the process that connected (dict)

16 Returns:

- 17 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the connection should be rejected
- 18 (integer)

19 See [pmix_server_client_connected_fn_t](#) for details

20 A.3.3.2 Client Finalized

21 Summary

22 Notify the host environment that a client called **PMIx_Finalize**.

23 Format

PMIx v4.0

```

Python
def clientfinalized(proc:dict is not None):
Python
  
```

25 IN *proc*

26 Python **proc** identifying the namespace/rank of the process that finalized (dict)

27 Returns: nothing

28 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**

PMIx v4.0

```
▼ Python ▼  
def clientaborted(args:dict is not None)  
▲ Python ▲
```

6 **IN args**

7 Python dictionary containing:

- 8 • 'caller': Python [proc](#) identifying the namespace/rank of the process calling abort (dict)
- 9 • 'status': PMIx status to be returned on exit (integer)
- 10 • 'msg': Optional string message to be printed (string)
- 11 • 'targets': Optional list of Python [proc](#) identifying the namespace/rank of the processes to
12 be aborted (list)

13 Returns:

- 14 • *rc* - [PMIX_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

15 See [pmix_server_abort_fn_t](#) for details

16 **A.3.3.4 Fence**

17 **Summary**

18 At least one client called either [PMIx_Fence](#) or [PMIx_Fence_nb](#)

19 **Format**

PMIx v4.0

```
▼ Python ▼  
def fence(args:dict is not None)  
▲ Python ▲
```

21 **IN args**

22 Python dictionary containing:

- 23 • 'procs': List of Python [proc](#) identifying the namespace/rank of the participating
24 processes (list)
- 25 • 'directives': Optional list of Python [info](#) containing directives controlling the operation
26 (list)
- 27 • 'data': Optional Python bytearray of data to be circulated during fence operation (bytearray)

28 Returns:

- *rc* - [PMIX_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- *data* - Python bytearray containing the aggregated data from all participants (bytearray)

See [pmix_server_fencefn_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)
```

IN args

Python dictionary containing:

- 'proc': Python [proc](#) of process whose data is being requested (dict)
- 'directives': Optional list of Python [info](#) containing directives controlling the operation (list)

Returns:

- *rc* - [PMIX_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)
- *data* - Python bytearray containing the data for the specified process (bytearray)

See [pmix_server_dmodex_req_fn_t](#) for details

A.3.3.6 Publish

Summary

Publish data per the PMIx API specification.

Format

PMIx v4.0

```
def publish(args:dict is not None)
```

IN args

Python dictionary containing:

- 'proc': Python [proc](#) dictionary of process publishing the data (dict)
- 'directives': List of Python [info](#) containing data and directives (list)

Returns:

- *rc* - [PMIX_SUCCESS](#) or a PMIx error code indicating the operation failed (integer)

See [pmix_server_publish_fn_t](#) for details

1 **A.3.3.7 Lookup**

2 **Summary**

3 Lookup published data.

4 **Format**

PMIx v4.0

▼ Python ▼
▲ Python ▲

5 **def lookup(args:dict is not None)**

6 **IN args**

7 Python dictionary containing:

- 8 • 'proc': Python **proc** of process seeking the data (dict)
- 9 • 'keys': List of Python strings (list)
- 10 • 'directives': Optional list of Python **info** containing directives (list)

11 Returns:

- 12 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 13 • *pdata* - List of **pdata** containing the returned results (list)

14 See [pmix_server_lookup_fn_t](#) for details

15 **A.3.3.8 Unpublish**

16 **Summary**

17 Delete data from the data store.

18 **Format**

PMIx v4.0

▼ Python ▼
▲ Python ▲

19 **def unpublish(args:dict is not None)**

20 **IN args**

21 Python dictionary containing:

- 22 • 'proc': Python **proc** of process unpublishing data (dict)
- 23 • 'keys': List of Python strings (list)
- 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_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**

PMIx v4.0

▼ Python ▼
▲ Python ▲

5 **def spawn(args:dict is not None)**

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 • *nspc* - 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**

PMIx v4.0

▼ Python ▼
▲ Python ▲

19 **def connect(args:dict is not None)**

20 **IN args**

21 Python dictionary containing:

- 22 • 'procs': List of Python [proc](#) identifying the namespace/rank of the participating processes (list)
- 23 • '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)

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**

PMIx v4.0

▼ Python ▼
▲ Python ▲

5 **def disconnect (args:dict is not None)**

6 **IN args**

7 Python dictionary containing:

- 8 • 'procs': List of Python **proc** identifying the namespace/rank of the participating
9 processes (list)
- 10 • 'directives': Optional list of Python **info** containing directives controlling the operation
11 (list)

12 Returns:

- 13 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

14 See [pmix_server_disconnect_fn_t](#) for details

15 **A.3.3.12 Register Events**

16 **Summary**

17 Register to receive notifications for the specified events.

18 **Format**

PMIx v4.0

▼ Python ▼
▲ Python ▲

19 **def register_events (args:dict is not None)**

20 **IN args**

21 Python dictionary containing:

- 22 • 'codes': List of Python integers (list)
- 23 • 'directives': Optional list of Python **info** containing directives controlling the operation
24 (list)

25 Returns:

- 26 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

27 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

PMIx v4.0

▼ Python ▼

5 `def deregister_events(args:dict is not None)`

▲ Python ▲

6 **IN** `args`

7 Python dictionary containing:

- 8 • 'codes': List of Python integers (list)

9 Returns:

- 10 • `rc` - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

11 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

PMIx v4.0

▼ Python ▼

16 `def notify_event(args:dict is not None)`

▲ Python ▲

17 **IN** `args`

18 Python dictionary containing:

- 19 • 'code': Python integer [pmix_status_t](#) (integer)
- 20 • 'source': Python [proc](#) of process that generated the event (dict)
- 21 • 'range': Python [range](#) in which the event is to be reported (integer)
- 22 • 'directives': Optional list of Python [info](#) directives (list)

23 Returns:

- 24 • `rc` - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)

25 See [pmix_server_notify_event_fn_t](#) for details

26 A.3.3.15 Query

27 Summary

28 Query information from the resource manager.

1 **Format**
PMIx v4.0 Python

2 `def query(args:dict is not None)`
 Python

3 **IN args**
 4 Python dictionary containing:

- 5 • 'source': Python **proc** of requesting process (dict)
- 6 • 'queries': List of Python **query** directives (list)

7 Returns:

- 8 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 9 • *info* - List of Python **info** containing the returned results (list)

10 See [pmix_server_query_fn_t](#) for details

11 A.3.3.16 Tool Connected

12 Summary

13 Register that a tool has connected to the server.

14 Format

PMIx v4.0 Python

15 `def tool_connected(args:dict is not None)`
 Python

16 IN args

17 Python dictionary containing:

- 18 • 'directives': Optional list of Python **info** info on the connecting tool (list)

19 Returns:

- 20 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- 21 • *proc* - Python **proc** containing the assigned namespace:rank for the tool (dict)

22 See [pmix_server_tool_connection_fn_t](#) for details

23 A.3.3.17 Log

24 Summary

25 Log data on behalf of a client.

1 **Format**
PMIx v4.0 Python

2 `def log(args:dict is not None)`
Python

3 **IN args**
4 Python dictionary containing:
5 • 'source': Python **proc** of requesting process (dict)
6 • 'data': Optional list of Python **info** containing data to be logged (list)
7 • 'directives': Optional list of Python **info** containing directives (list)
8 Returns:
9 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
10 See [pmix_server_log_fn_t](#) for details

11 **A.3.3.18 Allocate Resources**

12 **Summary**
13 Request allocation operations on behalf of a client.

14 **Format**
PMIx v4.0 Python

15 `def allocate(args:dict is not None)`
Python

16 **IN args**
17 Python dictionary containing:
18 • 'source': Python **proc** of requesting process (dict)
19 • 'action': Python **allocdir** specifying requested action (integer)
20 • 'directives': Optional list of Python **info** containing directives (list)
21 Returns:
22 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
23 • *refarginfo* - List of Python **info** containing results of requested operation (list)
24 See [pmix_server_alloc_fn_t](#) for details

25 **A.3.3.19 Job Control**

26 **Summary**
27 Execute a job control action on behalf of a client.

1 **Format**
PMIx v4.0 Python

2 `def job_control(args:dict is not None)`
Python

3 **IN args**
4 Python dictionary containing:
5 • 'source': Python **proc** of requesting process (dict)
6 • 'targets': List of Python **proc** specifying target processes (list)
7 • 'directives': Optional list of Python **info** containing directives (list)
8 Returns:
9 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
10 See [pmix_server_job_control_fn_t](#) for details

11 A.3.3.20 Monitor

12 **Summary**
13 Request that a client be monitored for activity.

14 **Format**
PMIx v4.0 Python

15 `def monitor(args:dict is not None)`
Python

16 **IN args**
17 Python dictionary containing:
18 • 'source': Python **proc** of requesting process (dict)
19 • 'monitor': Python **info** attribute indicating the type of monitor being requested (dict)
20 • 'error': Status code to be used when generating an event notification (integer) alerting that
21 the monitor has been triggered.
22 • 'directives': Optional list of Python **info** containing directives (list)
23 Returns:
24 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
25 See [pmix_server_monitor_fn_t](#) for details

26 A.3.3.21 Get Credential

27 **Summary**
28 Request a credential from the host environment

1 **Format**
PMIx v4.0 Python

2 `def get_credential(args:dict is not None)`
Python

3 **IN args**
4 Python dictionary containing:
5 • 'source': Python **proc** of requesting process (dict)
6 • 'directives': Optional list of Python **info** containing directives (list)

7 Returns:
8 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
9 • *cred* - Python **byteobject** containing returned credential (dict)
10 • *info* - List of Python **info** containing any additional info about the credential (list)
11 See [pmix_server_get_cred_fn_t](#) for details

12 **A.3.3.22 Validate Credential**

13 **Summary**
14 Request validation of a credential

15 **Format**
PMIx v4.0 Python

16 `def validate_credential(args:dict is not None)`
Python

17 **IN args**
18 Python dictionary containing:
19 • 'source': Python **proc** of requesting process (dict)
20 • 'credential': Python **byteobject** containing credential (dict)
21 • 'directives': Optional list of Python **info** containing directives (list)

22 Returns:
23 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
24 • *info* - List of Python **info** containing any additional info from the credential (list)
25 See [pmix_server_validate_cred_fn_t](#) for details

26 **A.3.3.23 IO Forward**

27 **Summary**
28 Request the specified IO channels be forwarded from the given array of processes.

1 **Format**
PMIx v4.0 Python

2 `def iof_pull(args:dict is not None)` Python

3 **IN args**
4 Python dictionary containing:
5 • 'sources': List of Python **proc** of processes whose IO is being requested (list)
6 • 'channels': Bitmask of Python **channel** identifying IO channels to be forwarded (integer)
7 • 'directives': Optional list of Python **info** containing directives (list)
8 Returns:
9 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
10 See [pmix_server_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**
PMIx v4.0 Python

15 `def iof_push(args:dict is not None)` Python

16 **IN args**
17 Python dictionary containing:
18 • 'source': Python **proc** of process whose input is being forwarded (dict)
19 • 'payload': Python **byteobject** containing input bytes (dict)
20 • 'targets': List of **proc** of processes that are to receive the payload (list)
21 • 'directives': Optional list of Python **info** containing directives (list)
22 Returns:
23 • *rc* - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
24 See [pmix_server_stdin_fn_t](#) for details

25 A.3.3.25 Group Operations

26 **Summary**
27 Request group operations (construct, destruct, etc.) on behalf of a set of processes.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

PMIx v4.0

Format

Python

```
def group(args:dict is not None)
```

Python

IN args

Python dictionary containing:

- 'op': Operation host is to perform on the specified group (integer)
- 'group': String identifier of target group (str)
- 'procs': List of Python **proc** of participating processes (dict)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- rc - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python **info** containing results of requested operation (list)

See [pmix_server_grp_fn_t](#) for details

A.3.3.26 Fabric Operations

Summary

Request fabric-related operations (e.g., information on a fabric) on behalf of a tool or other process.

Format

PMIx v4.0

Python

```
def fabric(args:dict is not None)
```

Python

IN args

Python dictionary containing:

- 'source': Python **proc** of requesting process (dict)
- 'op': Operation host is to perform on the specified fabric (integer)
- 'directives': Optional list of Python **info** containing directives (list)

Returns:

- rc - **PMIX_SUCCESS** or a PMIx error code indicating the operation failed (integer)
- refarginfo - List of Python **info** containing results of requested operation (list)

See [pmix_server_fabric_fn_t](#) for details

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

PMIx v4.0

▼ Python ▼

10 `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

PMIx v4.0

▼ Python ▼

19 `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**

PMIx v4.0

▼ Python _____ ▼

3 `vers = myclient.get_version()`

▲ Python _____ ▲

4 Returns:

- 5 • *vers* - Python string containing the version of the PMIx library (e.g., "3.1.4") (integer)

6 See [PMIx_Get_version](#) for description of all relevant attributes and behaviors

7 **A.4.4 Client.finalize**

8 **Summary**

9 Finalize the PMIx client library.

10 **Format**

PMIx v4.0

▼ Python _____ ▼

11 `rc = myclient.finalize(info:list)`

▲ Python _____ ▲

12 **IN** `info`

13 List of Python `info` dictionaries (list)

14 Returns:

- 15 • *rc* - [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

16 See [PMIx_Finalize](#) for description of all relevant attributes and behaviors

17 **A.4.5 Client.abort**

18 **Summary**

19 Request that the provided list of procs be aborted

1 **Format**

PMIx v4.0 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**

PMIx v4.0 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
PMIx v4.0

Format

Python

2 `rc = myclient.put(scope:integer, key:str, value:dict)`

Python

- 3 **IN** `scope`
4 Scope of the data being posted (integer)
5 **IN** `key`
6 String key of the data (string)
7 **IN** `value`
8 Python `value` dictionary (dict)

9 Returns:

- 10 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

11 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

PMIx v4.0

Python

16 `rc = myclient.commit()`

Python

17 Returns:

- 18 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

19 See **PMIx_Commit** for description of all relevant attributes and behaviors

20 A.4.9 Client.fence

21 Summary

22 Execute a blocking barrier across the processes identified in the specified list

1 **Format**

PMIx v4.0 Python

```
2 rc = myclient.fence(peers:list, directives:list)
```

Python

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_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**

PMIx v4.0 Python

```
14 rc, val = myclient.get(proc:dict, key:str, directives:list)
```

Python

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**
PMIx v4.0 Python

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**
PMIx v4.0 Python

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**
PMIx v4.0 Python

2 `rc = myclient.unpublish(keys:list, directives:list)`
 Python

3 **IN keys**
 4 List of Python string keys identifying data to be deleted (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_Unpublish** for description of all relevant attributes and behaviors

10 A.4.14 Client.spawn

11 **Summary**
 12 Spawn a new job.

13 **Format**
PMIx v4.0 Python

14 `rc, nspace = myclient.spawn(jobinfo:list, apps:list)`
 Python

15 **IN jobinfo**
 16 List of Python **info** dictionaries (list)

17 **IN apps**
 18 List of Python **app** dictionaries (list)

19 Returns:

20 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

21 • `nspace` - Python **nspace** of the new job (dict)

22 See **PMIx_Spawn** for description of all relevant attributes and behaviors

23 A.4.15 Client.connect

24 **Summary**
 25 Connect namespaces.

1 **Format**
PMIx v4.0 Python

```
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**
PMIx v4.0 Python

```
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 **nspace** on the given node.

1 **Format**

PMIx v4.0 Python

```
2 rc,procs = myclient.resolve_peers(node:str, nspace:str)
```

Python

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

12 **Summary**
 13 Return list of nodes hosting processes within the specified **nspace** .

14 **Format**

PMIx v4.0 Python

```
15 rc,nodes = myclient.resolve_nodes(nspace:str)
```

Python

16 **IN nspace**
 17 Python **nspace** (str)

18 Returns:

19 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

20 • *nodes* - List of Python string node names (list)

21 See **PMIx_Resolve_nodes** for description of all relevant attributes and behaviors

22 A.4.19 Client.query

23 **Summary**
 24 Query information about the system in general

1 **Format**
PMIx v4.0 Python

2 `rc, info = myclient.query(queries:list)`
Python

3 **IN queries**
4 List of Python `query` dictionaries (list)

5 Returns:

6 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

7 • `info` - List of Python `info` containing results of the query (list)

8 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**
PMIx v4.0 Python

13 `rc = myclient.log(data:list, directives:list)`
Python

14 **IN data**
15 List of Python `info` (list)

16 **IN directives**
17 Optional list of Python `info` (list)

18 Returns:

19 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

20 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**
PMIx v4.0 Python

```
2 rc, info = myclient.allocate(request:integer, directives:list)
```

Python

3 **IN request**
4 Python [allocdir](#) 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**
PMIx v4.0 Python

```
15 rc, info = myclient.job_ctrl(targets:list, directives:list)
```

Python

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

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 • *cred* - Python **byteobject** containing returned credential (dict)

24 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

12 **Summary**
13 Construct a new group composed of the specified processes and identified with the provided group
14 identifier

15 **Format**
PMIx v4.0 Python

16 `rc, info = myclient.construct_group(grp:string, members:list, directives:list)`
Python

- 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

28 **Summary**
29 Explicitly invite specified processes to join a group

1 **Format**

PMIx v4.0 Python

2 `rc, info = myclient.group_invite(grp:string, members:list, directives:list)`
 Python

3 **IN grp**

4 Python string identifier for the group (str)

5 **IN members**

6 List of Python **proc** dictionaries identifying processes to be invited (list)

7 **IN directives**

8 Optional list of Python **info** describing request (list)

9 Returns:

10 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

11 • *info* - List of Python **info** containing results of the request (list)

12 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**

PMIx v4.0 Python

17 `rc, info = myclient.group_join(grp:string, leader:dict, opt:integer, directives:list)`
 Python

18 **IN grp**

19 Python string identifier for the group (str)

20 **IN leader**

21 Python **proc** dictionary identifying process leading the group (dict)

22 **IN opt**

23 One of the **pmix_group_opt_t** values indicating decline/accept (integer)

24 **IN directives**

25 Optional list of Python **info** describing request (list)

26 Returns:

27 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

28 • *info* - List of Python **info** containing results of the request (list)

29 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

PMIx v4.0

Python

5 `rc = myclient.group_leave(grp:string, directives:list)`

Python

6 **IN** `grp`

7 Python string identifier for the group (str)

8 **IN** `directives`

9 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

PMIx v4.0

Python

17 `rc = myclient.group_destruct(grp:string, directives:list)`

Python

18 **IN** `grp`

19 Python string identifier for the group (str)

20 **IN** `directives`

21 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
PMIx v4.0

Format

Python

2 `rc, id = myclient.register_event_handler(codes:list, directives:list, cbfunc)`

Python

3 **IN codes**

List of Python integer status codes that should be reported to this handler (llist)

4 **IN directives**

Optional list of Python [info](#) describing request (list)

5 **IN cbfunc**

Python [evhandler](#) to be called when event is received (func)

6 Returns:

- *rc* - [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)
- *id* - PMIx reference identifier for handler (integer)

7 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**

PMIx reference identifier for handler (integer)

19 Returns: None

20 See [PMIx_Deregister_event_handler](#) for description of all relevant attributes and behaviors

23 A.4.33 Client.notify_event

24 Summary

25 Report an event for notification via any registered handler.

1 **Format**

PMIx v4.0 Python

```
2 rc = myclient.notify_event(status:integer, source:dict,
3                             range:integer, directives:list)
```

Python

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

16 **Summary**
 17 Register for access to fabric-related information, including communication cost matrix.

18 **Format**

PMIx v4.0 Python

```
19 rc, fabricinfo = myserver.fabric_register(directives:list)
```

Python

20 **IN directives**
 21 Optional list of Python **info** containing directives (list)

22 Returns:

23 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

24 • *fabricinfo* - List of Python **info** containing fabric info (list)

25 See **PMIx_Fabric_register** for details

26 A.4.35 Client.fabric_update

27 **Summary**
 28 Update fabric-related information, including communication cost matrix.

1 **Format**
PMIx v4.0 Python

2 `rc, fabricinfo = myserver.fabric_update()`
Python

3 Returns:

- 4 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 5 • `fabricinfo` - List of Python **info** containing updated fabric info (list)

6 See **PMIx_Fabric_update** for details

7 **A.4.36 Client.fabric_deregister**

8 **Summary**
9 Deregister fabric

10 **Format**
PMIx v4.0 Python

11 `rc = myserver.fabric_deregister()`
Python

12 Returns:

- 13 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

14 See **PMIx_Fabric_deregister** for details

15 **A.4.37 Client.fabric_get_vertex_info**

16 **Summary**
17 Given a communication cost matrix index for a specified fabric, return an array of information
18 describing the corresponding NIC.

19 **Format**
PMIx v4.0 Python

20 `rc, nicinfo = myserver.fabric_get_vertex_info(index:integer)`
Python

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_index

2 Summary

3 Given info describing a given vertex, return the corresponding communication cost matrix index

4 Format

PMIx v4.0

Python

```
5 rc, index = myserver.fabric_get_index(info:list)
```

Python

6 IN info

7 List of Python **info** containing vertex description (list)

8 Returns:

- 9 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 10 • *index* - Index of corresponding NIC (integer)

11 See [PMIx_Fabric_get_index](#) for details

12 A.4.39 Client.error_string

13 Summary

14 Pretty-print string representation of [pmix_status_t](#).

15 Format

PMIx v4.0

Python

```
16 rep = myclient.error_string(status:integer)
```

Python

17 IN status

18 PMIx status code (integer)

19 Returns:

- 20 • *rep* - String representation of the provided status code (str)

21 See [PMIx_Error_string](#) for further details

22 A.4.40 Client.proc_state_string

23 Summary

24 Pretty-print string representation of [pmix_proc_state_t](#).

1 **Format**
PMIx v4.0 Python

2 `rep = myclient.proc_state_string(state:integer)`
Python

3 **IN state**
4 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**
PMIx v4.0 Python

12 `rep = myclient.scope_string(scope:integer)`
Python

13 **IN scope**
14 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**
PMIx v4.0 Python

22 `rep = myclient.persistence_string(persistence:integer)`
Python

23 **IN persistence**
24 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**
PMIx v4.0 Python

2 `rep = myclient.data_type_string(dtype:integer)`
Python

3 **IN dtype**
4 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**

9 **Summary**

10 Pretty-print string representation of [pmix_alloc_directive_t](#).

11 **Format**
PMIx v4.0 Python

12 `rep = myclient.alloc_directive_string(adir:integer)`
Python

13 **IN adir**
14 PMIx allocation directive value (integer)

15 Returns:

- 16 • *rep* - String representation of the provided allocation directive (str)

17 See [PMIx Alloc directive string](#) for further details

18 **A.4.47 Client.iof_channel_string**

19 **Summary**

20 Pretty-print string representation of [pmix_iof_channel_t](#).

21 **Format**
PMIx v4.0 Python

22 `rep = myclient.iof_channel_string(channel:integer)`
Python

23 **IN channel**
24 PMIx IOF channel value (integer)

25 Returns:

- 26 • *rep* - String representation of the provided IOF channel (str)

27 See [PMIx IOF 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

PMIx v4.0

▼ Python ▼

5 `rep = myclient.job_state_string(state:integer)`

▲ Python ▲

6 IN state

7 PMIx job state value (integer)

8 Returns:

- 9 • *rep* - String representation of the provided job state (str)

10 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

PMIx v4.0

▼ Python ▼

15 `rep = myclient.get_attribute_string(attribute:str)`

▲ Python ▲

16 IN attribute

17 PMIx attribute name (string)

18 Returns:

- 19 • *rep* - String representation of the provided attribute (str)

20 See [PMIx_Get_attribute_string](#) for further details

21 A.4.50 Client.get_attribute_name

22 Summary

23 Pretty-print name of a PMIx attribute corresponding to the provided string

1 **Format**
PMIx v4.0 Python

2 `rep = myclient.get_attribute_name(attribute:str)`
Python

3 **IN** `attributestring`
4 Attribute string (string)

5 Returns:

6 • `rep` - Attribute name corresponding to the provided string (str)

7 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**
PMIx v4.0 Python

12 `rep = myclient.link_state_string(state:integer)`
Python

13 **IN** `state`
14 PMIx link state value (integer)

15 Returns:

16 • `rep` - String representation of the provided link state (str)

17 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**

PMIx v4.0 Python

```
2 rc = myserver.init(directives:list, map:dict)
```

Python

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

12 **Summary**

13 Finalize the PMIx server library

14 **Format**

PMIx v4.0 Python

```
15 rc = myserver.finalize()
```

Python

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

20 **Summary**

21 Generate a regular expression representation of the input strings.

1
PMIx v4.0

Format

Python

2 `rc, regex = myserver.generate_regex(input:list)`

Python

3 **IN** `input`

4 List of Python strings (e.g., node names) (list)

5 Returns:

- 6 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 7 • `regex` - Python **bytearray** containing regular expression representation of the input list
- 8 (**bytearray**)

9 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

PMIx v4.0

Python

14 `rc, regex = myserver.generate_ppn(input:list)`

Python

15 **IN** `input`

16 List of Python strings, each string consisting of a comma-delimited list of ranks on each node,
17 with the strings being in the same order as the node names provided to "generate_regex" (list)

18 Returns:

- 19 • `rc` - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 20 • `regex` - Python **bytearray** containing regular expression representation of the input list
- 21 (**bytearray**)

22 See [PMIx_generate_ppn](#) for details

23 A.5.5 Server.register_namespace

24 Summary

25 Setup the data about a particular namespace.

1
PMIx v4.0

Format

Python

2
3
4

```
rc = myserver.register_namespace(namespace:str,  
                                nlocalprocs:integer,  
                                directives:list)
```

Python

5
6
7
8
9
10

- IN namespace**
Python string containing the namespace (str)
- IN nlocalprocs**
Number of local processes (integer)
- IN directives**
List of Python [info](#) dictionaries (list)

11
12
13

Returns:

- *rc* - [PMIX_SUCCESS](#) or a negative value corresponding to a PMIx error constant (integer)

See [PMIx_server_register_namespace](#) for description of all relevant attributes and behaviors

14 **A.5.6 Server.deregister_namespace**

15
16

Summary

Deregister a namespace.

17
PMIx v4.0

Format

Python

18

```
myserver.deregister_namespace(namespace:str)
```

Python

19
20
21
22

- IN namespace**
Python string containing the namespace (str)
- Returns: None
- See [PMIx_server_deregister_namespace](#) for details

23 **A.5.7 Server.register_client**

24
25

Summary

Register a client process with the PMIx server library.

1 **Format**
PMIx v4.0 Python

2 `rc = myserver.register_client(proc:dict, uid:integer, gid:integer)`
Python

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**
PMIx v4.0 Python

16 `myserver.deregister_client(proc:dict)`
Python

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**
PMIx v4.0 Python

2 `rc = myserver.setup_fork(proc:dict, environ:dict)`
 Python

3 **IN** `proc`
 4 Python `proc` dictionary identifying the client process (dict)

5 **INOUT** `environ`
 6 Python dictionary containing the environment to be passed to the client (dict)

7 Returns:

8 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

9 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**
PMIx v4.0 Python

14 `rc, data = myserver.dmodex_request(proc:dict)`
 Python

15 **IN** `proc`
 16 Python `proc` dictionary identifying the process whose data is requested (dict)

17 Returns:

18 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

19 • `data` - Python `byteobject` containing the returned data (dict)

20 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**

PMIx v4.0 Python

2 `rc, info = myserver.setup_application(namespace:str, directives:list)`

Python

3 **IN namespace**

4 Namespace whose setup information is being requested (str)

5 **IN directives**

6 Python list of **info** directives

7 Returns:

8 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

9 • *info* - Python list of **info** dictionaries containing the returned data (list)

10 See **PMIx_server_setup_application** for details

11 A.5.12 Server.register_attributes

12 **Summary**

13 Register host environment attribute support for a function.

14 **Format**

PMIx v4.0 Python

15 `rc = myserver.register_attributes(function:str, attrs:list)`

Python

16 **IN function**

17 Name of the function (str)

18 **IN attrs**

19 Python list of **regattr** describing the supported attributes

20 Returns:

21 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

22 See **PMIx_Register_attributes** for details

23 A.5.13 Server.setup_local_support

24 **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**

PMIx v4.0 Python

```
2 rc = myserver.setup_local_support(namespace:str, info:list)
```

Python

3 **IN namespace**
 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

11 **Summary**

12 Function by which the host environment can pass forwarded IO to the PMIx server library for
 13 distribution to its clients.

14 **Format**

PMIx v4.0 Python

```
15 rc = myserver.iof_deliver(source:dict, channel:integer,  
16                          data:dict, directives:list)
```

Python

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

29 **Summary**

30 Collect inventory of resources on a node

1 **Format**
PMIx v4.0 Python

```
2 rc, info = myserver.collect_inventory(directives:list)
```

3 **IN directives**
4 Optional Python list of **info** containing directives (list)

5 Returns:

- 6 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)
- 7 • *info* - Python list of **info** containing the returned data (list)

8 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**
PMIx v4.0 Python

```
13 rc = myserver.deliver_inventory(info:list, directives:list)
```

14 **IN info**
15 - Python list of **info** dictionaries containing the inventory data (list)

16 **IN directives**
17 Python list of **info** dictionaries containing directives (list)

18 Returns:

- 19 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

20 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**
PMIx v4.0 Python

2 `rc,proc = mytool.init(info:list)`
Python

3 **IN** `info`
4 List of Python `info` directives (list)

5 Returns:

6 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

7 • `proc` - a Python `proc` (dict)

8 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**
PMIx v4.0 Python

13 `rc = mytool.finalize()`
Python

14 Returns:

15 • `rc` - `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant (integer)

16 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
PMIx v4.0

Format

Python

2 `rc,proc = mytool.connect_to_server(info:list)`

Python

3 **IN info**

List of Python **info** dictionaries (list)

5 Returns:

6 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

7 • *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

Python

PMIx v4.0

13 `rc,id = mytool.iof_pull(sources:list, channel:integer, directives:list, cbfunc:func)`

Python

14 **IN sources**

List of Python **proc** dictionaries of processes whose IO is being requested (list)

16 **IN channel**

Python **channel** bitmask identifying IO channels to be forwarded (integer)

18 **IN directives**

List of Python **info** dictionaries describing request (list)

20 **IN cbfunc**

Python **iofcbfunc** to receive IO payloads (func)

22 Returns:

23 • *rc* - **PMIX_SUCCESS** or a negative value corresponding to a PMIx error constant (integer)

24 • *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**
PMIx v4.0 Python

```
2 rc = mytool.iof_deregister(id:integer, directives:list)
```

Python

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**
PMIx v4.0 Python

```
14 rc = mytool.iof_push(targets:list, data:dict, directives:list)
```

Python

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_Commit 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('nspace':"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 formatted to fit the document layout.

```

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

```

Python

```

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('nspace':"testnspace", 'rank':0,
4                                   uid, gid)
5
6      print("RegClient ", rc)
7      # setup the fork
8      rc = myserver.setup_fork('nspace':"testnspace", 'rank':0, env)
9      print("SetupFrk", rc)
10
11     # setup the client argv
12     args = ["/client.py"]
13     # open a subprocess with stdout and stderr
14     # as distinct pipes so we can capture their
15     # output as the process runs
16     p = subprocess.Popen(args, env=env,
17                          stdout=subprocess.PIPE, stderr=subprocess.PIPE)
18     # define storage to catch the output
19     stdout = []
20     stderr = []
21     # loop until the pipes close
22     while True:
23         reads = [p.stdout.fileno(), p.stderr.fileno()]
24         ret = select.select(reads, [], [])
25
26         stdout_done = True
27         stderr_done = True
28
29         for fd in ret[0]:
30             # if the data
31             if fd == p.stdout.fileno():
32                 read = p.stdout.readline()
33                 if read:
34                     read = read.decode('utf-8').rstrip()
35                     print('stdout: ' + read)
36                     stdout_done = False
37             elif fd == p.stderr.fileno():
38                 read = p.stderr.readline()
39                 if read:
40                     read = read.decode('utf-8').rstrip()
41                     print('stderr: ' + read)
42                     stderr_done = False
43
44         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

1 This document represents the work of many people who have contributed to the PMIx community.
2 Without the hard work and dedication of these people this document would not have been possible.
3 The sections below list some of the active participants and organizations in the various PMIx
4 standard iterations.

5 **B.1 Version 3.0**

6 The following list includes some of the active participants in the PMIx v3 standardization process.

- 7 • Ralph H. Castain, Andrew Friedley, Brandon Yates
- 8 • Joshua Hursey
- 9 • Aurelien Bouteiller and George Bosilca
- 10 • Dirk Schubert
- 11 • Kevin Harms

12 The following institutions supported this effort through time and travel support for the people listed
13 above.

- 14 • Intel Corporation
- 15 • IBM, Inc.
- 16 • University of Tennessee, Knoxville
- 17 • The Exascale Computing Project, an initiative of the US Department of Energy
- 18 • National Science Foundation
- 19 • Argonne National Laboratory
- 20 • Allinea (ARM)

1 **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.

Index

General terms and other items not induced in the other indices.

application, [10](#), [13](#), [16](#), [41](#), [99](#), [165](#), [169](#), [344](#)

client, [7](#)

fabric, [17](#)

fabric plane, [17](#), [254](#)

fabric planes, [17](#), [244](#)

fabrics, [17](#)

host environment, [17](#)

job, [10](#), [11](#), [13](#), [16](#), [41](#), [43](#), [99](#), [160](#), [164](#), [165](#), [167](#), [169](#), [177](#), [179](#), [344](#), [345](#), [431](#)

namespace, [16](#)

rank, [16](#), [43](#), [169](#)

resource manager, [17](#)

scheduler, [17](#), [253](#)

server, [7](#)

session, [10](#), [11](#), [13](#), [16](#), [41](#), [99](#), [164](#), [344](#)

slot, [16](#)

slots, [16](#)

tool, [7](#)

workflow, [16](#)

workflows, [16](#), [375](#)

Index of APIs

PMIx_Abort, [9](#), [62](#), [63](#), [190](#), [191](#), [299](#), [391](#), [405](#)
 PMIxClient.abort (Python), [404](#)

PMIx_Alloc_directive_string, [10](#), [380](#), [424](#)
 PMIxClient.alloc_directive_string (Python), [424](#)

PMIx_Allocation_request, [12–14](#), [100](#), [101](#), [106](#), [357](#)

PMIx_Allocation_request_nb, [10](#), [88](#), [104](#), [107](#), [357](#), [413](#)
 PMIxClient.allocate (Python), [412](#)

PMIx_Commit, [9](#), [33](#), [34](#), [46](#), [46](#), [176](#), [196](#), [373](#), [406](#)
 PMIxClient.commit (Python), [406](#)

PMIx_Connect, [9](#), [10](#), [67](#), [73](#), [75](#), [77](#), [79](#), [260–262](#), [291](#), [410](#)
 PMIxClient.connect (Python), [409](#)

PMIx_Connect_nb, [9](#), [75](#), [75](#)

pmix_connection_cbfunc_t, [217](#), [373](#)

pmix_credential_cbfunc_t, [147](#), [231](#), [374](#)

PMIx_Data_copy, [10](#), [141](#)

PMIx_Data_copy_payload, [10](#), [142](#)

PMIx_Data_pack, [10](#), [137](#), [138](#), [157](#), [158](#)

PMIx_Data_print, [10](#), [141](#)

PMIx_Data_range_string, [10](#), [379](#), [423](#)
 PMIxClient.data_range_string (Python), [423](#)

PMIx_Data_type_string, [10](#), [379](#), [424](#)
 PMIxClient.data_type_string (Python), [423](#)

PMIx_Data_unpack, [10](#), [139](#)

PMIx_Deregister_event_handler, [10](#), [14](#), [129](#), [418](#)
 PMIxClient.deregister_event_handler (Python), [418](#)

PMIx_Disconnect, [9](#), [10](#), [75](#), [77](#), [79](#), [81](#), [262](#), [291](#), [410](#)
 PMIxClient.disconnect (Python), [410](#)

PMIx_Disconnect_nb, [9](#), [79](#), [81](#), [262](#)

pmix_dmodex_response_fn_t, [176](#), [372](#)

PMIx_Error_string, [9](#), [378](#), [421](#)
 PMIxClient.error_string (Python), [421](#)

pmix_event_notification_cbfunc_fn_t, [369](#), [369](#), [370](#)

pmix_evhdr_reg_cbfunc_t, [127](#), [368](#), [368](#)

PMIx_Fabric_deregister, [255](#), [420](#)
 PMIxClient.fabric_deregister (Python), [420](#)

PMIx_Fabric_get_index, [258](#), [421](#)
 PMIxClient.fabric_get_index (Python), [421](#)

PMIx_Fabric_get_vertex_info, [255](#), [420](#)

- PMIxClient.fabric_get_vertex_info (Python), [420](#)
- PMIx_Fabric_register, [247](#), [253](#), [419](#)
 - PMIxClient.fabric_register (Python), [419](#)
- PMIx_Fabric_update, [254](#), [255](#), [420](#)
 - PMIxClient.fabric_update (Python), [419](#)
- PMIx_Fence, [3](#), [6](#), [9](#), [15](#), [33](#), [46](#), [48](#), [50](#), [75](#), [79](#), [156](#), [176](#), [191](#), [194](#), [260](#), [266](#), [271](#), [391](#), [407](#)
 - PMIxClient.fence (Python), [406](#)
- PMIx_Fence_nb, [9](#), [13](#), [48](#), [50](#), [191](#), [194](#), [366](#), [391](#)
- PMIx_Finalize, [9](#), [25](#), [25](#), [26](#), [72](#), [189](#), [190](#), [291](#), [299](#), [348](#), [390](#), [404](#)
 - PMIxClient.finalize (Python), [404](#)
- PMIx_generate_ppn, [9](#), [158](#), [428](#)
 - PMIxServer.generate_ppn (Python), [428](#)
- PMIx_generate_regex, [9](#), [157](#), [158](#), [164](#), [428](#)
 - PMIxServer.generate_regex (Python), [427](#)
- PMIx_Get, [3](#), [9–11](#), [25](#), [34](#), [35](#), [37](#), [39–43](#), [45](#), [65–67](#), [69–72](#), [94](#), [99](#), [100](#), [159](#), [162](#), [185](#), [204–206](#), [244](#), [251](#), [259](#), [266](#), [292](#), [305](#), [337–341](#), [345](#), [347–354](#), [356](#), [357](#), [359](#), [360](#), [407](#)
 - PMIxClient.get (Python), [407](#)
- PMIx_Get_attribute_name, [380](#), [426](#)
 - PMIxClient.get_attribute_name (Python), [425](#)
- PMIx_Get_attribute_string, [380](#), [425](#)
 - PMIxClient.get_attribute_string (Python), [425](#)
- PMIx_Get_credential, [12](#), [14](#), [145](#), [232](#), [361](#), [414](#)
 - PMIxClient.get_credential (Python), [414](#)
- PMIx_Get_credential_nb, [146](#)
- PMIx_Get_nb, [9](#), [19](#), [37](#), [367](#)
- PMIx_Get_version, [9](#), [19](#), [21](#), [404](#)
 - PMIxClient.get_version (Python), [404](#)
- PMIx_Group_construct, [261](#), [263](#), [265](#), [266](#), [269](#), [415](#)
 - PMIxClient.group_construct (Python), [415](#)
- PMIx_Group_construct_nb, [266](#), [269](#)
- PMIx_Group_destruct, [262](#), [269](#), [271](#), [273](#), [417](#)
 - PMIxClient.group_destruct (Python), [417](#)
- PMIx_Group_destruct_nb, [271](#), [273](#)
- PMIx_Group_invite, [261](#), [273](#), [276](#), [277](#), [279](#), [416](#)
 - PMIxClient.group_invite (Python), [415](#)
- PMIx_Group_invite_nb, [277](#)
- PMIx_Group_join, [261](#), [276](#), [279](#), [279](#), [281](#), [282](#), [284](#), [328](#), [416](#)
 - PMIxClient.group_join (Python), [416](#)
- PMIx_Group_join_nb, [279](#), [282](#), [284](#)
- PMIx_Group_leave, [262](#), [284](#), [285](#), [286](#), [417](#)
 - PMIxClient.group_leave (Python), [417](#)
- PMIx_Group_leave_nb, [285](#)
- pmix_hdlr_reg_cbfunc_t, [82](#), [84](#), [377](#)

pmix_info_cbfunc_t, 95, 104, 110, 113, 116, 117, 184, 218, 223, 225, 228, 239, 241, 267, 277, 283, 364, **367**, 367
 PMIx_Info_directives_string, 10, **379**, 423
 PMIxClient.info_directives_string (Python), 423
 PMIx_Init, 10, 21, **23**, 25, 66, 70, 188, 205, 352, 356, 403
 PMIxClient.init (Python), 403
 PMIx_Initialized, 9, **21**, 403
 PMIxClient.initialized (Python), 403
 pmix_iof_cbfunc_t, 82, **376**, 389
 PMIx_IOF_channel_string, 12, **380**, 424
 PMIxClient.iof_channel_string (Python), 424
 PMIx_IOF_deregister, 12, 14, **84**, 437
 PMIxTool.iof_deregister (Python), 436
 PMIx_IOF_pull, 12, 14, **82**, 84, 436
 PMIxTool.iof_pull (Python), 436
 PMIx_IOF_push, 12, 14, **85**, 437
 PMIxTool.iof_push (Python), 437
 PMIx_Job_control, 11, 14, **107**, 109, 112, 113, 227, 359
 PMIx_Job_control_nb, 10, 88, 107, **110**, 164, 359, 413
 PMIxClient.job_ctrl (Python), 413
 PMIx_Job_state_string, **380**, 425
 PMIxClient.job_state_string (Python), 425
 PMIx_Link_state_string, **381**, 426
 PMIxClient.link_state_string (Python), 426
 PMIx_Log, 11, **118**, 120, 412
 PMIxClient.log (Python), 412
 PMIx_Log_nb, 10, **120**, 123, 354
 PMIx_Lookup, 9, 51, **54**, 57, 58, 318, 407, 408
 PMIxClient.lookup (Python), 408
 pmix_lookup_cbfunc_t, 199, **366**, 366
 PMIx_Lookup_nb, **57**, 366, 367
 pmix_modex_cbfunc_t, 192, 195, **364**, 364
 pmix_notification_fn_t, 127, **370**, 370, 390
 PMIx_Notify_event, 10, 14, **130**, 419
 PMIxClient.notify_event (Python), 418
 pmix_op_cbfunc_t, 53, 60, 76, 80, 86, 121, 130, 131, 159, 172–174, 181, 183, 185, 188, 189, 191, 196, 201, 207, 210, 212, 214, 215, 221, 226, 229, 235, 237, 272, 285, **366**, 366, 369, 372
 PMIx_Persistence_string, 10, **379**, 422
 PMIxClient.persistence_string (Python), 422
 PMIx_Proc_state_string, 10, **378**, 422
 PMIxClient.proc_state_string (Python), 421
 PMIx_Process_monitor, 12, 14, **113**, 117
 PMIx_Process_monitor_nb, 10, 88, **115**, 117, 360, 414

- PMIXClient.monitor (Python), 413
- PMIX_Publish, 9, [51](#), [52–54](#), [197](#), [198](#), [302](#), [303](#), [348](#), [408](#)
 - PMIXClient.publish (Python), 407
- PMIX_Publish_nb, 9, [53](#), [54](#)
- PMIX_Put, 9, [33](#), [33](#), [34](#), [37](#), [40](#), [46](#), [48](#), [72](#), [99](#), [176](#), [196](#), [266](#), [276](#), [301](#), [302](#), [305](#), [373](#), [406](#)
 - PMIXClient.put (Python), 405
- PMIX_Query_info, [90](#), [94](#), [99](#), [251](#)
- PMIX_Query_info_nb, [10](#), [11](#), [45](#), [72](#), [88](#), [94](#), [94](#), [99](#), [100](#), [180](#), [244](#), [259](#), [260](#), [323](#), [345](#), [353](#), [412](#)
 - PMIXClient.query (Python), 411
- PMIX_Register_attributes, [13](#), [179](#), [432](#)
 - PMIXServer.register_attributes (Python), 432
- PMIX_Register_event_handler, [10](#), [14](#), [88](#), [126](#), [369](#), [418](#)
 - PMIXClient.register_event_handler (Python), 417
- pmix_release_cbfunc_t, [364](#), [364](#)
- PMIX_Resolve_nodes, 9, [89](#), [411](#)
 - PMIXClient.resolve_nodes (Python), 411
- PMIX_Resolve_peers, 9, [88](#), [411](#)
 - PMIXClient.resolve_peers (Python), 410
- PMIX_Scope_string, [10](#), [379](#), [422](#)
 - PMIXClient.scope_string (Python), 422
- pmix_server_abort_fn_t, [190](#), [391](#)
- pmix_server_alloc_fn_t, [223](#), [398](#)
- pmix_server_client_connected_fn_t, [144](#), [174](#), [187](#), [188](#), [366](#), [390](#)
- pmix_server_client_finalized_fn_t, [189](#), [190](#), [390](#)
- PMIX_server_collect_inventory, [12](#), [183](#), [434](#)
 - PMIXServer.collect_inventory (Python), 433
- pmix_server_connect_fn_t, [72](#), [207](#), [209](#), [211](#), [394](#)
- PMIX_server_deliver_inventory, [12](#), [184](#), [434](#)
 - PMIXServer.deliver_inventory (Python), 434
- PMIX_server_deregister_client, 9, [174](#), [430](#)
 - PMIXServer.deregister_client (Python), 430
- pmix_server_deregister_events_fn_t, [213](#), [396](#)
- PMIX_server_deregister_nspace, 9, [172](#), [175](#), [429](#)
 - PMIXServer.deregister_nspace (Python), 429
- pmix_server_disconnect_fn_t, [209](#), [211](#), [395](#)
- pmix_server_dmodex_req_fn_t, [11](#), [12](#), [195](#), [364](#), [392](#)
- PMIX_server_dmodex_request, 9, [175](#), [176](#), [372](#), [373](#), [431](#)
 - PMIXServer.dmodex_request (Python), 431
- pmix_server_fabric_fn_t, [240](#), [247](#), [253](#), [402](#)
- pmix_server_fencenb_fn_t, [13](#), [191](#), [194](#), [364](#), [392](#)
- PMIX_server_finalize, 9, [156](#), [427](#)
 - PMIXServer.finalize (Python), 427
- pmix_server_get_cred_fn_t, [231](#), [234](#), [400](#)

- pmix_server_grp_fn_t, [238](#), [402](#)
- PMIx_server_init, [9](#), [21](#), [153](#), [180](#), [186](#), [250](#), [427](#)
 - PMIxServer.init (Python), [426](#)
- PMIx_server_IOF_deliver, [12](#), [81](#), [182](#), [433](#)
 - PMIxServer.iof_deliver (Python), [433](#)
- pmix_server_iof_fn_t, [234](#), [401](#)
- pmix_server_job_control_fn_t, [225](#), [399](#)
- pmix_server_listener_fn_t, [216](#)
- pmix_server_log_fn_t, [221](#), [398](#)
- pmix_server_lookup_fn_t, [198](#), [393](#)
- pmix_server_module_t, [154](#), [156](#), [180](#), [181](#), [186](#), [186](#)
- pmix_server_monitor_fn_t, [228](#), [399](#)
- pmix_server_notify_event_fn_t, [215](#), [371](#), [396](#)
- pmix_server_publish_fn_t, [196](#), [392](#)
- pmix_server_query_fn_t, [217](#), [397](#)
- PMIx_server_register_client, [9](#), [144](#), [173](#), [174](#), [188](#), [190](#), [430](#)
 - PMIxServer.register_client (Python), [429](#)
- pmix_server_register_events_fn_t, [211](#), [395](#)
- PMIx_server_register_nspace, [9](#), [11](#), [19](#), [157](#), [158](#), [164](#), [167](#), [344](#), [345](#), [366](#), [429](#)
 - PMIxServer.register_nspace (Python), [428](#)
- PMIx_server_setup_application, [10](#), [13](#), [177](#), [182](#), [185](#), [371](#), [372](#), [432](#)
 - PMIxServer.setup_application (Python), [431](#)
- PMIx_server_setup_fork, [9](#), [175](#), [431](#)
 - PMIxServer.setup_fork (Python), [430](#)
- PMIx_server_setup_local_support, [10](#), [181](#), [433](#)
 - PMIxServer.setup_local_support (Python), [432](#)
- pmix_server_spawn_fn_t, [202](#), [365](#), [394](#)
- pmix_server_stdin_fn_t, [237](#), [401](#)
- pmix_server_tool_connection_fn_t, [144](#), [220](#), [397](#)
- pmix_server_unpublish_fn_t, [200](#), [393](#)
- pmix_server_validate_cred_fn_t, [232](#), [400](#)
- pmix_setup_application_cbfunc_t, [177](#), [371](#)
- PMIx_Spawn, [9](#), [13](#), [63](#), [63](#), [64](#), [68](#), [69](#), [72](#), [104](#), [164](#), [175](#), [202](#), [207](#), [224](#), [321](#), [342](#), [351](#), [356](#), [357](#), [394](#), [409](#)
 - PMIxClient.spawn (Python), [409](#)
- pmix_spawn_cbfunc_t, [68](#), [203](#), [365](#), [365](#)
- PMIx_Spawn_nb, [9](#), [68](#), [321](#), [365](#)
- PMIx_Store_internal, [9](#), [40](#), [405](#)
 - PMIxClient.store_internal (Python), [405](#)
- PMIx_tool_connect_to_server, [12](#), [29](#), [31](#), [436](#)
 - PMIxTool.connect_to_server (Python), [435](#)
- pmix_tool_connection_cbfunc_t, [220](#), [373](#)
- PMIx_tool_finalize, [10](#), [30](#), [435](#)

PMIxTool.finalize (Python), [435](#)
PMIx_tool_init, [10](#), [21](#), [27](#), [30](#), [32](#), [338](#), [435](#)
PMIxTool.init (Python), [434](#)
PMIx_Unpublish, [9](#), [58](#), [60](#), [61](#), [409](#)
PMIxClient.unpublish (Python), [408](#)
PMIx_Unpublish_nb, [9](#), [60](#)
PMIx_Validate_credential, [12](#), [14](#), [148](#), [415](#)
PMIxClient.validate_credential (Python), [414](#)
PMIx_Validate_credential_nb, [150](#)
pmix_validation_cbfunc_t, [151](#), [233](#), [375](#)
pmix_value_cbfunc_t, [19](#), [367](#), [367](#)

Index of Support Macros

PMIX_APP_CONSTRUCT, [322](#)
PMIX_APP_CREATE, [322](#)
PMIX_APP_DESTRUCT, [322](#)
PMIX_APP_FREE, [322](#)
PMIX_APP_INFO_CREATE, [11](#), [12](#), [323](#)
PMIX_ARGV_APPEND, [331](#)
PMIX_ARGV_APPEND_UNIQUE, [332](#)
PMIX_ARGV_COPY, [334](#)
PMIX_ARGV_COUNT, [334](#)
PMIX_ARGV_FREE, [333](#)
PMIX_ARGV_JOIN, [334](#)
PMIX_ARGV_SPLIT, [333](#)
PMIX_BYTE_OBJECT_CREATE, [329](#)
PMIX_BYTE_OBJECT_DESTRUCT, [328](#)
PMIX_BYTE_OBJECT_FREE, [329](#)
PMIX_BYTE_OBJECT_LOAD, [329](#)
PMIX_CHECK_KEY, [294](#)
PMIX_CHECK_NAMESPACE, [295](#)
PMIX_CHECK_PROCID, [298](#)
PMIX_COORD_CONSTRUCT, [245](#)
PMIX_COORD_CREATE, [245](#)
PMIX_COORD_DESTRUCT, [245](#)
PMIX_COORD_FREE, [245](#)
PMIX_DATA_ARRAY_CONSTRUCT, [304](#), [330](#)
PMIX_DATA_ARRAY_CREATE, [304](#), [331](#)
PMIX_DATA_ARRAY_DESTRUCT, [304](#), [330](#)
PMIX_DATA_ARRAY_FREE, [305](#)
PMIX_DATA_ARRAY_RELEASE, [331](#)
PMIX_DATA_BUFFER_CONSTRUCT, [135](#), [138](#), [140](#)
PMIX_DATA_BUFFER_CREATE, [135](#), [138](#), [140](#)
PMIX_DATA_BUFFER_DESTRUCT, [136](#)
PMIX_DATA_BUFFER_LOAD, [136](#)
PMIX_DATA_BUFFER_RELEASE, [135](#)
PMIX_DATA_BUFFER_UNLOAD, [137](#), [157](#), [158](#)
PMIX_ENVAR_CONSTRUCT, [316](#)
PMIX_ENVAR_CREATE, [317](#)
PMIX_ENVAR_DESTRUCT, [317](#)
PMIX_ENVAR_FREE, [317](#)

PMIX_ENVAR_LOAD, [318](#)
PMIx_Heartbeat, [10](#), [117](#)
PMIX_INFO_CONSTRUCT, [310](#)
PMIX_INFO_CREATE, [310](#), [313](#), [315](#)
PMIX_INFO_DESTRUCT, [310](#)
PMIX_INFO_FREE, [311](#)
PMIX_INFO_IS_END, [11](#), [13](#), [315](#)
PMIX_INFO_IS_OPTIONAL, [315](#)
PMIX_INFO_IS_REQUIRED, [313](#), [314](#)
PMIX_INFO_LOAD, [311](#)
PMIX_INFO_OPTIONAL, [314](#)
PMIX_INFO_REQUIRED, [313](#), [314](#)
PMIX_INFO_TRUE, [312](#)
PMIX_INFO_XFER, [164](#), [312](#)
PMIX_PDATA_CONSTRUCT, [318](#)
PMIX_PDATA_CREATE, [319](#)
PMIX_PDATA_DESTRUCT, [319](#)
PMIX_PDATA_FREE, [319](#)
PMIX_PDATA_LOAD, [320](#)
PMIX_PDATA_XFER, [321](#)
PMIX_PROC_CONSTRUCT, [297](#), [328](#)
PMIX_PROC_CREATE, [297](#)
PMIX_PROC_DESTRUCT, [297](#)
PMIX_PROC_FREE, [89](#), [297](#)
PMIX_PROC_INFO_CONSTRUCT, [300](#)
PMIX_PROC_INFO_CREATE, [301](#)
PMIX_PROC_INFO_DESTRUCT, [300](#)
PMIX_PROC_INFO_FREE, [301](#)
PMIX_PROC_LOAD, [298](#)
PMIX_QUERY_CONSTRUCT, [323](#)
PMIX_QUERY_CREATE, [324](#)
PMIX_QUERY_DESTRUCT, [324](#)
PMIX_QUERY_FREE, [324](#)
PMIX_QUERY_QUALIFIERS_CREATE, [11](#), [12](#), [324](#)
PMIX_REGATTR_CONSTRUCT, [326](#)
PMIX_REGATTR_CREATE, [326](#)
PMIX_REGATTR_DESTRUCT, [326](#)
PMIX_REGATTR_FREE, [327](#)
PMIX_REGATTR_LOAD, [327](#)
PMIX_REGATTR_XFER, [327](#)
PMIX_SETENV, [335](#)
PMIX_SYSTEM_EVENT, [293](#)
PMIX_VALUE_CONSTRUCT, [306](#)

PMIX_VALUE_CREATE, [307](#)
PMIX_VALUE_DESTRUCT, [306](#)
PMIX_VALUE_FREE, [307](#)
PMIX_VALUE_GET_NUMBER, [309](#)
PMIX_VALUE_LOAD, [307](#)
PMIX_VALUE_UNLOAD, [308](#)
PMIX_VALUE_XFER, [309](#)

Index of Data Structures

pmix_alloc_directive_t, [223](#), [315](#), 315, 337, 380, 386, 424
pmix_app_t, [11](#), [12](#), [64](#), [68](#), [203](#), [321](#), 321–323, 332, 335, 387
pmix_byte_object_t, [86](#), [145](#), [149](#), [151](#), [183](#), [233](#), [237](#), [328](#), 328, 329, 336, 375, 385
pmix_coord_t, [244](#), 244, 245, 337
pmix_coord_view_t, [246](#), 251
pmix_data_array_t, [11](#), [12](#), [93](#), [98](#), [100](#), [103](#), [106](#), [167](#), [169](#), [170](#), [178](#), [219](#), [225](#), [241](#), [244](#), [249](#), [251](#),
[252](#), [254](#), [303](#), 303–305, [330](#), 330, 331, 337, 353, 354, 358, 386
pmix_data_buffer_t, [134](#), 134–139, 143
pmix_data_range_t, [131](#), [215](#), [302](#), 302, 303, 337, 379, 385, 423
pmix_data_type_t, [138](#), 140–142, 304, 307, 309, 311, 320, 327, 330, 331, [336](#), 336, 337, 379, 384,
423
pmix_envar_t, [316](#), 316–318, 337, 386
pmix_fabric_operation_t, [241](#), [247](#), 247
pmix_fabric_t, [242](#), [243](#), [247](#), 247, 250, 251, 253–256, 258
pmix_group_operation_t, [238](#), [262](#), 262, 280, 283
pmix_group_opt_t, [328](#), 328, 416
pmix_info_directives_t, [313](#), 313, 337, 379, 386, 423
pmix_info_t, [3](#), [10](#), [11](#), [13](#), [15](#), [23](#), 25–27, 52, 56, 82, 84, 86, 90, 94, 100–106, 108–110, 112–114,
117, 120, 123, 131, 145, 147, 149, 151, 154, 156, 164, 167, 169, 170, 178, 183–185, 215,
220–222, 224, 225, 228, 229, 235, 248, 249, 252, 256, 258, 263, 265, 267, 270, 272, 274,
277, 280, 283–285, 294, 302, [310](#), 310–315, 323–325, 327, 337, 344, 345, 355, 358, 359,
368–370, 376, 377, 386, 388
pmix_iof_channel_t, [82](#), [183](#), [235](#), [316](#), 316, 337, 377, 380, 386, 424
pmix_job_state_t, [302](#), 302, 380, 425
pmix_key_t, [33](#), [35](#), [294](#), 294, 327, 384
pmix_link_state_t, [246](#), 246, 250, 252, 257, 381, 387, 426
pmix_nspace_t, [295](#), 295, 298, 365, 384, 385
pmix_pdata_t, [56](#), [318](#), 318–321, 366, 367, 386
pmix_persistence_t, [303](#), 303, 337, 379, 385, 422
pmix_proc_info_t, [93](#), [98](#), [219](#), [299](#), 299–301, 337, 353, 354, 385
pmix_proc_state_t, [298](#), 298, 337, 378, 385, 421
pmix_proc_t, [23](#), [25](#), [27](#), [29](#), [32](#), [37](#), 47–49, 62, 128, 131, 132, 138, 139, 162, 173–176, 183, 188,
189, 191, 192, 195, 196, 199, 201, 203, 207, 210, 215, 217, 221, 223, 226, 229, 231, 233,
235, 237, 238, 241, 263, 267, 274, 277, 281, [296](#), 296–298, 320, 327, 336, 343, 350, 370,
374, 377, 385
pmix_query_t, [11](#), [12](#), [92](#), [97](#), [99](#), 217, 219, [323](#), 323, 324, 337, 387
pmix_rank_t, [296](#), 296, 298, 337, 385
pmix_regattr_t, [13](#), [100](#), [180](#), [325](#), 325–327, 337, 362, 387

pmix_scope_t, 34, [301](#), 301, 337, 379, 385, 422
pmix_status_t, 126, 131, 212, 214, 215, [289](#), 289, 293, 308, 309, 331, 332, 335, 336, 368–370,
372–376, 378, 384, 396, 421
pmix_value_t, 33, 34, [305](#), 305–309, 336, 367, 386

Index of Constants

PMIX_ALLOC_DIRECTIVE, [337](#)
PMIX_ALLOC_EXTEND, [315](#)
PMIX_ALLOC_EXTERNAL, [315](#)
PMIX_ALLOC_NEW, [315](#)
PMIX_ALLOC_REAQUIRE, [315](#)
PMIX_ALLOC_RELEASE, [315](#)
PMIX_APP, [336](#)
PMIX_APP_WILDCARD, [288](#)
PMIX_BOOL, [336](#)
PMIX_BUFFER, [336](#)
PMIX_BYTE, [336](#)
PMIX_BYTE_OBJECT, [336](#)
PMIX_COMMAND, [337](#)
PMIX_COMPRESSED_STRING, [337](#)
PMIX_CONNECT_REQUESTED, [291](#)
PMIX_COORD, [337](#)
PMIX_COORD_LOGICAL_VIEW, [246](#)
PMIX_COORD_PHYSICAL_VIEW, [246](#)
PMIX_COORD_VIEW_UNDEF, [246](#)
PMIX_DATA_ARRAY, [337](#)
PMIX_DATA_RANGE, [337](#)
PMIX_DATA_TYPE, [337](#)
PMIX_DOUBLE, [336](#)
PMIX_ENVAR, [337](#)
PMIX_ERR_BAD_PARAM, [290](#)
PMIX_ERR_COMM_FAILURE, [290](#)
PMIX_ERR_DATA_VALUE_NOT_FOUND, [290](#)
PMIX_ERR_DEBUGGER_RELEASE, [289](#)
PMIX_ERR_DUPLICATE_KEY, [291](#)
PMIX_ERR_EVENT_REGISTRATION, [291](#)
PMIX_ERR_GET_MALLOC_REQD, [292](#)
PMIX_ERR_HANDSHAKE_FAILED, [289](#)
PMIX_ERR_IN_ERRNO, [290](#)
PMIX_ERR_INIT, [290](#)
PMIX_ERR_INVALID_ARG, [290](#)
PMIX_ERR_INVALID_ARGS, [290](#)
PMIX_ERR_INVALID_CRED, [289](#)
PMIX_ERR_INVALID_KEY, [290](#)

PMIX_ERR_INVALID_KEY_LENGTH, [290](#)
PMIX_ERR_INVALID_KEYVALP, [290](#)
PMIX_ERR_INVALID_LENGTH, [290](#)
PMIX_ERR_INVALID_NAMESPACE, [290](#)
PMIX_ERR_INVALID_NUM_ARGS, [290](#)
PMIX_ERR_INVALID_NUM_PARSED, [290](#)
PMIX_ERR_INVALID_OPERATION, [291](#)
PMIX_ERR_INVALID_SIZE, [290](#)
PMIX_ERR_INVALID_TERMINATION, [291](#)
PMIX_ERR_INVALID_VAL, [290](#)
PMIX_ERR_INVALID_VAL_LENGTH, [290](#)
PMIX_ERR_IOF_COMPLETE, [292](#)
PMIX_ERR_IOF_FAILURE, [292](#)
PMIX_ERR_JOB_TERMINATED, [291](#)
PMIX_ERR_LOST_CONNECTION_TO_CLIENT, [290](#)
PMIX_ERR_LOST_CONNECTION_TO_SERVER, [290](#)
PMIX_ERR_LOST_PEER_CONNECTION, [290](#)
PMIX_ERR_NO_PERMISSIONS, [290](#)
PMIX_ERR_NODE_DOWN, [293](#)
PMIX_ERR_NODE_OFFLINE, [293](#)
PMIX_ERR_NOMEM, [290](#)
PMIX_ERR_NOT_FOUND, [290](#)
PMIX_ERR_NOT_IMPLEMENTED, [290](#)
PMIX_ERR_NOT_SUPPORTED, [290](#)
PMIX_ERR_OUT_OF_RESOURCE, [290](#)
PMIX_ERR_PACK_FAILURE, [290](#)
PMIX_ERR_PACK_MISMATCH, [290](#)
PMIX_ERR_PARTIAL_SUCCESS, [291](#)
PMIX_ERR_PROC_ABORTED, [289](#)
PMIX_ERR_PROC_ABORTING, [289](#)
PMIX_ERR_PROC_CHECKPOINT, [289](#)
PMIX_ERR_PROC_ENTRY_NOT_FOUND, [289](#)
PMIX_ERR_PROC_MIGRATE, [289](#)
PMIX_ERR_PROC_REQUESTED_ABORT, [289](#)
PMIX_ERR_PROC_RESTART, [289](#)
PMIX_ERR_READY_FOR_HANDSHAKE, [289](#)
PMIX_ERR_REPEAT_ATTR_REGISTRATION, [292](#)
PMIX_ERR_RESOURCE_BUSY, [290](#)
PMIX_ERR_SERVER_FAILED_REQUEST, [289](#)
PMIX_ERR_SERVER_NOT_AVAIL, [290](#)
PMIX_ERR_SILENT, [289](#)
PMIX_ERR_SYS_BASE, [292](#)
PMIX_ERR_SYS_OTHER, [293](#)

PMIX_ERR_TIMEOUT, [290](#)
PMIX_ERR_TYPE_MISMATCH, [289](#)
PMIX_ERR_UNKNOWN_DATA_TYPE, [289](#)
PMIX_ERR_UNPACK_FAILURE, [290](#)
PMIX_ERR_UNPACK_INADEQUATE_SPACE, [289](#)
PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER, [290](#)
PMIX_ERR_UNREACH, [290](#)
PMIX_ERR_UPDATE_ENDPOINTS, [291](#)
PMIX_ERR_WOULD_BLOCK, [289](#)
PMIX_ERROR, [289](#)
PMIX_EVENT_ACTION_COMPLETE, [293](#)
PMIX_EVENT_ACTION_DEFERRED, [293](#)
PMIX_EVENT_NO_ACTION_TAKEN, [293](#)
PMIX_EVENT_PARTIAL_ACTION_TAKEN, [293](#)
PMIX_EXISTS, [289](#)
PMIX_EXTERNAL_ERR_BASE, [293](#)
PMIX_FABRIC_COORDS_UPDATED, [243](#)
PMIX_FABRIC_GET_DEVICE_INDEX, [247](#)
PMIX_FABRIC_GET_VERTEX_INFO, [247](#)
PMIX_FABRIC_REQUEST_INFO, [247](#)
PMIX_FABRIC_UPDATE_INFO, [247](#)
PMIX_FABRIC_UPDATE_PENDING, [243](#)
PMIX_FABRIC_UPDATED, [243](#)
PMIX_FLOAT, [336](#)
PMIX_FWD_ALL_CHANNELS, [316](#)
PMIX_FWD_NO_CHANNELS, [316](#)
PMIX_FWD_STDDIAG_CHANNEL, [316](#)
PMIX_FWD_STDERR_CHANNEL, [316](#)
PMIX_FWD_STDIN_CHANNEL, [316](#)
PMIX_FWD_STDOUT_CHANNEL, [316](#)
PMIX_GDS_ACTION_COMPLETE, [291](#)
PMIX_GLOBAL, [301](#)
PMIX_GROUP_ACCEPT, [262](#), [328](#)
PMIX_GROUP_CONSTRUCT, [262](#)
PMIX_GROUP_CONSTRUCT_ABORT, [292](#)
PMIX_GROUP_CONSTRUCT_COMPLETE, [292](#)
PMIX_GROUP_CONTEXT_ID_ASSIGNED, [292](#)
PMIX_GROUP_DECLINE, [262](#), [328](#)
PMIX_GROUP_DESTRUCT, [262](#)
PMIX_GROUP_INVITE_ACCEPTED, [292](#)
PMIX_GROUP_INVITE_DECLINED, [292](#)
PMIX_GROUP_INVITE_FAILED, [292](#)
PMIX_GROUP_INVITED, [291](#)

PMIX_GROUP_LEADER_FAILED, [292](#)
PMIX_GROUP_LEADER_SELECTED, [292](#)
PMIX_GROUP_LEFT, [292](#)
PMIX_GROUP_MEMBER_FAILED, [292](#)
PMIX_GROUP_MEMBERSHIP_UPDATE, [292](#)
PMIX_INFO, [336](#)
PMIX_INFO_ARRAY_END, [313](#)
PMIX_INFO_DIRECTIVES, [337](#)
PMIX_INFO_REQD, [313](#)
PMIX_INT, [336](#)
PMIX_INT16, [336](#)
PMIX_INT32, [336](#)
PMIX_INT64, [336](#)
PMIX_INT8, [336](#)
PMIX_INTERNAL, [301](#)
PMIX_IOF_CHANNEL, [337](#)
PMIX_JCTRL_CHECKPOINT, [290](#)
PMIX_JCTRL_CHECKPOINT_COMPLETE, [290](#)
PMIX_JCTRL_PREEMPT_ALERT, [290](#)
PMIX_JOB_STATE_CONNECTED, [302](#)
PMIX_JOB_STATE_LAUNCH_UNDERWAY, [302](#)
PMIX_JOB_STATE_PREPPED, [302](#)
PMIX_JOB_STATE_RUNNING, [302](#)
PMIX_JOB_STATE_SUSPENDED, [302](#)
PMIX_JOB_STATE_TERMINATED, [302](#)
PMIX_JOB_STATE_TERMINATED_WITH_ERROR, [302](#)
PMIX_JOB_STATE_UNDEF, [302](#)
PMIX_JOB_STATE_UNTERMINATED, [302](#)
PMIX_KVAL, [336](#)
PMIX_LAUNCH_COMPLETE, [291](#)
PMIX_LAUNCH_DIRECTIVE, [291](#)
PMIX_LAUNCHER_READY, [291](#)
PMIX_LINK_DOWN, [246](#)
PMIX_LINK_STATE_UNKNOWN, [246](#)
PMIX_LINK_UP, [246](#)
PMIX_LOCAL, [301](#)
PMIX_MAX_KEYLEN, [288](#)
PMIX_MAX_NSLEN, [288](#)
PMIX_MODEL_DECLARED, [291](#)
PMIX_MODEL_RESOURCES, [291](#)
PMIX_MONITOR_FILE_ALERT, [291](#)
PMIX_MONITOR_HEARTBEAT_ALERT, [291](#)
PMIX_NOTIFY_ALLOC_COMPLETE, [290](#)

PMIX_OPENMP_PARALLEL_ENTERED, [291](#)
PMIX_OPENMP_PARALLEL_EXITED, [291](#)
PMIX_OPERATION_IN_PROGRESS, [291](#)
PMIX_OPERATION_SUCCEEDED, [291](#)
PMIX_PDATA, [336](#)
PMIX_PERSIST, [337](#)
PMIX_PERSIST_APP, [303](#)
PMIX_PERSIST_FIRST_READ, [303](#)
PMIX_PERSIST_INDEF, [303](#)
PMIX_PERSIST_INVALID, [303](#)
PMIX_PERSIST_PROC, [303](#)
PMIX_PERSIST_SESSION, [303](#)
PMIX_PID, [336](#)
PMIX_POINTER, [337](#)
PMIX_PROC, [336](#)
PMIX_PROC_HAS_CONNECTED, [291](#)
PMIX_PROC_INFO, [337](#)
PMIX_PROC_RANK, [337](#)
PMIX_PROC_STATE, [337](#)
PMIX_PROC_STATE_ABORTED, [299](#)
PMIX_PROC_STATE_ABORTED_BY_SIG, [299](#)
PMIX_PROC_STATE_CALLED_ABORT, [299](#)
PMIX_PROC_STATE_CANNOT_RESTART, [299](#)
PMIX_PROC_STATE_COMM_FAILED, [299](#)
PMIX_PROC_STATE_CONNECTED, [299](#)
PMIX_PROC_STATE_ERROR, [299](#)
PMIX_PROC_STATE_FAILED_TO_LAUNCH, [299](#)
PMIX_PROC_STATE_FAILED_TO_START, [299](#)
PMIX_PROC_STATE_KILLED_BY_CMD, [299](#)
PMIX_PROC_STATE_LAUNCH_UNDERWAY, [299](#)
PMIX_PROC_STATE_MIGRATING, [299](#)
PMIX_PROC_STATE_PREPPED, [299](#)
PMIX_PROC_STATE_RESTART, [299](#)
PMIX_PROC_STATE_RUNNING, [299](#)
PMIX_PROC_STATE_TERM_NON_ZERO, [299](#)
PMIX_PROC_STATE_TERM_WO_SYNC, [299](#)
PMIX_PROC_STATE_TERMINATE, [299](#)
PMIX_PROC_STATE_TERMINATED, [299](#)
PMIX_PROC_STATE_UNDEF, [299](#)
PMIX_PROC_STATE_UNTERMINATED, [299](#)
PMIX_PROC_TERMINATED, [291](#)
PMIX_QUERY, [337](#)
PMIX_QUERY_PARTIAL_SUCCESS, [290](#)

PMIX_RANGE_CUSTOM, [302](#)
PMIX_RANGE_GLOBAL, [302](#)
PMIX_RANGE_INVALID, [303](#)
PMIX_RANGE_LOCAL, [302](#)
PMIX_RANGE_NAMESPACE, [302](#)
PMIX_RANGE_PROC_LOCAL, [303](#)
PMIX_RANGE_RM, [302](#)
PMIX_RANGE_SESSION, [302](#)
PMIX_RANGE_UNDEF, [302](#)
PMIX_RANK_INVALID, [296](#)
PMIX_RANK_LOCAL_NODE, [296](#)
PMIX_RANK_LOCAL_PEERS, [296](#)
PMIX_RANK_UNDEF, [296](#)
PMIX_RANK_VALID, [296](#)
PMIX_RANK_WILDCARD, [296](#)
PMIX_REGATTR, [337](#)
PMIX_REGEX, [337](#)
PMIX_REMOTE, [301](#)
PMIX_SCOPE, [337](#)
PMIX_SCOPE_UNDEF, [301](#)
PMIX_SIZE, [336](#)
PMIX_STATUS, [336](#)
PMIX_STRING, [336](#)
PMIX_SUCCESS, [289](#)
PMIX_TIME, [336](#)
PMIX_TIMEVAL, [336](#)
PMIX_UINT, [336](#)
PMIX_UINT16, [336](#)
PMIX_UINT32, [336](#)
PMIX_UINT64, [336](#)
PMIX_UINT8, [336](#)
PMIX_UNDEF, [336](#)
PMIX_VALUE, [336](#)

Index of Attributes

PMIX_ADD_ENVAR, [357](#)
PMIX_ADD_HOST, [65](#), [70](#), [204](#), [351](#)
PMIX_ADD_HOSTFILE, [65](#), [70](#), [204](#), [351](#)
PMIX_ALLOC_BANDWIDTH, [103](#), [106](#), [178](#), [225](#), [358](#), [358](#)
PMIX_ALLOC_CPU_LIST, [102](#), [105](#), [224](#), [357](#)
PMIX_ALLOC_FABRIC, [102](#), [105](#), [178](#), [224](#), [357](#), [357](#)
PMIX_ALLOC_FABRIC_ENDPTS, [102](#), [103](#), [105](#), [106](#), [178](#), [224](#), [358](#), [358](#)
PMIX_ALLOC_FABRIC_ENDPTS_NODE, [103](#), [106](#), [178](#), [358](#), [358](#)
PMIX_ALLOC_FABRIC_ID, [102](#), [105](#), [178](#), [224](#), [358](#), [358](#)
PMIX_ALLOC_FABRIC_PLANE, [103](#), [106](#), [178](#), [225](#), [358](#), [358](#)
PMIX_ALLOC_FABRIC_QOS, [103](#), [106](#), [178](#), [225](#), [358](#), [358](#)
PMIX_ALLOC_FABRIC_SEC_KEY, [103](#), [106](#), [178](#), [225](#), [358](#), [359](#), [359](#)
PMIX_ALLOC_FABRIC_TYPE, [102](#), [103](#), [105](#), [106](#), [178](#), [224](#), [225](#), [358](#), [358](#)
PMIX_ALLOC_ID, [13](#), [104](#), [224](#), [357](#)
PMIX_ALLOC_MEM_SIZE, [102](#), [105](#), [224](#), [357](#)
PMIX_ALLOC_NETWORK (**Deprecated**), [357](#)
PMIX_ALLOC_NETWORK_ENDPTS (**Deprecated**), [358](#)
PMIX_ALLOC_NETWORK_ENDPTS_NODE (**Deprecated**), [358](#)
PMIX_ALLOC_NETWORK_ID (**Deprecated**), [358](#)
PMIX_ALLOC_NETWORK_PLANE (**Deprecated**), [358](#)
PMIX_ALLOC_NETWORK_QOS (**Deprecated**), [358](#)
PMIX_ALLOC_NETWORK_SEC_KEY (**Deprecated**), [359](#)
PMIX_ALLOC_NETWORK_TYPE (**Deprecated**), [358](#)
PMIX_ALLOC_NODE_LIST, [102](#), [105](#), [224](#), [357](#)
PMIX_ALLOC_NUM_CPU_LIST, [102](#), [105](#), [224](#), [357](#)
PMIX_ALLOC_NUM_CPUS, [102](#), [105](#), [224](#), [357](#)
PMIX_ALLOC_NUM_NODES, [102](#), [105](#), [224](#), [357](#)
PMIX_ALLOC_REQ_ID, [13](#), [102](#), [105](#), [357](#)
PMIX_ALLOC_TIME, [102](#), [105](#), [178](#), [224](#), [358](#)
PMIX_ALLOCATED_NODELIST, [162](#), [343](#)
PMIX_ANL_MAP, [162](#), [349](#)
PMIX_APP_INFO, [36](#), [39](#), [43](#), [91](#), [96](#), [344](#)
PMIX_APP_INFO_ARRAY, [169](#), [345](#), [345](#)
PMIX_APP_MAP_REGEX, [349](#)
PMIX_APP_MAP_TYPE, [349](#)
PMIX_APP_RANK, [161](#), [342](#)
PMIX_APP_SIZE, [43](#), [161](#), [169](#), [346](#)
PMIX_APPEND_ENVAR, [357](#)

PMIX_APPLDR, [161](#), [169](#), [343](#)
PMIX_APPNUM, [36](#), [39](#), [43](#), [91](#), [96](#), [161](#), [169](#), [342](#), [344](#), [345](#)
PMIX_ARCH, [342](#)
PMIX_ATTR_UNDEF, [337](#)
PMIX_AVAIL_PHYS_MEMORY, [162](#), [346](#)
PMIX_BINDTO, [66](#), [70](#), [162](#), [205](#), [352](#)
PMIX_CLEANUP_EMPTY, [109](#), [111](#), [360](#)
PMIX_CLEANUP_IGNORE, [109](#), [111](#), [360](#)
PMIX_CLEANUP_LEAVE_TOPDIR, [109](#), [112](#), [360](#)
PMIX_CLEANUP_RECURSIVE, [109](#), [111](#), [360](#)
PMIX_CLIENT_ATTRIBUTES, [14](#), [91](#), [96](#), [362](#)
PMIX_CLIENT_AVG_MEMORY, [346](#)
PMIX_CLIENT_FUNCTIONS, [362](#)
PMIX_CLUSTER_ID, [342](#)
PMIX_COLLECT_DATA, [47](#), [49](#), [193](#), [347](#)
PMIX_COLLECTIVE_ALGO, [10](#), [47](#), [50](#), [74](#), [77](#), [193](#), [208](#), [348](#)
PMIX_COLLECTIVE_ALGO_REQD, [47](#), [50](#), [74](#), [77](#), [193](#), [208](#)
PMIX_COLLECTIVE_ALGO_REQD (Deprecated), [348](#)
PMIX_CONNECT_MAX_RETRIES, [28](#), [339](#)
PMIX_CONNECT_RETRY_DELAY, [28](#), [339](#)
PMIX_CONNECT_SYSTEM_FIRST, [28](#), [30](#), [31](#), [339](#)
PMIX_CONNECT_TO_SYSTEM, [28](#), [30](#), [31](#), [338](#)
PMIX_COSPAWN_APP, [352](#)
PMIX_CPU_LIST, [67](#), [71](#), [206](#), [353](#)
PMIX_CPUS_PER_PROC, [66](#), [71](#), [206](#), [353](#)
PMIX_CPUSET, [341](#)
PMIX_CRED_TYPE, [232](#), [361](#)
PMIX_CREDENTIAL, [341](#)
PMIX_CRYPT_KEY, [361](#)
PMIX_DAEMON_MEMORY, [346](#)
PMIX_DATA_SCOPE, [35](#), [39](#), [348](#)
PMIX_DEBUG_APP_DIRECTIVES, [356](#)
PMIX_DEBUG_JOB, [356](#)
PMIX_DEBUG_JOB_DIRECTIVES, [356](#)
PMIX_DEBUG_STOP_IN_INIT, [356](#)
PMIX_DEBUG_STOP_ON_EXEC, [356](#)
PMIX_DEBUG_WAIT_FOR_NOTIFY, [356](#)
PMIX_DEBUG_WAITING_FOR_NOTIFY, [356](#)
PMIX_DEBUGGER_DAEMONS, [66](#), [71](#), [205](#), [352](#)
PMIX_DISPLAY_MAP, [65](#), [70](#), [205](#), [351](#)
PMIX_DSTPATH, [339](#)
PMIX_EMBED_BARRIER, [26](#), [348](#)
PMIX_ENUM_VALUE, [14](#), [325](#), [362](#)

PMIX_EVENT_ACTION_TIMEOUT, 128, [351](#)
PMIX_EVENT_AFFECTED_PROC, 128, 132, [350](#)
PMIX_EVENT_AFFECTED_PROCS, 128, 132, [350](#)
PMIX_EVENT_BASE, 24, 29, 155, [338](#)
PMIX_EVENT_CUSTOM_RANGE, 128, 132, [350](#)
PMIX_EVENT_DO_NOT_CACHE, [350](#)
PMIX_EVENT_HDLR_AFTER, 127, [350](#)
PMIX_EVENT_HDLR_APPEND, 128, [350](#)
PMIX_EVENT_HDLR_BEFORE, 127, [350](#)
PMIX_EVENT_HDLR_FIRST, 127, [349](#)
PMIX_EVENT_HDLR_FIRST_IN_CATEGORY, 127, [349](#)
PMIX_EVENT_HDLR_LAST, 127, [349](#)
PMIX_EVENT_HDLR_LAST_IN_CATEGORY, 127, [350](#)
PMIX_EVENT_HDLR_NAME, 127, [349](#)
PMIX_EVENT_HDLR_PREPEND, 128, [350](#)
PMIX_EVENT_NO_TERMINATION, [351](#)
PMIX_EVENT_NON_DEFAULT, 132, [350](#)
PMIX_EVENT_PROXY, [350](#)
PMIX_EVENT_RETURN_OBJECT, 128, [350](#)
PMIX_EVENT_SILENT_TERMINATION, 128, [350](#)
PMIX_EVENT_TERMINATE_JOB, 128, [350](#)
PMIX_EVENT_TERMINATE_NODE, 128, [350](#)
PMIX_EVENT_TERMINATE_PROC, 128, [350](#)
PMIX_EVENT_TERMINATE_SESSION, 128, [350](#)
PMIX_EVENT_TEXT_MESSAGE, [350](#)
PMIX_EVENT_WANT_TERMINATION, [351](#)
PMIX_EXIT_CODE, [343](#)
PMIX_FABRIC_COORDINATE, [251](#)
PMIX_FABRIC_COST_MATRIX, 248, [250](#)
PMIX_FABRIC_DEVICE, 249, [252](#)
PMIX_FABRIC_DEVICE_ADDRESS, 249, [252](#), 257
PMIX_FABRIC_DEVICE_BUS_TYPE, 250, [252](#), 256
PMIX_FABRIC_DEVICE_DRIVER, 249, [252](#), 257
PMIX_FABRIC_DEVICE_FIRMWARE, 249, [252](#), 257
PMIX_FABRIC_DEVICE_ID, 249, [252](#), 257
PMIX_FABRIC_DEVICE_INDEX, 247, [252](#)
PMIX_FABRIC_DEVICE_MTU, 250, [252](#), 257
PMIX_FABRIC_DEVICE_NAME, 249, [252](#), 256
PMIX_FABRIC_DEVICE_PCI_DEVID, 250, [252](#), 252, 256–258
PMIX_FABRIC_DEVICE_SPEED, 250, [252](#), 257
PMIX_FABRIC_DEVICE_STATE, 250, [252](#), 257
PMIX_FABRIC_DEVICE_TYPE, 250, [252](#), 257
PMIX_FABRIC_DEVICE_VENDOR, 249, [252](#), 256

PMIX_FABRIC_DIMS, 248, [251](#)
 PMIX_FABRIC_ENDPT, [251](#)
 PMIX_FABRIC_GROUPS, 248, [250](#)
 PMIX_FABRIC_IDENTIFIER, 241, 248, [251](#), 254
 PMIX_FABRIC_INDEX, 247, [251](#)
 PMIX_FABRIC_NUM_VERTICES, 248, [251](#)
 PMIX_FABRIC_PLANE, 241, 244, 249, [251](#), 254
 PMIX_FABRIC_SHAPE, 249, [251](#)
 PMIX_FABRIC_SHAPE_STRING, 249, [251](#)
 PMIX_FABRIC_VENDOR, 241, 248, [251](#), 254
 PMIX_FABRIC_VIEW, [251](#)
 PMIX_FWD_STDDIAG, 12, [352](#)
 PMIX_FWD_STDERR, 66, 71, 205, 220, [352](#)
 PMIX_FWD_STDIN, 66, 71, 205, 220, [352](#)
 PMIX_FWD_STDOUT, 66, 71, 205, 220, [352](#)
 PMIX_GDS_MODULE, 25, 29, 156, [341](#)
 PMIX_GET_STATIC_VALUES, 36, 39, [348](#)
 PMIX_GLOBAL_RANK, 163, [342](#)
 PMIX_GROUP_ASSIGN_CONTEXT_ID, 239, 240, 264, 268, 275, 278, [363](#)
 PMIX_GROUP_CONTEXT_ID, 240, [363](#)
 PMIX_GROUP_ENDPT_DATA, 239, 240, [363](#)
 PMIX_GROUP_ID, 240, [363](#)
 PMIX_GROUP_INVITE_DECLINE, [363](#)
 PMIX_GROUP_LEADER, 264, 265, 268, 277, 282, [363](#)
 PMIX_GROUP_LOCAL_ONLY, 239, 264, 268, [363](#)
 PMIX_GROUP_MEMBERSHIP, 240, 265, [363](#)
 PMIX_GROUP_NOTIFY_TERMINATION, 264–266, 268, 271, 275, 278, [363](#)
 PMIX_GROUP_OPTIONAL, 239, 264, 266, 268, 274, 278, [363](#)
 PMIX_GRPID, 51, 53, 55, 57, 59, 61, 92, 97, 102, 105, 108, 111, 114, 116, 118, 121, 146, 147,
 149, 151, 197–200, 202, 204, 213, 218, 220, 222, 224, 227, 229, 231, 234, 235, 238, [339](#),
 376
 PMIX_HOST, 65, 69, 204, [351](#)
 PMIX_HOST_ATTRIBUTES, 14, 92, 96, 100, [362](#)
 PMIX_HOST_FUNCTIONS, [362](#)
 PMIX_HOSTFILE, 65, 69, 204, [351](#)
 PMIX_HOSTNAME, 36, 39, 45, 91, 96, 161, 163, 249, 250, 252, 256–258, [343](#), 343–345
 PMIX_HWLOC_HOLE_KIND, [347](#)
 PMIX_HWLOC_SHARE_TOPO, [347](#)
 PMIX_HWLOC_SHMEM_ADDR, [347](#)
 PMIX_HWLOC_SHMEM_FILE, [347](#)
 PMIX_HWLOC_SHMEM_SIZE, [347](#)
 PMIX_HWLOC_XML_V1, 162, [347](#)
 PMIX_HWLOC_XML_V2, 162, [347](#)

PMIX_IMMEDIATE, [35](#), [39](#), [347](#)
PMIX_INDEX_ARGV, [66](#), [71](#), [206](#), [352](#)
PMIX_IOF_BUFFERING_SIZE, [83](#), [87](#), [236](#), [361](#)
PMIX_IOF_BUFFERING_TIME, [83](#), [87](#), [236](#), [361](#)
PMIX_IOF_CACHE_SIZE, [83](#), [87](#), [236](#), [361](#)
PMIX_IOF_COMPLETE, [361](#), [377](#), [389](#)
PMIX_IOF_DROP_NEWEST, [83](#), [87](#), [236](#), [361](#)
PMIX_IOF_DROP_OLDEST, [83](#), [87](#), [236](#), [361](#)
PMIX_IOF_TAG_OUTPUT, [83](#), [361](#)
PMIX_IOF_TIMESTAMP_OUTPUT, [83](#), [361](#)
PMIX_IOF_XML_OUTPUT, [84](#), [361](#)
PMIX_JOB_CONTINUOUS, [67](#), [71](#), [206](#), [353](#)
PMIX_JOB_CTRL_CANCEL, [109](#), [112](#), [227](#), [359](#)
PMIX_JOB_CTRL_CHECKPOINT, [109](#), [112](#), [227](#), [359](#)
PMIX_JOB_CTRL_CHECKPOINT_EVENT, [109](#), [112](#), [227](#), [359](#)
PMIX_JOB_CTRL_CHECKPOINT_METHOD, [109](#), [112](#), [228](#), [359](#)
PMIX_JOB_CTRL_CHECKPOINT_SIGNAL, [109](#), [112](#), [227](#), [359](#)
PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT, [109](#), [112](#), [227](#), [359](#)
PMIX_JOB_CTRL_ID, [108](#), [109](#), [111](#), [112](#), [227](#), [359](#), [359](#)
PMIX_JOB_CTRL_KILL, [108](#), [111](#), [227](#), [359](#)
PMIX_JOB_CTRL_PAUSE, [108](#), [111](#), [227](#), [359](#)
PMIX_JOB_CTRL_PREEMPTIBLE, [109](#), [112](#), [228](#), [359](#)
PMIX_JOB_CTRL_PROVISION, [109](#), [112](#), [228](#), [359](#)
PMIX_JOB_CTRL_PROVISION_IMAGE, [109](#), [112](#), [228](#), [359](#)
PMIX_JOB_CTRL_RESTART, [109](#), [112](#), [227](#), [359](#)
PMIX_JOB_CTRL_RESUME, [108](#), [111](#), [227](#), [359](#)
PMIX_JOB_CTRL_SIGNAL, [108](#), [111](#), [227](#), [359](#)
PMIX_JOB_CTRL_TERMINATE, [108](#), [111](#), [227](#), [360](#)
PMIX_JOB_INFO, [35](#), [39](#), [43](#), [91](#), [96](#), [344](#)
PMIX_JOB_INFO_ARRAY, [11](#), [167](#), [344](#), [345](#)
PMIX_JOB_NUM_APPS, [43](#), [162](#), [167](#), [346](#)
PMIX_JOB_RECOVERABLE, [67](#), [71](#), [206](#), [353](#)
PMIX_JOB_SIZE, [11](#), [13](#), [37](#), [40](#), [43](#), [160](#), [167](#), [168](#), [345](#)
PMIX_JOB_TERM_STATUS, [348](#)
PMIX_JOBID, [36](#), [39](#), [43](#), [91](#), [96](#), [160](#), [167](#), [342](#), [344](#), [345](#)
PMIX_LAUNCHER, [26](#), [339](#)
PMIX_LOCAL_CPUSSETS, [160](#), [171](#), [343](#)
PMIX_LOCAL_PEERS, [160](#), [170](#), [343](#)
PMIX_LOCAL_PROCS, [162](#), [343](#)
PMIX_LOCAL_RANK, [91](#), [92](#), [96](#), [97](#), [160](#), [342](#)
PMIX_LOCAL_SIZE, [160](#), [346](#)
PMIX_LOCAL_TOPO, [346](#)
PMIX_LOCALITY, [343](#)

PMIX_LOCALITY_STRING, [347](#)
PMIX_LOCALLDR, [162](#), [343](#)
PMIX_LOG_EMAIL, [120](#), [123](#), [222](#), [355](#)
PMIX_LOG_EMAIL_ADDR, [120](#), [123](#), [222](#), [355](#)
PMIX_LOG_EMAIL_MSG, [120](#), [123](#), [222](#), [355](#)
PMIX_LOG_EMAIL_SENDER_ADDR, [355](#)
PMIX_LOG_EMAIL_SERVER, [355](#)
PMIX_LOG_EMAIL_SRVR_PORT, [355](#)
PMIX_LOG_EMAIL_SUBJECT, [120](#), [123](#), [222](#), [355](#)
PMIX_LOG_GENERATE_TIMESTAMP, [119](#), [122](#), [355](#)
PMIX_LOG_GLOBAL_DATASTORE, [120](#), [123](#), [355](#)
PMIX_LOG_GLOBAL_SYSLOG, [119](#), [122](#), [355](#)
PMIX_LOG_JOB_RECORD, [120](#), [123](#), [355](#)
PMIX_LOG_LOCAL_SYSLOG, [119](#), [122](#), [355](#)
PMIX_LOG_MSG, [222](#), [355](#)
PMIX_LOG_ONCE, [119](#), [122](#), [355](#)
PMIX_LOG_SOURCE, [119](#), [122](#), [354](#)
PMIX_LOG_STDERR, [119](#), [122](#), [222](#), [354](#)
PMIX_LOG_STDOUT, [119](#), [122](#), [222](#), [354](#)
PMIX_LOG_SYSLOG, [119](#), [122](#), [222](#), [354](#)
PMIX_LOG_SYSLOG_PRI, [119](#), [122](#), [355](#)
PMIX_LOG_TAG_OUTPUT, [119](#), [122](#), [355](#)
PMIX_LOG_TIMESTAMP, [119](#), [122](#), [355](#)
PMIX_LOG_TIMESTAMP_OUTPUT, [119](#), [122](#), [355](#)
PMIX_LOG_XML_OUTPUT, [119](#), [122](#), [355](#)
PMIX_MAP_BLOB, [349](#)
PMIX_MAPBY, [65](#), [70](#), [162](#), [205](#), [351](#)
PMIX_MAPPER, [65](#), [70](#), [205](#), [351](#), [351](#)
PMIX_MAX_PROCS, [13](#), [45](#), [160](#), [325](#), [345](#), [346](#), [346](#)
PMIX_MAX_RESTARTS, [67](#), [72](#), [206](#), [353](#)
PMIX_MAX_VALUE, [14](#), [325](#), [362](#)
PMIX_MERGE_STDERR_STDOUT, [66](#), [71](#), [206](#), [352](#)
PMIX_MIN_VALUE, [14](#), [325](#), [362](#)
PMIX_MODEL_AFFINITY_POLICY, [340](#)
PMIX_MODEL_CPU_TYPE, [340](#)
PMIX_MODEL_LIBRARY_NAME, [163](#), [179](#), [340](#)
PMIX_MODEL_LIBRARY_VERSION, [163](#), [179](#), [340](#)
PMIX_MODEL_NUM_CPUS, [340](#)
PMIX_MODEL_NUM_THREADS, [340](#)
PMIX_MODEL_PHASE_NAME, [340](#)
PMIX_MODEL_PHASE_TYPE, [340](#)
PMIX_MONITOR_APP_CONTROL, [114](#), [116](#), [230](#), [360](#)
PMIX_MONITOR_CANCEL, [114](#), [116](#), [230](#), [360](#)

PMIX_MONITOR_FILE, [114](#), [115](#), [117](#), [230](#), [360](#)
PMIX_MONITOR_FILE_ACCESS, [114](#), [117](#), [230](#), [360](#)
PMIX_MONITOR_FILE_CHECK_TIME, [115](#), [117](#), [230](#), [361](#)
PMIX_MONITOR_FILE_DROPS, [115](#), [117](#), [230](#), [361](#)
PMIX_MONITOR_FILE_MODIFY, [115](#), [117](#), [230](#), [360](#)
PMIX_MONITOR_FILE_SIZE, [114](#), [117](#), [230](#), [360](#)
PMIX_MONITOR_HEARTBEAT, [114](#), [116](#), [230](#), [360](#)
PMIX_MONITOR_HEARTBEAT_DROPS, [114](#), [117](#), [230](#), [360](#)
PMIX_MONITOR_HEARTBEAT_TIME, [114](#), [116](#), [230](#), [360](#)
PMIX_MONITOR_ID, [114](#), [116](#), [230](#), [360](#)
PMIX_NO_OVERSUBSCRIBE, [67](#), [71](#), [206](#), [353](#)
PMIX_NO_PROCS_ON_HEAD, [67](#), [71](#), [206](#), [353](#)
PMIX_NODE_INFO, [36](#), [39](#), [45](#), [91](#), [96](#), [344](#)
PMIX_NODE_INFO_ARRAY, [168](#), [170](#), [345](#), [345](#)
PMIX_NODE_LIST, [343](#)
PMIX_NODE_MAP, [13](#), [160](#), [167–169](#), [179](#), [349](#)
PMIX_NODE_RANK, [161](#), [342](#)
PMIX_NODE_SIZE, [45](#), [162](#), [346](#)
PMIX_NODEID, [36](#), [39](#), [45](#), [91](#), [96](#), [161](#), [250](#), [252](#), [256–258](#), [343](#), [343–345](#)
PMIX_NON_PMI, [66](#), [70](#), [205](#), [352](#)
PMIX_NOTIFY_COMPLETION, [67](#), [348](#)
PMIX_NPROC_OFFSET, [162](#), [342](#)
PMIX_NSDIR, [342](#), [342](#)
PMIX_NSPACE, [36](#), [39](#), [43](#), [91](#), [92](#), [96](#), [97](#), [168](#), [342](#), [344](#), [345](#)
PMIX_NUM_NODES, [37](#), [40](#), [41](#), [43](#), [167](#), [168](#), [346](#)
PMIX_NUM_SLOTS, [346](#)
PMIX_OPTIONAL, [35](#), [38](#), [348](#)
PMIX_OUTPUT_TO_FILE, [66](#), [71](#), [206](#), [352](#)
PMIX_PARENT_ID, [64](#), [69](#), [204](#), [343](#)
PMIX_PERSISTENCE, [52](#), [54](#), [197](#), [348](#)
PMIX_PERSONALITY, [65](#), [70](#), [205](#), [351](#)
PMIX_PPR, [65](#), [70](#), [205](#), [351](#)
PMIX_PREFIX, [65](#), [69](#), [204](#), [351](#)
PMIX_PRELOAD_BIN, [65](#), [70](#), [204](#), [352](#)
PMIX_PRELOAD_FILES, [65](#), [70](#), [204](#), [352](#)
PMIX_PREPEND_ENVAR, [357](#)
PMIX_PROC_BLOB, [349](#)
PMIX_PROC_DATA, [169](#), [349](#)
PMIX_PROC_MAP, [13](#), [160](#), [167](#), [168](#), [179](#), [349](#)
PMIX_PROC_PID, [343](#)
PMIX_PROC_STATE_STATUS, [348](#)
PMIX_PROC_TERM_STATUS, [348](#)
PMIX_PROC_URI, [94](#), [98](#), [343](#)

PMIX_PROCDIR, [342](#)
 PMIX_PROCID, [91](#), [92](#), [96](#), [97](#), [163](#), [342](#)
 PMIX_PROGRAMMING_MODEL, [163](#), [179](#), [340](#)
 PMIX_PSET_NAME, [259](#), [339](#)
 PMIX_QUERY_ALLOC_STATUS, [93](#), [98](#), [219](#), [354](#)
 PMIX_QUERY_ATTRIBUTE_SUPPORT, [91](#), [96](#), [99](#), [354](#)
 PMIX_QUERY_AUTHORIZATIONS, [354](#)
 PMIX_QUERY_DEBUG_SUPPORT, [93](#), [98](#), [219](#), [354](#)
 PMIX_QUERY_JOB_STATUS, [93](#), [98](#), [218](#), [353](#)
 PMIX_QUERY_LOCAL_ONLY, [219](#), [354](#)
 PMIX_QUERY_LOCAL_PROC_TABLE, [93](#), [98](#), [219](#), [354](#)
 PMIX_QUERY_MEMORY_USAGE, [93](#), [98](#), [219](#), [354](#)
 PMIX_QUERY_NAMESPACES, [93](#), [97](#), [218](#), [353](#)
 PMIX_QUERY_NUM_PSETS, [354](#)
 PMIX_QUERY_PROC_TABLE, [93](#), [98](#), [219](#), [353](#)
 PMIX_QUERY_PSET_NAMES, [354](#)
 PMIX_QUERY_QUEUE_LIST, [93](#), [98](#), [218](#), [353](#)
 PMIX_QUERY_QUEUE_STATUS, [93](#), [98](#), [218](#), [353](#)
 PMIX_QUERY_REFRESH_CACHE, [90](#), [94](#), [95](#), [99](#), [353](#)
 PMIX_QUERY_REPORT_AVG, [93](#), [98](#), [219](#), [354](#)
 PMIX_QUERY_REPORT_MINMAX, [93](#), [98](#), [219](#), [354](#)
 PMIX_QUERY_SPAWN_SUPPORT, [93](#), [98](#), [219](#), [354](#)
 PMIX_RANGE, [52](#), [54](#), [55](#), [58](#), [59](#), [61](#), [115](#), [128](#), [197](#), [199](#), [202](#), [216](#), [240](#), [348](#)
 PMIX_RANK, [91](#), [92](#), [96](#), [97](#), [160](#), [342](#)
 PMIX_RANKBY, [66](#), [70](#), [162](#), [205](#), [352](#)
 PMIX_RECONNECT_SERVER, [339](#)
 PMIX_REGISTER_CLEANUP, [108](#), [111](#), [360](#)
 PMIX_REGISTER_CLEANUP_DIR, [108](#), [111](#), [360](#)
 PMIX_REGISTER_NODATA, [159](#), [349](#)
 PMIX_REPORT_BINDINGS, [67](#), [71](#), [206](#), [353](#)
 PMIX_REQUESTOR_IS_CLIENT, [64](#), [69](#), [339](#)
 PMIX_REQUESTOR_IS_TOOL, [64](#), [69](#), [339](#)
 PMIX_RM_NAME, [356](#)
 PMIX_RM_VERSION, [356](#)
 PMIX_SEND_HEARTBEAT, [360](#)
 PMIX_SERVER_ATTRIBUTES, [14](#), [91](#), [96](#), [362](#)
 PMIX_SERVER_ENABLE_MONITORING, [338](#)
 PMIX_SERVER_FUNCTIONS, [362](#)
 PMIX_SERVER_GATEWAY, [338](#)
 PMIX_SERVER_HOSTNAME, [339](#)
 PMIX_SERVER_NAMESPACE, [31](#), [154](#), [161](#), [338](#)
 PMIX_SERVER_PIDINFO, [28](#), [29](#), [31](#), [338](#)
 PMIX_SERVER_RANK, [154](#), [161](#), [338](#)

PMIX_SERVER_REMOTE_CONNECTIONS, [155](#), [338](#)
 PMIX_SERVER_SCHEDULER, [250](#), [253](#)
 PMIX_SERVER_SYSTEM_SUPPORT, [154](#), [338](#)
 PMIX_SERVER_TMPDIR, [29](#), [30](#), [154](#), [338](#)
 PMIX_SERVER_TOOL_SUPPORT, [144](#), [154](#), [156](#), [338](#)
 PMIX_SERVER_URI, [27](#), [29](#), [31](#), [93](#), [98](#), [339](#)
 PMIX_SESSION_ID, [35](#), [39](#), [42](#), [91](#), [95](#), [161](#), [167](#), [343](#), [344](#)
 PMIX_SESSION_INFO, [35](#), [39](#), [41](#), [90](#), [95](#), [344](#)
 PMIX_SESSION_INFO_ARRAY, [11](#), [160](#), [167](#), [344](#), [345](#)
 PMIX_SET_ENVAR, [357](#)
 PMIX_SET_SESSION_CWD, [65](#), [69](#), [204](#), [352](#)
 PMIX_SETUP_APP_ALL, [177](#), [362](#)
 PMIX_SETUP_APP_ENVARS, [177](#), [362](#)
 PMIX_SETUP_APP_NONENVARS, [177](#), [362](#)
 PMIX_SINGLE_LISTENER, [24](#), [340](#)
 PMIX_SOCKET_MODE, [24](#), [28](#), [155](#), [340](#)
 PMIX_SPAWN_TOOL, [353](#)
 PMIX_SPAWNED, [64](#), [69](#), [204](#), [341](#)
 PMIX_STDIN_TGT, [66](#), [70](#), [205](#), [352](#)
 PMIX_SYSTEM_TMPDIR, [30](#), [154](#), [338](#)
 PMIX_TAG_OUTPUT, [66](#), [71](#), [205](#), [352](#)
 PMIX_TCP_DISABLE_IPV4, [24](#), [29](#), [155](#), [341](#)
 PMIX_TCP_DISABLE_IPV6, [24](#), [29](#), [155](#), [341](#)
 PMIX_TCP_IF_EXCLUDE, [24](#), [28](#), [155](#), [341](#)
 PMIX_TCP_IF_INCLUDE, [24](#), [28](#), [155](#), [341](#)
 PMIX_TCP_IPV4_PORT, [24](#), [28](#), [155](#), [341](#)
 PMIX_TCP_IPV6_PORT, [24](#), [28](#), [155](#), [341](#)
 PMIX_TCP_REPORT_URI, [24](#), [28](#), [155](#), [341](#)
 PMIX_TCP_URI, [28](#), [29](#), [341](#)
 PMIX_TDIR_RMCLEAN, [342](#)
 PMIX_THREADING_MODEL, [340](#)
 PMIX_TIME_REMAINING, [88](#), [93](#), [98](#), [219](#), [354](#)
 PMIX_TIMEOUT, [3](#), [15](#), [36](#), [37](#), [39](#), [40](#), [47](#), [48](#), [50](#), [52](#), [54–56](#), [58](#), [59](#), [61](#), [74](#), [77](#), [78](#), [80](#), [81](#), [146](#),
[148](#), [150](#), [152](#), [193](#), [196](#), [197](#), [200](#), [202](#), [206](#), [208](#), [211](#), [232](#), [234](#), [262](#), [264–266](#), [269–271](#),
[273](#), [275](#), [279](#), [281](#), [283](#), [284](#), [347](#)
 PMIX_TIMESTAMP_OUTPUT, [66](#), [71](#), [206](#), [352](#)
 PMIX_TMPDIR, [342](#), [342](#)
 PMIX_TOOL_ATTRIBUTES, [14](#), [92](#), [97](#), [362](#)
 PMIX_TOOL_DO_NOT_CONNECT, [27](#), [29](#), [32](#), [339](#)
 PMIX_TOOL_FUNCTIONS, [362](#)
 PMIX_TOOL_NAMESPACE, [27](#), [338](#)
 PMIX_TOOL_RANK, [27](#), [338](#)
 PMIX_TOPOLOGY, [346](#)

PMIX_TOPOLOGY_FILE, [346](#)
PMIX_TOPOLOGY_SIGNATURE, [347](#)
PMIX_TOPOLOGY_XML, [346](#)
PMIX_UNIV_SIZE, [11](#), [13](#), [37](#), [40](#), [41](#), [160](#), [167](#), [345](#)
PMIX_UNSET_ENVAR, [357](#)
PMIX_USERID, [51](#), [53](#), [55](#), [57](#), [59](#), [61](#), [92](#), [97](#), [102](#), [105](#), [108](#), [111](#), [114](#), [116](#), [118](#), [121](#), [146](#), [147](#),
[149](#), [151](#), [197–202](#), [204](#), [212](#), [218](#), [220](#), [222](#), [224](#), [226](#), [229](#), [231](#), [233](#), [235](#), [238](#), [339](#), [376](#)
PMIX_USOCK_DISABLE, [24](#), [155](#), [340](#)
PMIX_VERSION_INFO, [339](#)
PMIX_WAIT, [55](#), [56](#), [58](#), [199](#), [347](#)
PMIX_WDIR, [64](#), [69](#), [204](#), [351](#)