



# Process Management Interface for Exascale (PMIx) Standard

**Version 2.0 (draft)**

**November 2017**

This document describes the Process Management Interface for Exascale (PMIx) Standard, version 2.0 (draft).

**Comments:** Please send comments on the PMIx standard to the PMIx Community by subscribing to the PMIx developers mailing list at <https://groups.google.com/forum/#!forum/pmix>. Please 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 from an unsubscribed e-mail address will be ignored.

Copyright © 2017 PMIx Standard Review Board.

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

*This page intentionally left blank*

# Contents

---

<b>1. Introduction</b>	<b>1</b>
1.1. 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?	2
1.2.3. What is <i>not</i> defined in the standard?	2
1.3. PMIx Architecture Overview	3
1.3.1. The PMIx Reference Implementation	3
1.3.2. The PMIx Reference Server	3
1.4. Organization of this document	3
<b>2. PMIx Terms and Conventions</b>	<b>5</b>
2.1. Notional Conventions	5
2.2. Semantics	6
2.3. Naming Conventions	6
2.4. Procedure Conventions	6
<b>3. Data Structures and Types</b>	<b>7</b>
3.1. Constants	7
3.1.1. Reserved attributes	7
3.1.2. Process state Definitions	23
3.1.3. Error Constants	24
3.2. Data Types	27
3.2.1. Packing Types	27
3.2.2. <code>pmix_scope_t</code>	28
3.2.3. <code>pmix_data_range_t</code>	29
3.2.4. <code>pmix_persistence_t</code>	30
3.2.5. <code>pmix_info_directives_t</code>	30
3.2.6. <code>pmix_alloc_directive_t</code>	31

3.3.	Data Packing/Unpacking . . . . .	31
3.3.1.	Byte Object . . . . .	31
3.4.	Data Structures . . . . .	32
3.4.1.	Process Structure . . . . .	32
3.4.2.	Process Info Structure . . . . .	33
3.4.3.	Data Array Structure . . . . .	33
3.4.4.	Value Structure . . . . .	34
3.4.5.	Info and Info Array Structures . . . . .	35
3.4.6.	Lookup Return Structure . . . . .	36
3.4.7.	App Structure . . . . .	37
3.4.8.	Query Structure . . . . .	37
3.4.9.	Modex Structure . . . . .	38
3.5.	Callback Functions . . . . .	38
3.5.1.	Release Callback Function . . . . .	38
3.5.2.	Modex Callback Function . . . . .	39
3.5.3.	Spawn Callback Function . . . . .	39
3.5.4.	Op Callback Function . . . . .	40
3.5.5.	Lookup Callback Function . . . . .	40
3.5.6.	Value Callback Function . . . . .	41
3.5.7.	Info Function . . . . .	41
3.6.	Other Support Functions . . . . .	42
3.6.1.	Unsorted Function . . . . .	42
3.6.2.	Key/Value Pair Management . . . . .	42
<b>4.</b>	<b>Initialization and Finalization</b>	<b>45</b>
4.1.	Query . . . . .	45
4.1.1.	<b>PMIx_Initialized</b> . . . . .	45
4.1.2.	<b>PMIx_Get_version</b> . . . . .	46
4.2.	Client . . . . .	46
4.2.1.	<b>PMIx_Init</b> . . . . .	46
4.2.2.	<b>PMIx_Finalize</b> . . . . .	47
4.3.	Tool . . . . .	48
4.3.1.	<b>PMIx_tool_init</b> . . . . .	48
4.3.2.	<b>PMIx_tool_finalize</b> . . . . .	49

4.4.	Server	49
4.4.1.	<b>PMIx_server_init</b>	50
4.4.2.	<b>PMIx_server_finalize</b>	50
<b>5.</b>	<b>Key/Value Management</b>	<b>52</b>
5.1.	Setting and Accessing Key/Value Pairs	52
5.1.1.	<b>PMIx_Put</b>	52
5.1.2.	<b>PMIx_Get</b>	53
5.1.3.	<b>PMIx_Get_nb</b>	54
5.1.4.	<b>PMIx_Store_internal</b>	55
5.2.	Exchanging Key/Value Pairs	56
5.2.1.	<b>PMIx_Commit</b>	56
5.2.2.	<b>PMIx_Fence</b>	57
5.2.3.	<b>PMIx_Fence_nb</b>	58
5.3.	Publish and Lookup Data	59
5.3.1.	<b>PMIx_Publish</b>	59
5.3.2.	<b>PMIx_Publish_nb</b>	60
5.3.3.	<b>PMIx_Lookup</b>	61
5.3.4.	<b>PMIx_Lookup_nb</b>	62
5.3.5.	<b>PMIx_Unpublish</b>	63
5.3.6.	<b>PMIx_Unpublish_nb</b>	64
<b>6.</b>	<b>Process Management</b>	<b>66</b>
6.1.	Abort	66
6.1.1.	<b>PMIx_Abort</b>	66
6.2.	Process Creation	67
6.2.1.	<b>PMIx_Spawn</b>	67
6.2.2.	<b>PMIx_Spawn_nb</b>	69
6.3.	Connecting and Disconnecting Processes	69
6.3.1.	<b>PMIx_Connect</b>	70
6.3.2.	<b>PMIx_Connect_nb</b>	71
6.3.3.	<b>PMIx_Disconnect</b>	71
6.3.4.	<b>PMIx_Disconnect_nb</b>	72

6.4. Query . . . . .	73
6.4.1. <b>PMIx_Resolve_peers</b> . . . . .	73
6.4.2. <b>PMIx_Resolve_nodes</b> . . . . .	74
6.4.3. <b>PMIx_Query_info_nb</b> . . . . .	75
<b>7. Job Allocation Management</b>	<b>77</b>
7.1. Allocation Requests . . . . .	77
7.1.1. <b>PMIx_Allocation_request_nb</b> . . . . .	77
7.1.2. <b>PMIx_Job_control_nb</b> . . . . .	78
7.2. Process and Job Monitoring . . . . .	79
7.2.1. <b>PMIx_Process_monitor_nb</b> . . . . .	79
7.2.2. <b>PMIx_Heartbeat</b> . . . . .	80
<b>8. Event Notification</b>	<b>82</b>
8.1. Logging . . . . .	82
8.1.1. <b>PMIx_Log_nb</b> . . . . .	82
8.2. Notification and Management . . . . .	83
8.2.1. <b>PMIx_Register_event_handler</b> . . . . .	83
8.2.2. <b>PMIx_Deregister_event_handler</b> . . . . .	84
8.2.3. <b>PMIx_Notify_event</b> . . . . .	84
8.2.4. Event Notification Callback Function . . . . .	85
8.2.5. Notification Callback Function . . . . .	86
8.2.6. Event Handler Registration Function . . . . .	88
<b>9. Data Packing and Unpacking</b>	<b>89</b>
9.1. General Routines . . . . .	89
9.1.1. <b>PMIx_Data_pack</b> . . . . .	89
9.1.2. <b>PMIx_Data_unpack</b> . . . . .	90
9.1.3. <b>PMIx_Data_copy</b> . . . . .	93
9.1.4. <b>PMIx_Data_print</b> . . . . .	94
9.1.5. <b>PMIx_Data_copy_payload</b> . . . . .	94
<b>10. Server Specific Interfaces</b>	<b>95</b>
10.0.1. <b>PMIx_generate_regex</b> . . . . .	95
10.0.2. <b>PMIx_generate_ppn</b> . . . . .	95

10.0.3.	<code>PMIx_server_register_nspace</code>	96
10.0.4.	<code>PMIx_server_deregister_nspace</code>	97
10.0.5.	<code>PMIx_server_register_client</code>	98
10.0.6.	<code>PMIx_server_deregister_client</code>	99
10.0.7.	<code>PMIx_server_setup_fork</code>	99
10.0.8.	<code>PMIx_server_dmodex_request</code>	100
10.0.9.	<code>PMIx_server_setup_application</code>	101
10.0.10.	<code>PMIx_server_setup_local_support</code>	102
10.1.	Server Function Pointers	102
10.1.1.	<code>pmix_server_module_t</code> Module	103
10.1.2.	<code>pmix_server_client_connected_fn_t</code>	104
10.1.3.	<code>pmix_server_client_finalized_fn_t</code>	104
10.1.4.	<code>pmix_server_abort_fn_t</code>	105
10.1.5.	<code>pmix_server_fence_nb_fn_t</code>	106
10.1.6.	<code>pmix_server_dmodex_req_fn_t</code>	107
10.1.7.	<code>pmix_server_publish_fn_t</code>	108
10.1.8.	<code>pmix_server_lookup_fn_t</code>	109
10.1.9.	<code>pmix_server_unpublish_fn_t</code>	110
10.1.10.	<code>pmix_server_spawn_fn_t</code>	111
10.1.11.	<code>pmix_server_connect_fn_t</code>	112
10.1.12.	<code>pmix_server_disconnect_fn_t</code>	113
10.1.13.	<code>pmix_server_register_events_fn_t</code>	114
10.1.14.	<code>pmix_server_deregister_events_fn_t</code>	115
10.1.15.	<code>pmix_server_notify_event_fn_t</code>	116
10.1.16.	<code>pmix_connection_cbfunc_t</code>	117
10.1.17.	<code>pmix_server_listener_fn_t</code>	118
10.1.18.	<code>pmix_server_query_fn_t</code>	118
10.1.19.	<code>pmix_tool_connection_cbfunc_t</code>	119
10.1.20.	<code>pmix_server_tool_connection_fn_t</code>	120
10.1.21.	<code>pmix_server_log_fn_t</code>	121
10.1.22.	<code>pmix_server_alloc_fn_t</code>	122
10.1.23.	<code>pmix_server_job_control_fn_t</code>	123
10.1.24.	<code>pmix_server_monitor_fn_t</code>	123

<b>A. Document Revision History</b>	<b>125</b>
A.1. Version 2.0: Date TBD . . . . .	125
A.2. Version 1.0: ad hoc release . . . . .	125
<b>B. Acknowledgements</b>	<b>126</b>
B.1. Version 2.0 . . . . .	126
B.2. Version 1.0 . . . . .	126
<b>Bibliography</b>	<b>127</b>
<b>Index</b>	<b>128</b>



## CHAPTER 1

# Introduction

---

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

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

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

- 27 a) augment those APIs to eliminate some current restrictions that impact scalability,
- 28 b) extend the breadth of the PMI definitions to providing an abstraction layer for SMS interactions,
- 29 c) establish a standards-like body for maintaining the definitions, and
- 30 d) provide a reference implementation of the PMIx standard that demonstrates the desired level of  
31 scalability and features.

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

## 1 1.1 Charter

2 The charter of the PMIx community is to:

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

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

## 14 1.2 PMIx Standard Overview

15 ...

### 16 1.2.1 Who should use the standard?

17 ...

### 18 1.2.2 What is defined in the standard?

19 ...

### 20 1.2.3 What is *not* defined in the standard?

21 The PMIx Standard does not include anything, either stated or implied, regarding implementation.  
22 It instead focuses exclusively on defining APIs and associated attribute key strings, and describing  
23 the expected behavior of those entities. How that behavior is realized is entirely at the discretion of  
24 the implementer.

25 As previously noted, system environments and PMIx library implementers are free to return “not  
26 supported” for any request. Thus, users should design their applications accordingly.

## 1 1.3 PMIx Architecture Overview

2 This section presents a brief overview the PMIx Architecture [1].

3 ...

### 4 1.3.1 The PMIx Reference Implementation

5 Note that the definition of the PMIx Standard is not contingent upon use of the PMIx Reference  
6 Implementation. Any implementation that supports the defined APIs is a PMIx Standard compliant  
7 implementation, and some environments have chosen to pursue their own custom implementation.  
8 The PMIx Reference Implementation is provided solely for the following purposes:

- 9 • Validation of the standard.  
10 No proposed change and/or extension to the PMIx standard is accepted without an accompanying  
11 prototype implementation in the PMIx Reference Implementation. This ensures that the proposal  
12 has undergone at least some minimal level of scrutiny and testing before being considered.
- 13 • Ease of adoption.  
14 The PMIx Reference Implementation is designed to be particularly easy for resource managers  
15 (and the SMS in general) to adopt, thus facilitating a rapid uptake into that community for  
16 application portability. Both client and server PMIx libraries are included, along with examples  
17 of client usage and server-side integration. A list of supported environments and versions is  
18 provided on the PMIx web site [www.pmix.org](http://www.pmix.org)

19 The PMIx Reference Implementation targets support for the Linux operating system. A reasonable  
20 effort is made to support all major, modern Linux distributions; however, validation is limited to the  
21 most recent 2-3 releases of RedHat Enterprise Linux (RHEL), Fedora, CentOS, and SUSE Linux  
22 Enterprise Server (SLES). In addition, development support is maintained for Mac OSX.  
23 Production support for vendor-specific operating systems is included as provided by the vendor.

### 24 1.3.2 The PMIx Reference Server

25 ...

## 26 1.4 Organization of this document

27 The remainder of this document is structured as follows:

- 28 • Introduction and Overview in Chapter 1 on page 1
- 29 • Terms and Conventions in Chapter 2 on page 5

- 1           • Data Structures and Types in Chapter 3 on page 7
- 2           • PMIx Initialization and Finalization in Chapter 4 on page 45
- 3           • Key/Value Management in Chapter 5 on page 52
- 4           • Process Management in Chapter 6 on page 66
- 5           • Job Management in Chapter 7 on page 77
- 6           • Event Notification in Chapter 8 on page 82
- 7           • Data Packing and Unpacking in Chapter 9 on page 89
- 8           • PMIx Server Specific Interfaces in Chapter 10 on page 95

## CHAPTER 2

# PMIx Terms and Conventions

---

1 Define “attributes” and how they are used, intent is to allow for definition of flexible APIs that can  
2 change behavior based on attributes instead of modifying function signature. Include description of  
3 data types.

4 This document borrows freely from other standards (most notably from the Message Passing  
5 Interface (MPI) and OpenMP standards) in its use of notation and conventions in an attempt to  
6 reduce confusion.

## 7 2.1 Notional Conventions

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

▼  C 

10 C specific text...

11 `int foo = 42;`

▲  C 

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

▼ 

14 Note: General text...

▲ 

▼  Rationale 

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

## Advice to implementers

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

## 2.2 Semantics

8 The following terms will be taken to mean:

- 9 • *shall* and *will* indicate that the specified behavior is *required* of all conforming implementations
- 10 • *should* and *may* indicate behaviors that a quality implementation would include, but are not  
11 required of all conforming implementations

## 2.3 Naming Conventions

13 ...

## 2.4 Procedure Conventions

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

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

## CHAPTER 3

# Data Structures and Types

---

1 ...

## 2 3.1 Constants

3 The constant defined in this section may be used before calling any library initialization routine.

4 **PMIX\_MAX\_NSLEN**

5 (integer) Maximum namespace string length

6 **PMIX\_MAX\_KEYLEN**

7 (integer) Maximum key string length

8 The `pmix_rank_t` structure is an integer type for rank values.

9 The following constants can be used to set the `pmix_rank_t`.

10 **PMIX\_RANK\_UNDEF**

11 Define a value for requests for job-level data where the info itself is not associated with any  
12 specific rank, or when a request involves a rank that is not known. For example, when  
13 someone requests info thru one of the legacy interfaces where the rank is typically encoded  
14 into the key itself since there is no rank parameter in the API itself.

15 **PMIX\_RANK\_WILDCARD**

16 Define a value to indicate that the user wants the data for the given key from every rank that  
17 posted that key.

18 **PMIX\_RANK\_LOCAL\_NODE**

19 Special rank value used to define groups of ranks for use in collectives. All ranks on a local  
20 node.

### 21 3.1.1 Reserved attributes

22 Define a set of “standard” PMIx attributes that can be queried. Implementations (and users) are free  
23 to extend as desired, so the get functions need to be capable of handling the “not found” condition.  
24 Note that these are attributes of the system and the job as opposed to values the application (or  
25 underlying MPI library) might choose to expose - i.e., they are values provided by the resource  
26 manager as opposed to the application. Thus, these keys are **RESERVED**.

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

28 Constant representing an undefined attribute.

### 1 3.1.1.1 Initialization attributes

2 ...

3 **PMIX\_EVENT\_BASE** "pmix.evbase" (struct event\_base \*)  
4 Pointer to libevent event\_base to use in place of the internal progress thread.

5 **PMIX\_SERVER\_TOOL\_SUPPORT** "pmix.srvr.tool" (bool)  
6 The host RM wants to declare itself as willing to accept tool connection requests.

7 **PMIX\_SERVER\_REMOTE\_CONNECTIONS** "pmix.srvr.remote" (bool)  
8 Allow connections from remote tools (do not use loopback device).

9 **PMIX\_SERVER\_SYSTEM\_SUPPORT** "pmix.srvr.sys" (bool)  
10 The host RM wants to declare itself as being the local system server for PMIx connection  
11 requests.

12 **PMIX\_SERVER\_TMPDIR** "pmix.srvr.tmpdir" (char\*)  
13 temp directory where PMIx server will place client rendezvous points and contact info.

14 **PMIX\_SYSTEM\_TMPDIR** "pmix.sys.tmpdir" (char\*)  
15 temp directory for this system, where PMIx server will place tool rendezvous points and  
16 contact info.

17 **PMIX\_REGISTER\_NODATA** "pmix.reg.nodata" (bool)  
18 Registration is for nspace only, do not copy job data.

19 **PMIX\_SERVER\_ENABLE\_MONITORING** "pmix.srv.monitor" (bool)  
20 Enable PMIx internal monitoring by server

21 **PMIX\_SERVER\_NAMESPACE** "pmix.srv.namespace" (char\*)  
22 Name of the nspace to use for this server.

23 **PMIX\_SERVER\_RANK** "pmix.srv.rank" (pmix\_rank\_t)  
24 Rank of this server

### 25 3.1.1.2 Tool-related attributes

26 ...

27 **PMIX\_TOOL\_NAMESPACE** "pmix.tool.namespace" (char\*)  
28 Name of the nspace to use for this tool.

29 **PMIX\_TOOL\_RANK** "pmix.tool.rank" (uint32\_t)  
30 Rank of this tool.

31 **PMIX\_SERVER\_PIDINFO** "pmix.srvr.pidinfo" (pid\_t)  
32 PID of the target server for a tool.

33 **PMIX\_CONNECT\_TO\_SYSTEM** "pmix.cnct.sys" (bool)  
34 The requestor requires that a connection be made only to a local system-level PMIx server.

35 **PMIX\_CONNECT\_SYSTEM\_FIRST** "pmix.cnct.sys.first" (bool)  
36 Preferentially look for a system-level PMIx server first.

37 **PMIX\_SERVER\_URI** "pmix.srvr.uri" (char\*)  
38 URI of server to be contacted.

39 **PMIX\_SERVER\_HOSTNAME** "pmix.srvr.host" (char\*)  
40 node where target server is located



1           **PMIX\_CONNECT\_MAX\_RETRIES** "pmix.tool.mretries" (uint32\_t)  
 2           maximum number of times to try to connect to server  
 3           **PMIX\_CONNECT\_RETRY\_DELAY** "pmix.tool.retry" (uint32\_t)  
 4           time in seconds between connection attempts  
 5           **PMIX\_TOOL\_DO\_NOT\_CONNECT** "pmix.tool.nocon" (bool)  
 6           The tool wants to use internal PMIx support, but does not want to connect to a PMIx server.

### 7 **3.1.1.3 Identification attributes**

8           ...

9           **PMIX\_USERID** "pmix.euid" (uint32\_t)  
 10           Effective user id  
 11           **PMIX\_GRPID** "pmix.egid" (uint32\_t)  
 12           Effective group id  
 13           **PMIX\_DSTPATH** "pmix.dstpath" (char\*)  
 14           Path to dstore files  
 15           **PMIX\_VERSION\_INFO** "pmix.version" (char\*)  
 16           PMIx version of contactor  
 17           **PMIX\_PROGRAMMING\_MODEL** "pmix.pgm.model" (char\*)  
 18           Programming model being initialized (e.g., "MPI" or "OpenMP")  
 19           **PMIX\_MODEL\_LIBRARY\_NAME** "pmix.mdl.name" (char\*)  
 20           Programming model implementation ID (e.g., "OpenMPI" or "MPICH")  
 21           **PMIX\_MODEL\_LIBRARY\_VERSION** "pmix.mld.vrs" (char\*)  
 22           Programming model version string (e.g., "2.1.1")  
 23           **PMIX\_THREADING\_MODEL** "pmix.threads" (char\*)  
 24           Threading model used (e.g., "pthreads")  
 25           **PMIX\_REQUESTOR\_IS\_TOOL** "pmix.req.tool" (bool)  
 26           Requesting process is a tool  
 27           **PMIX\_REQUESTOR\_IS\_CLIENT** "pmix.req.client" (bool)  
 28           Requesting process is a client process

### 29 **3.1.1.4 USOCK rendezvous socket attributes**

30           ...

31           **PMIX\_USOCK\_DISABLE** "pmix.usock.disable" (bool)  
 32           Disable legacy usock support  
 33           **PMIX\_SOCKET\_MODE** "pmix.sockmode" (uint32\_t)  
 34           POSIX *mode\_t* (9 bits valid)  
 35           **PMIX\_SINGLE\_LISTENER** "pmix.sing.listnr" (bool)  
 36           Use only one rendezvous socket, letting priorities and/or MCA param select the active  
 37           transport.

### 1 3.1.1.5 TCP connection attributes

2 ...

3 **PMIX\_TCP\_REPORT\_URI** "pmix.tcp.repuri" (char\*)  
4 output URI. '-' for stdout, '+' for stderr, or filename

5 **PMIX\_TCP\_URI** "pmix.tcp.uri" (char\*)  
6 URI of server to connect to, or file:<name of file containing it>

7 **PMIX\_TCP\_IF\_INCLUDE** "pmix.tcp.ifinclude" (char\*)  
8 Comma-delimited list of devices and/or CIDR notation

9 **PMIX\_TCP\_IF\_EXCLUDE** "pmix.tcp.ifexclude" (char\*)  
10 Comma-delimited list of devices and/or CIDR notation

11 **PMIX\_TCP\_IPV4\_PORT** "pmix.tcp.ipv4" (int)  
12 IPv4 port to be used

13 **PMIX\_TCP\_IPV6\_PORT** "pmix.tcp.ipv6" (int)  
14 IPv6 port to be used

15 **PMIX\_TCP\_DISABLE\_IPV4** "pmix.tcp.disipv4" (bool)  
16 true to disable IPv4 family

17 **PMIX\_TCP\_DISABLE\_IPV6** "pmix.tcp.disipv6" (bool)  
18 true to disable IPv6 family

### 19 3.1.1.6 GDS attributes

20 ...

21 **PMIX\_GDS\_MODULE** "pmix.gds.mod" (char\*)  
22 Comma-delimited string of desired modules

### 23 3.1.1.7 General proc-level attributes

24 ...

25 **PMIX\_CPUSET** "pmix.cpuset" (char\*)  
26 hwloc bitmap applied to proc upon launch

27 **PMIX\_CREDENTIAL** "pmix.cred" (char\*)  
28 Security credential assigned to proc

29 **PMIX\_SPAWNED** "pmix.spawned" (bool)  
30 true if this proc resulted from a call to [PMIx\\_Spawn](#)

31 **PMIX\_ARCH** "pmix.arch" (uint32\_t)  
32 datatype architecture flag

### 1 3.1.1.8 Scratch directory locations for use by applications 2 attributes

3 ...

4 **PMIX\_TMPDIR** "pmix.tmpdir" (char\*)  
5 Top-level tmp dir assigned to session

6 **PMIX\_NSDIR** "pmix.nsdire" (char\*)  
7 Sub-tmpdir assigned to namespace

8 **PMIX\_PROCDIR** "pmix.pdir" (char\*)  
9 Sub-nsdir assigned to proc

10 **PMIX\_TDIR\_RMCLEAN** "pmix.tdir.rmclean" (bool)  
11 Resource Manager will clean session directories

### 12 3.1.1.9 Information about relative ranks as assigned by the 13 RM attributes

14 ...

15 **PMIX\_PROCID** "pmix.procid" (pmix\_proc\_t)  
16 Process identifier

17 **PMIX\_NAMESPACE** "pmix.namespace" (char\*)  
18 nspace of a job

19 **PMIX\_JOBID** "pmix.jobid" (char\*)  
20 jobid assigned by scheduler

21 **PMIX\_APPNUM** "pmix.appnum" (uint32\_t)  
22 app number within the job

23 **PMIX\_RANK** "pmix.rank" (pmix\_rank\_t)  
24 process rank within the job

25 **PMIX\_GLOBAL\_RANK** "pmix.grank" (pmix\_rank\_t)  
26 rank spanning across all jobs in this session

27 **PMIX\_APP\_RANK** "pmix.apprank" (pmix\_rank\_t)  
28 rank within this app

29 **PMIX\_NPROC\_OFFSET** "pmix.offset" (pmix\_rank\_t)  
30 starting global rank of this job

31 **PMIX\_LOCAL\_RANK** "pmix.lrank" (uint16\_t)  
32 rank on this node within this job

33 **PMIX\_NODE\_RANK** "pmix.nrank" (uint16\_t)  
34 rank on this node spanning all jobs

35 **PMIX\_LOCALLDR** "pmix.lldr" (pmix\_rank\_t)  
36 lowest rank on this node within this job

37 **PMIX\_APPLDR** "pmix.aldr" (pmix\_rank\_t)  
38 lowest rank in this app within this job

39 **PMIX\_PROC\_PID** "pmix.ppid" (pid\_t)  
40 pid of specified proc

1 **PMIX\_SESSION\_ID** "pmix.session.id" (uint32\_t)  
 2 session identifier  
 3 **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)  
 4 Comma-delimited list of nodes running procs for the specified nspace  
 5 **PMIX\_ALLOCATED\_NODELIST** "pmix.alist" (char\*)  
 6 comma-delimited list of all nodes in this allocation regardless of whether or not they  
 7 currently host procs.  
 8 **PMIX\_HOSTNAME** "pmix.hname" (char\*)  
 9 Name of the host the specified proc is on  
 10 **PMIX\_NODEID** "pmix.nodeid" (uint32\_t)  
 11 Node identifier where the specified proc is located  
 12 **PMIX\_LOCAL\_PEERS** "pmix.lpeers" (char\*)  
 13 Comma-delimited string of ranks on this node within the specified nspace  
 14 **PMIX\_LOCAL\_PROCS** "pmix.lprocs" (pmix\_proc\_t array)  
 15 array of pmix\_proc\_t of procs on the specified node  
 16 **PMIX\_LOCAL\_CPUSSETS** "pmix.lcpus" (char\*)  
 17 colon-delimited cpusets of local peers within the specified nspace  
 18 **PMIX\_PROC\_URI** "pmix.puri" (char\*)  
 19 URI containing contact info for proc  
 20 **PMIX\_LOCALITY** "pmix.loc" (uint16\_t)  
 21 relative locality of two procs  
 22 **PMIX\_PARENT\_ID** "pmix.parent" (pmix\_proc\_t)  
 23 process identifier of my parent process

### 24 3.1.1.10 Size information attributes

25 ...

26 **PMIX\_UNIV\_SIZE** "pmix.univ.size" (uint32\_t)  
 27 Number of procs in this nspace  
 28 **PMIX\_JOB\_SIZE** "pmix.job.size" (uint32\_t)  
 29 Number of procs in this job  
 30 **PMIX\_JOB\_NUM\_APPS** "pmix.job.napps" (uint32\_t)  
 31 Number of apps in this job  
 32 **PMIX\_APP\_SIZE** "pmix.app.size" (uint32\_t)  
 33 Number of procs in this application  
 34 **PMIX\_LOCAL\_SIZE** "pmix.local.size" (uint32\_t)  
 35 Number of procs in this job on this node  
 36 **PMIX\_NODE\_SIZE** "pmix.node.size" (uint32\_t)  
 37 Number of procs across all jobs on this node  
 38 **PMIX\_MAX\_PROCS** "pmix.max.size" (uint32\_t)  
 39 Max number of procs for this job  
 40 **PMIX\_NUM\_NODES** "pmix.num.nodes" (uint32\_t)  
 41 Number of nodes in this nspace

### 1 3.1.1.11 Memory information attributes

2 ...

3 **PMIX\_AVAIL\_PHYS\_MEMORY** "pmix.pmem" (uint64\_t)  
4 Total available physical memory on this node  
5 **PMIX\_DAEMON\_MEMORY** "pmix.dmn.mem" (float)  
6 Mbytes of memory currently used by daemon  
7 **PMIX\_CLIENT\_AVG\_MEMORY** "pmix.cl.mem.avg" (float)  
8 Average Mbytes of memory used by client processes

### 9 3.1.1.12 Topology information attributes

10 ...

11 **PMIX\_NET\_TOPO** "pmix.ntopo" (char\*)  
12 xml-representation of network topology  
13 **PMIX\_LOCAL\_TOPO** "pmix.ltopo" (char\*)  
14 xml-representation of local node topology  
15 **PMIX\_NODE\_LIST** "pmix.nlist" (char\*)  
16 comma-delimited list of nodes running procs for this job  
17 **PMIX\_TOPOLOGY** "pmix.topo" (hwloc\_topology\_t)  
18 pointer to the PMIx client's internal topology object  
19 **PMIX\_TOPOLOGY\_SIGNATURE** "pmix.toposig" (char\*)  
20 topology signature string  
21 **PMIX\_LOCALITY\_STRING** "pmix.locstr" (char\*)  
22 string describing a proc's location  
23 **PMIX\_HWLOC\_SHMEM\_ADDR** "pmix.hwlocaddr" (size\_t)  
24 address of HWLOC shared memory segment  
25 **PMIX\_HWLOC\_SHMEM\_SIZE** "pmix.hwlocsize" (size\_t)  
26 size of HWLOC shared memory segment  
27 **PMIX\_HWLOC\_SHMEM\_FILE** "pmix.hwlocfile" (char\*)  
28 path to HWLOC shared memory file  
29 **PMIX\_HWLOC\_XML\_V1** "pmix.hwlocxml1" (char\*)  
30 XML representation of local topology using HWLOC v1.x format  
31 **PMIX\_HWLOC\_XML\_V2** "pmix.hwlocxml2" (char\*)  
32 XML representation of local topology using HWLOC v2.x format

### 1 3.1.1.13 Request-related attributes

2 ...

3 **PMIX\_COLLECT\_DATA** "pmix.collect" (bool)

4 Collect data and return it at the end of the operation

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

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

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

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

12 **PMIX\_WAIT** "pmix.wait" (int)

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

15 **PMIX\_COLLECTIVE\_ALGO** "pmix.calgo" (char\*)

16 comma-delimited list of algorithms to use for collective

17 **PMIX\_COLLECTIVE\_ALGO\_REQD** "pmix.calreqd" (bool)

18 if true, indicates that the requested choice of algo is mandatory

19 **PMIX\_NOTIFY\_COMPLETION** "pmix.notecomp" (bool)

20 notify parent process upon termination of child job

21 **PMIX\_RANGE** "pmix.range" (pmix\_data\_range\_t)

22 value for calls to publish/lookup/unpublish or for monitoring event notifications

23 **PMIX\_PERSISTENCE** "pmix.persist" (pmix\_persistence\_t)

24 value for calls to publish

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

26 Scope of the data to be found in a [PMIx\\_Get](#) call

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

28 look only in the client’s local data store for the requested value - do not request data from the  
29 server if not found

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

31 Execute a blocking fence operation before executing the specified operation. By default,  
32 [PMIx\\_Finalize](#) does not include an internal barrier operation. This attribute directs  
33 [PMIx\\_Finalize](#) to execute a barrier as part of the finalize operation.

34 **PMIX\_JOB\_TERM\_STATUS** "pmix.job.term.status" (pmix\_status\_t)

35 status returned upon job termination

36 **PMIX\_PROC\_STATE\_STATUS** "pmix.proc.state" (pmix\_proc\_state\_t)

37 process state

### 1 3.1.1.14 Sever to Convenience library attributes

2 Attributes used by host server to pass data to the server convenience library - the data will then be  
3 parsed and provided to the local clients.

4 **PMIX\_REGISTER\_NODATA** "pmix.reg.nodata" (bool)  
5 Registration is for nspace only, do not copy job data  
6 **PMIX\_PROC\_DATA** "pmix.pdata" (pmix\_data\_array\_t)  
7 starts with rank, then contains more data  
8 **PMIX\_NODE\_MAP** "pmix.nmap" (char\*)  
9 regex of nodes containing procs for this job  
10 **PMIX\_PROC\_MAP** "pmix.pmap" (char\*)  
11 regex describing procs on each node within this job  
12 **PMIX\_ANL\_MAP** "pmix.anlmap" (char\*)  
13 process mapping in ANL notation (used in PMI-1/PMI-2)  
14 **PMIX\_APP\_MAP\_TYPE** "pmix.apmap.type" (char\*)  
15 type of mapping used to layout the application (e.g., cyclic)  
16 **PMIX\_APP\_MAP\_REGEX** "pmix.apmap.regex" (char\*)  
17 regex describing the result of the mapping

### 18 3.1.1.15 Sever to Client attributes

19 Attributes used internally to communicate data from the server to the client

20 **PMIX\_PROC\_BLOB** "pmix.pblob" (pmix\_byte\_object\_t)  
21 packed blob of process data  
22 **PMIX\_MAP\_BLOB** "pmix.mblob" (pmix\_byte\_object\_t)  
23 packed blob of process location

### 24 3.1.1.16 Event handler registration and notification attributes

25 ...

26 **PMIX\_EVENT\_HDLR\_NAME** "pmix.evname" (char\*)  
27 string name identifying this handler  
28 **PMIX\_EVENT\_JOB\_LEVEL** "pmix.evjob" (bool)  
29 register for job-specific events only  
30 **PMIX\_EVENT\_ENVIRO\_LEVEL** "pmix.evenv" (bool)  
31 register for environment events only  
32 **PMIX\_EVENT\_HDLR\_FIRST** "pmix.evfirst" (bool)  
33 invoke this event handler before any other handlers  
34 **PMIX\_EVENT\_HDLR\_LAST** "pmix.evlast" (bool)  
35 invoke this event handler after all other handlers have been called  
36 **PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY** "pmix.evfirstcat" (bool)  
37 invoke this event handler before any other handlers in this category  
38 **PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY** "pmix.evlastcat" (bool)

1           invoke this event handler after all other handlers in this category have been called  
2 **PMIX\_EVENT\_HDLR\_BEFORE** "pmix.evbefore" (char\*)  
3           put this event handler immediately before the one specified in the (char\*) value  
4 **PMIX\_EVENT\_HDLR\_AFTER** "pmix.evafter" (char\*)  
5           put this event handler immediately after the one specified in the (char\*) value  
6 **PMIX\_EVENT\_HDLR\_PREPEND** "pmix.evprepend" (bool)  
7           prepend this handler to the precedence list within its category  
8 **PMIX\_EVENT\_HDLR\_APPEND** "pmix.evappend" (bool)  
9           append this handler to the precedence list within its category  
10 **PMIX\_EVENT\_CUSTOM\_RANGE** "pmix.evrangle" (pmix\_data\_array\_t\*)  
11           array of pmix\_proc\_t defining range of event notification  
12 **PMIX\_EVENT\_AFFECTED\_PROC** "pmix.evproc" (pmix\_proc\_t)  
13           single proc that was affected  
14 **PMIX\_EVENT\_AFFECTED\_PROCS** "pmix.evaffected" (pmix\_data\_array\_t\*)  
15           array of pmix\_proc\_t defining affected procs  
16 **PMIX\_EVENT\_NON\_DEFAULT** "pmix.evnondef" (bool)  
17           event is not to be delivered to default event handlers  
18 **PMIX\_EVENT\_RETURN\_OBJECT** "pmix.evobject" (void\*)  
19           object to be returned whenever the registered cbfunc is invoked. NOTE: the object will *only*  
20           be returned to the process that registered it.  
21 **PMIX\_EVENT\_DO\_NOT\_CACHE** "pmix.evnocache" (bool)  
22           instruct the PMIx server not to cache the event  
23 **PMIX\_EVENT\_SILENT\_TERMINATION** "pmix.evsilentterm" (bool)  
24           do not generate an event when this job normally terminates

### 25 3.1.1.17 Fault tolerance attributes

26 ...

27 **PMIX\_EVENT\_TERMINATE\_SESSION** "pmix.evterm.sess" (bool)  
28           RM intends to terminate session  
29 **PMIX\_EVENT\_TERMINATE\_JOB** "pmix.evterm.job" (bool)  
30           RM intends to terminate this job  
31 **PMIX\_EVENT\_TERMINATE\_NODE** "pmix.evterm.node" (bool)  
32           RM intends to terminate all procs on this node  
33 **PMIX\_EVENT\_TERMINATE\_PROC** "pmix.evterm.proc" (bool)  
34           RM intends to terminate just this process  
35 **PMIX\_EVENT\_ACTION\_TIMEOUT** "pmix.evtimeout" (int)  
36           time in sec before RM will execute error response  
37 **PMIX\_EVENT\_NO\_TERMINATION** "pmix.evnoterm" (bool)  
38           indicates that the handler has satisfactorily handled the event and believes termination of the  
39           application is not required.  
40 **PMIX\_EVENT\_WANT\_TERMINATION** "pmix.evterm" (bool)  
41           indicates that the handler has determined that the application should be terminated



### 1 3.1.1.18 Spawn attributes

2 attributes used to describe "spawn" attributes

3 **PMIX\_PERSONALITY** "pmix.pers" (char\*)

4 name of personality to use

5 **PMIX\_HOST** "pmix.host" (char\*)

6 comma-delimited list of hosts to use for spawned procs

7 **PMIX\_HOSTFILE** "pmix.hostfile" (char\*)

8 hostfile to use for spawned procs

9 **PMIX\_ADD\_HOST** "pmix.addhost" (char\*)

10 comma-delimited list of hosts to add to allocation

11 **PMIX\_ADD\_HOSTFILE** "pmix.addhostfile" (char\*)

12 hostfile to add to existing allocation

13 **PMIX\_PREFIX** "pmix.prefix" (char\*)

14 prefix to use for starting spawned procs

15 **PMIX\_WDIR** "pmix.wdir" (char\*)

16 working directory for spawned procs

17 **PMIX\_MAPPER** "pmix.mapper" (char\*)

18 mapper to use for placing spawned procs

19 **PMIX\_DISPLAY\_MAP** "pmix.dispmap" (bool)

20 display process map upon spawn

21 **PMIX\_PPR** "pmix.ppr" (char\*)

22 Number of procs to spawn on each identified resource

23 **PMIX\_MAPBY** "pmix.mapby" (char\*)

24 mapping policy

25 **PMIX\_RANKBY** "pmix.rankby" (char\*)

26 ranking policy

27 **PMIX\_BINDTO** "pmix.bindto" (char\*)

28 binding policy

29 **PMIX\_PRELOAD\_BIN** "pmix.preloadbin" (bool)

30 preload binaries

31 **PMIX\_PRELOAD\_FILES** "pmix.preloadfiles" (char\*)

32 comma-delimited list of files to pre-position

33 **PMIX\_NON\_PMI** "pmix.nonpmi" (bool)

34 spawned procs will not call **PMIx\_Init**

35 **PMIX\_STDIN\_TGT** "pmix.stdin" (uint32\_t)

36 spawned proc rank that is to receive stdin

37 **PMIX\_FWD\_STDIN** "pmix.fwd.stdin" (bool)

38 forward my stdin to the designated proc

39 **PMIX\_FWD\_STDOUT** "pmix.fwd.stdout" (bool)

40 forward stdout from spawned procs to me

41 **PMIX\_FWD\_STDERR** "pmix.fwd.stderr" (bool)

1 forward stderr from spawned procs to me  
2 **PMIX\_DEBUGGER\_DAEMONS** "pmix.debugger" (bool)  
3 spawned app consists of debugger daemons  
4 **PMIX\_COSPAWN\_APP** "pmix.cospawn" (bool)  
5 designated app is to be spawned as a disconnected job - i.e., not part of the "comm\_world" of  
6 the job  
7 **PMIX\_SET\_SESSION\_CWD** "pmix.ssn cwd" (bool)  
8 set the application's current working directory to the session working directory assigned by  
9 the RM  
10 **PMIX\_TAG\_OUTPUT** "pmix.tagout" (bool)  
11 tag application output with the ID of the source  
12 **PMIX\_TIMESTAMP\_OUTPUT** "pmix.tsout" (bool)  
13 timestamp output from applications  
14 **PMIX\_MERGE\_STDERR\_STDOUT** "pmix.mergeerrout" (bool)  
15 merge stdout and stderr streams from application procs  
16 **PMIX\_OUTPUT\_TO\_FILE** "pmix.outfile" (char\*)  
17 output application output to given file  
18 **PMIX\_INDEX\_ARGV** "pmix.indxargv" (bool)  
19 mark the argv with the rank of the proc  
20 **PMIX\_CPUS\_PER\_PROC** "pmix.cpusperproc" (uint32\_t)  
21 Number of cpus to assign to each rank  
22 **PMIX\_NO\_PROCS\_ON\_HEAD** "pmix.nolocal" (bool)  
23 do not place procs on the head node  
24 **PMIX\_NO\_OVERSUBSCRIBE** "pmix.noover" (bool)  
25 do not oversubscribe the cpus  
26 **PMIX\_REPORT\_BINDINGS** "pmix.repbinding" (bool)  
27 report bindings of the individual procs  
28 **PMIX\_CPU\_LIST** "pmix.cpulist" (char\*)  
29 list of cpus to use for this job  
30 **PMIX\_JOB\_RECOVERABLE** "pmix.recover" (bool)  
31 application supports recoverable operations  
32 **PMIX\_JOB\_CONTINUOUS** "pmix.continuous" (bool)  
33 application is continuous, all failed procs should be immediately restarted.  
34 **PMIX\_MAX\_RESTARTS** "pmix.maxrestarts" (uint32\_t)  
35 max number of times to restart a job

### 36 3.1.1.19 Query attributes

37 ...  
38 **PMIX\_QUERY\_NAMESPACES** "pmix.qry.ns" (char\*)  
39 request a comma-delimited list of active namespaces  
40 **PMIX\_QUERY\_JOB\_STATUS** "pmix.qry.jst" (pmix\_status\_t)  
41 status of a specified currently executing job  
42 **PMIX\_QUERY\_QUEUE\_LIST** "pmix.qry.qlst" (char\*)

```

1         request a comma-delimited list of scheduler queues
2 PMIX_QUERY_QUEUE_STATUS "pmix.qry.qst" (TBD)
3         status of a specified scheduler queue
4 PMIX_QUERY_PROC_TABLE "pmix.qry.ptable" (char*)
5         input nspace of job whose info is being requested returns (pmix_data_array_t) an array of
6         pmix_proc_info_t.
7 PMIX_QUERY_LOCAL_PROC_TABLE "pmix.qry.lptable" (char*)
8         input nspace of job whose info is being requested returns (pmix_data_array_t) an array of
9         pmix_proc_info_t for procs in job on same node
10 PMIX_QUERY_AUTHORIZATIONS "pmix.qry.auths" (bool)
11         return operations tool is authorized to perform
12 PMIX_QUERY_SPAWN_SUPPORT "pmix.qry.spawn" (bool)
13         return a comma-delimited list of supported spawn attributes
14 PMIX_QUERY_DEBUG_SUPPORT "pmix.qry.debug" (bool)
15         return a comma-delimited list of supported debug attributes
16 PMIX_QUERY_MEMORY_USAGE "pmix.qry.mem" (bool)
17         return info on memory usage for the procs indicated in the qualifiers
18 PMIX_QUERY_LOCAL_ONLY "pmix.qry.local" (bool)
19         constrain the query to local information only
20 PMIX_QUERY_REPORT_AVG "pmix.qry.avg" (bool)
21         report average values
22 PMIX_QUERY_REPORT_MINMAX "pmix.qry.minmax" (bool)
23         report minimum and maximum value
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 nspace

```

### 28 3.1.1.20 Log attributes

```

29     ...
30 PMIX_LOG_STDERR "pmix.log.stderr" (char*)
31     log string to stderr
32 PMIX_LOG_STDOUT "pmix.log.stdout" (char*)
33     log string to stdout
34 PMIX_LOG_SYSLOG "pmix.log.syslog" (char*)
35     log data to syslog - defaults to ERROR priority unless
36 PMIX_LOG_MSG "pmix.log.msg" (pmix_byte_object_t)
37     message blob to be sent somewhere
38 PMIX_LOG_EMAIL "pmix.log.email" (pmix_data_array_t)
39     log via email based on pmix_info_t containing directives
40 PMIX_LOG_EMAIL_ADDR "pmix.log.emaddr" (char*)
41     comma-delimited list of email addresses that are to recv msg
42 PMIX_LOG_EMAIL_SUBJECT "pmix.log.emsub" (char*)

```

1 subject line for email  
2 **PMIX\_LOG\_EMAIL\_MSG** "pmix.log.emmsg" (char\*)  
3 msg to be included in email

#### 4 **3.1.1.21 Debugger attributes**

5 ...  
6 **PMIX\_DEBUG\_STOP\_ON\_EXEC** "pmix.dbg.exec" (bool)  
7 job is being spawned under debugger - instruct it to pause on start  
8 **PMIX\_DEBUG\_STOP\_IN\_INIT** "pmix.dbg.init" (bool)  
9 instruct job to stop during PMIx init  
10 **PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY** "pmix.dbg.notify" (bool)  
11 block at desired point until receiving debugger release notification  
12 **PMIX\_DEBUG\_JOB** "pmix.dbg.job" (char\*)  
13 nspace of the job to be debugged - the RM/PMIx server are  
14 **PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY** "pmix.dbg.waiting" (bool)  
15 job to be debugged is waiting for a release

#### 16 **3.1.1.22 Resource manager attributes**

17 ...  
18 **PMIX\_RM\_NAME** "pmix.rm.name" (char\*)  
19 string name of the resource manager  
20 **PMIX\_RM\_VERSION** "pmix.rm.version" (char\*)  
21 RM version string

#### 22 **3.1.1.23 Environment variable attributes**

23 ...  
24 **PMIX\_SET\_ENVAR** "pmix.set.envar" (char\*)  
25 string "key=value" value shall be put into the environment  
26 **PMIX\_UNSET\_ENVAR** "pmix.unset.envar" (char\*)  
27 unset envar specified in string

#### 28 **3.1.1.24 Job Allocation attributes**

29 ...  
30 **PMIX\_ALLOC\_ID** "pmix.alloc.id" (char\*)  
31 provide a string identifier for this allocation request which can later be used to query status of  
32 the request  
33 **PMIX\_ALLOC\_NUM\_NODES** "pmix.alloc.nnodes" (uint64\_t)  
34 number of nodes  
35 **PMIX\_ALLOC\_NODE\_LIST** "pmix.alloc.nlist" (char\*)  
36 regex of specific nodes  
37 **PMIX\_ALLOC\_NUM\_CPUS** "pmix.alloc.ncpus" (uint64\_t)

```

1         number of cpus
2     PMIX_ALLOC_NUM_CPU_LIST "pmix.alloc.ncpulist" (char*)
3         regex of the number of cpus for each node
4     PMIX_ALLOC_CPU_LIST "pmix.alloc.cpulist" (char*)
5         regex of specific cpus indicating the cpus involved.
6     PMIX_ALLOC_MEM_SIZE "pmix.alloc.msize" (float)
7         number of Mbytes
8     PMIX_ALLOC_NETWORK "pmix.alloc.net" (array)
9         array of pmix_info_t describing network resources. If not given as part of an info struct that
10        identifies the impacted nodes, then the description will be applied across all nodes in the
11        requestor's allocation
12    PMIX_ALLOC_NETWORK_ID "pmix.alloc.netid" (char*)
13        name of network
14    PMIX_ALLOC_BANDWIDTH "pmix.alloc.bw" (float)
15        Mbits/sec
16    PMIX_ALLOC_NETWORK_QOS "pmix.alloc.netqos" (char*)
17        quality of service level
18    PMIX_ALLOC_TIME "pmix.alloc.time" (uint32_t)
19        time in seconds

```

### 20 3.1.1.25 Job control attributes

```

21    ...
22    PMIX_JOB_CTRL_ID "pmix.jctrl.id" (char*)
23        provide a string identifier for this request
24    PMIX_JOB_CTRL_PAUSE "pmix.jctrl.pause" (bool)
25        pause the specified processes
26    PMIX_JOB_CTRL_RESUME "pmix.jctrl.resume" (bool)
27        "un-pause" the specified processes
28    PMIX_JOB_CTRL_CANCEL "pmix.jctrl.cancel" (char*)
29        cancel the specified request (NULL implies cancel all requests from this requestor)
30    PMIX_JOB_CTRL_KILL "pmix.jctrl.kill" (bool)
31        forcibly terminate the specified processes and cleanup
32    PMIX_JOB_CTRL_RESTART "pmix.jctrl.restart" (char*)
33        restart the specified processes using the given checkpoint ID
34    PMIX_JOB_CTRL_CHECKPOINT "pmix.jctrl.ckpt" (char*)
35        checkpoint the specified processes and assign the given ID to it
36    PMIX_JOB_CTRL_CHECKPOINT_EVENT "pmix.jctrl.ckptev" (bool)
37        use event notification to trigger process checkpoint
38    PMIX_JOB_CTRL_CHECKPOINT_SIGNAL "pmix.jctrl.ckptsig" (int)
39        use the given signal to trigger process checkpoint
40    PMIX_JOB_CTRL_CHECKPOINT_TIMEOUT "pmix.jctrl.ckptsig" (int)
41        time in seconds to wait for checkpoint to complete

```

1 **PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD**  
2 "pmix.jctrl.ckmethod" (pmix\_data\_array\_t)  
3 array of pmix\_info\_t declaring each method and value supported by this application  
4 **PMIX\_JOB\_CTRL\_SIGNAL** "pmix.jctrl.sig" (int)  
5 send given signal to specified processes  
6 **PMIX\_JOB\_CTRL\_PROVISION** "pmix.jctrl.pvn" (char\*)  
7 regex identifying nodes that are to be provisioned  
8 **PMIX\_JOB\_CTRL\_PROVISION\_IMAGE** "pmix.jctrl.pvning" (char\*)  
9 name of the image that is to be provisioned  
10 **PMIX\_JOB\_CTRL\_PREEMPTIBLE** "pmix.jctrl.preempt" (bool)  
11 job can be pre-empted  
12 **PMIX\_JOB\_CTRL\_TERMINATE** "pmix.jctrl.term" (bool)  
13 politely terminate the specified procs

### 14 3.1.1.26 Monitoring attributes

15 ...  
16 **PMIX\_MONITOR\_ID** "pmix.monitor.id" (char\*)  
17 provide a string identifier for this request  
18 **PMIX\_MONITOR\_CANCEL** "pmix.monitor.cancel" (char\*)  
19 identifier to be canceled (NULL means cancel all monitoring for this process)  
20 **PMIX\_MONITOR\_APP\_CONTROL** "pmix.monitor.appctrl" (bool)  
21 the application desires to control the response to a monitoring event  
22 **PMIX\_MONITOR\_HEARTBEAT** "pmix.monitor.mbeat" (void)  
23 register to have the server monitor the requestor for heartbeats  
24 **PMIX\_SEND\_HEARTBEAT** "pmix.monitor.beat" (void)  
25 send heartbeat to local server  
26 **PMIX\_MONITOR\_HEARTBEAT\_TIME** "pmix.monitor.btime" (uint32\_t)  
27 time in seconds before declaring heartbeat missed  
28 **PMIX\_MONITOR\_HEARTBEAT\_DROPS** "pmix.monitor.bdrop" (uint32\_t)  
29 number of heartbeats that can be missed before generating the event  
30 **PMIX\_MONITOR\_FILE** "pmix.monitor.fmon" (char\*)  
31 register to monitor file for signs of life  
32 **PMIX\_MONITOR\_FILE\_SIZE** "pmix.monitor.fsize" (bool)  
33 monitor size of given file is growing to determine app is running  
34 **PMIX\_MONITOR\_FILE\_ACCESS** "pmix.monitor.faccess" (char\*)  
35 monitor time since last access of given file to determine app is running  
36 **PMIX\_MONITOR\_FILE\_MODIFY** "pmix.monitor.fmod" (char\*)  
37 monitor time since last modified of given file to determine app is running  
38 **PMIX\_MONITOR\_FILE\_CHECK\_TIME** "pmix.monitor.ftime" (uint32\_t)  
39 time in seconds between checking file  
40 **PMIX\_MONITOR\_FILE\_DROPS** "pmix.monitor.fdrop" (uint32\_t)  
41 number of file checks that can be missed before generating the event

## 1 3.1.2 Process state Definitions

2 The `pmix_proc_state_t` structure is an unsigned integer type (`uint8_t`) for process state  
3 values.

4 **PMIX\_PROC\_STATE\_UNDEF**  
5 undefined process state

6 **PMIX\_PROC\_STATE\_PREPPED**  
7 process is ready to be launched

8 **PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY**  
9 launch process underway

10 **PMIX\_PROC\_STATE\_RESTART**  
11 the proc is ready for restart

12 **PMIX\_PROC\_STATE\_TERMINATE**  
13 process is marked for termination

14 **PMIX\_PROC\_STATE\_RUNNING**  
15 daemon has locally fork'd process

16 **PMIX\_PROC\_STATE\_CONNECTED**  
17 proc connected to PMIx server

18 **PMIX\_PROC\_STATE\_UNTERMINATED**  
19 Define a "boundary" so users can easily and quickly determine if a proc is still running or  
20 not - any value less than this one means that the proc has not terminated.

21 **PMIX\_PROC\_STATE\_TERMINATED**  
22 process has terminated and is no longer running

23 **PMIX\_PROC\_STATE\_ERROR**  
24 Define a boundary so users can easily and quickly determine if a proc abnormally terminated  
25 - leave a little room for future expansion.

26 **PMIX\_PROC\_STATE\_KILLED\_BY\_CMD**  
27 process was killed by cmd

28 **PMIX\_PROC\_STATE\_ABORTED**  
29 process aborted

30 **PMIX\_PROC\_STATE\_FAILED\_TO\_START**  
31 process failed to start

32 **PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG**  
33 process aborted by signal

34 **PMIX\_PROC\_STATE\_TERM\_WO\_SYNC**  
35 process exit'd without calling `PMIx_Finalize`

36 **PMIX\_PROC\_STATE\_COMM\_FAILED**  
37 process communication has failed

38 **PMIX\_PROC\_STATE\_CALLED\_ABORT**  
39 process called `PMIx_Abort`

40 **PMIX\_PROC\_STATE\_MIGRATING**  
41 process failed and is waiting for resources before restarting

```

1     PMIX_PROC_STATE_CANNOT_RESTART
2         process failed and cannot be restarted
3     PMIX_PROC_STATE_TERM_NON_ZERO
4         process exited with a non-zero status, indicating abnormal
5     PMIX_PROC_STATE_FAILED_TO_LAUNCH
6         unable to launch process

```

### 7 3.1.3 Error Constants

8 The `pmix_status_t` structure is an integer type for return status. The table below defines the  
9 possible values for `pmix_status_t`. PMIx errors are always negative, with 0 reserved for  
10 success.

```

11     PMIX_ERR_BASE
12         Error base

```

13 v1.x error values - must be fixed in place for backward compatability. Note that some number of  
14 these have been deprecated and may not be returned by v2.x and above clients or servers. However,  
15 they must always be at least defined to ensure older codes will compile.

▼ C ▼

```

16     //..
17     #define PMIX_SUCCESS                                0
18     #define PMIX_ERROR                                  -1           // general e
19     #define PMIX_ERR_SILENT                            -2           // internal-
20     /* debugger release flag */
21     #define PMIX_ERR_DEBUGGER_RELEASE                  -3
22     /* fault tolerance */
23     #define PMIX_ERR_PROC_RESTART                      -4
24     #define PMIX_ERR_PROC_CHECKPOINT                  -5
25     #define PMIX_ERR_PROC_MIGRATE                     -6
26     /* abort */
27     #define PMIX_ERR_PROC_ABORTED                     -7
28     #define PMIX_ERR_PROC_REQUESTED_ABORT             -8
29     #define PMIX_ERR_PROC_ABORTING                   -9
30     /* communication failures */
31     #define PMIX_ERR_SERVER_FAILED_REQUEST             -10
32     #define PMIX_EXISTS                                -11
33     #define PMIX_ERR_INVALID_CRED                      -12           // internal-
34     #define PMIX_ERR_HANDSHAKE_FAILED                 -13           // internal-
35     #define PMIX_ERR_READY_FOR_HANDSHAKE              -14           // internal-
36     #define PMIX_ERR_WOULD_BLOCK                      -15
37     #define PMIX_ERR_UNKNOWN_DATA_TYPE                -16           // internal-

```



```

1      #define PMIX_ERR_PROC_ENTRY_NOT_FOUND          -17      // intern
2      #define PMIX_ERR_TYPE_MISMATCH                -18      // intern
3      #define PMIX_ERR_UNPACK_INADEQUATE_SPACE      -19      // intern
4      #define PMIX_ERR_UNPACK_FAILURE              -20      // intern
5      #define PMIX_ERR_PACK_FAILURE                -21      // intern
6      #define PMIX_ERR_PACK_MISMATCH              -22      // intern
7      #define PMIX_ERR_NO_PERMISSIONS              -23
8      #define PMIX_ERR_TIMEOUT                    -24
9      #define PMIX_ERR_UNREACH                    -25
10     #define PMIX_ERR_IN_ERRNO                    -26      // intern
11     #define PMIX_ERR_BAD_PARAM                  -27
12     #define PMIX_ERR_RESOURCE_BUSY              -28      // intern
13     #define PMIX_ERR_OUT_OF_RESOURCE            -29
14     #define PMIX_ERR_DATA_VALUE_NOT_FOUND      -30
15     #define PMIX_ERR_INIT                      -31
16     #define PMIX_ERR_NOMEM                     -32      // intern
17     #define PMIX_ERR_INVALID_ARG               -33      // intern
18     #define PMIX_ERR_INVALID_KEY              -34      // intern
19     #define PMIX_ERR_INVALID_KEY_LENGTH        -35      // intern
20     #define PMIX_ERR_INVALID_VAL              -36      // intern
21     #define PMIX_ERR_INVALID_VAL_LENGTH       -37      // intern
22     #define PMIX_ERR_INVALID_LENGTH           -38      // intern
23     #define PMIX_ERR_INVALID_NUM_ARGS         -39      // intern
24     #define PMIX_ERR_INVALID_ARGS             -40      // intern
25     #define PMIX_ERR_INVALID_NUM_PARSED       -41      // intern
26     #define PMIX_ERR_INVALID_KEYVALP         -42      // intern
27     #define PMIX_ERR_INVALID_SIZE             -43
28     #define PMIX_ERR_INVALID_NAMESPACE         -44
29     #define PMIX_ERR_SERVER_NOT_AVAIL         -45      // intern
30     #define PMIX_ERR_NOT_FOUND                -46
31     #define PMIX_ERR_NOT_SUPPORTED            -47
32     #define PMIX_ERR_NOT_IMPLEMENTED          -48
33     #define PMIX_ERR_COMM_FAILURE             -49
34     #define PMIX_ERR_UNPACK_READ_PAST_END_OF_BUFFER -50      // intern
35
36     /* define a starting point for v2.x error values */
37     #define PMIX_ERR_V2X_BASE                  -100
38
39     /* v2.x communication errors */
40     #define PMIX_ERR_LOST_CONNECTION_TO_SERVER (PMIX_ERR_V2X_BASE - 1)
41     #define PMIX_ERR_LOST_PEER_CONNECTION    (PMIX_ERR_V2X_BASE - 2)
42     #define PMIX_ERR_LOST_CONNECTION_TO_CLIENT (PMIX_ERR_V2X_BASE - 3)
43     /* used by the query system */

```

```

1      #define PMIX_QUERY_PARTIAL_SUCCESS                (PMIX_ERR_V2X_BASE - 4)
2      /* request responses */
3      #define PMIX_NOTIFY_ALLOC_COMPLETE                (PMIX_ERR_V2X_BASE - 5)
4      /* job control */
5      #define PMIX_JCTRL_CHECKPOINT                    (PMIX_ERR_V2X_BASE - 6)
6      #define PMIX_JCTRL_CHECKPOINT_COMPLETE            (PMIX_ERR_V2X_BASE - 7)
7
8      #define PMIX_JCTRL_PREEMPT_ALERT                  (PMIX_ERR_V2X_BASE - 8)
9      /* monitoring */
10     #define PMIX_MONITOR_HEARTBEAT_ALERT              (PMIX_ERR_V2X_BASE - 9)
11     #define PMIX_MONITOR_FILE_ALERT                  (PMIX_ERR_V2X_BASE - 10)
12
13     /* define a starting point for operational error constants so
14      * we avoid renumbering when making additions */
15     #define PMIX_ERR_OP_BASE        PMIX_ERR_V2X_BASE-30
16
17     /* operational */
18     #define PMIX_ERR_EVENT_REGISTRATION                (PMIX_ERR_OP_BASE - 14)
19     #define PMIX_ERR_JOB_TERMINATED                   (PMIX_ERR_OP_BASE - 15)
20     #define PMIX_ERR_UPDATE_ENDPOINTS                 (PMIX_ERR_OP_BASE - 16)
21     #define PMIX_MODEL_DECLARED                       (PMIX_ERR_OP_BASE - 17)
22     #define PMIX_GDS_ACTION_COMPLETE                  (PMIX_ERR_OP_BASE - 18)
23
24     /* define a starting point for system error constants so
25      * we avoid renumbering when making additions */
26     #define PMIX_ERR_SYS_BASE        PMIX_ERR_OP_BASE-100
27
28     /* system failures */
29     #define PMIX_ERR_NODE_DOWN                        (PMIX_ERR_SYS_BASE - 1)
30     #define PMIX_ERR_NODE_OFFLINE                     (PMIX_ERR_SYS_BASE - 2)
31
32
33     /* define a starting point for event handler error constants so
34      * we avoid renumbering when making additions */
35     #define PMIX_ERR_EVHDLR_BASE        PMIX_ERR_SYS_BASE-100
36
37     /* used by event handlers */
38     #define PMIX_EVENT_NO_ACTION_TAKEN                 (PMIX_ERR_EVHDLR_BASE - 1)
39     #define PMIX_EVENT_PARTIAL_ACTION_TAKEN           (PMIX_ERR_EVHDLR_BASE - 2)
40     #define PMIX_EVENT_ACTION_DEFERRED                (PMIX_ERR_EVHDLR_BASE - 3)
41     #define PMIX_EVENT_ACTION_COMPLETE                (PMIX_ERR_EVHDLR_BASE - 4)
42
43     /* define a starting point for PMIx internal error codes

```

```

1      * that are never exposed outside the library */
2      #define PMIX_INTERNAL_ERR_BASE          -1000
3
4      /* define a starting point for user-level defined error
5       * constants - negative values larger than this are guaranteed
6       * not to conflict with PMIx values. Definitions should always
7       * be based on the PMIX_EXTERNAL_ERR_BASE constant and -not- a
8       * specific value as the value of the constant may change */
9      #define PMIX_EXTERNAL_ERR_BASE          -2000

```

C

## 10 3.2 Data Types

11 ...

### 12 3.2.1 Packing Types


13 The `pmix_data_type_t` structure is an integer type for defining the data type for  
14 packing/unpacking purposes. The table below defines the possible values for  
15 `pmix_data_type_t`.

	C	
16		// ...
17	#define PMIX_UNDEF	0
18	#define PMIX_BOOL	1 // converted to/from native true/false
19	#define PMIX_BYTE	2 // a byte of data
20	#define PMIX_STRING	3 // NULL-terminated string
21	#define PMIX_SIZE	4 // size_t
22	#define PMIX_PID	5 // OS-pid
23	#define PMIX_INT	6
24	#define PMIX_INT8	7
25	#define PMIX_INT16	8
26	#define PMIX_INT32	9
27	#define PMIX_INT64	10
28	#define PMIX_UINT	11
29	#define PMIX_UINT8	12
30	#define PMIX_UINT16	13
31	#define PMIX_UINT32	14
32	#define PMIX_UINT64	15
33	#define PMIX_FLOAT	16

```

1      #define PMIX_DOUBLE          17
2      #define PMIX_TIMEVAL        18
3      #define PMIX_TIME           19
4      #define PMIX_STATUS        20 // needs to be tracked separately from i
5                                     // when we are embedded and it needs to
6                                     // host error definitions
7      #define PMIX_VALUE          21
8      #define PMIX_PROC           22
9      #define PMIX_APP            23
10     #define PMIX_INFO           24
11     #define PMIX_PDATA          25
12     #define PMIX_BUFFER         26
13     #define PMIX_BYTE_OBJECT    27
14     #define PMIX_KVAL           28
15     #define PMIX_MODEX          29
16     #define PMIX_PERSIST        30
17     #define PMIX_POINTER        31
18     #define PMIX_SCOPE          32
19     #define PMIX_DATA_RANGE     33
20     #define PMIX_COMMAND        34
21     #define PMIX_INFO_DIRECTIVES 35
22     #define PMIX_DATA_TYPE      36
23     #define PMIX_PROC_STATE     37
24     #define PMIX_PROC_INFO      38
25     #define PMIX_DATA_ARRAY     39
26     #define PMIX_PROC_RANK      40
27     #define PMIX_QUERY          41
28     #define PMIX_COMPRESSED_STRING 42 // string compressed with zlib
29     #define PMIX_ALLOC_DIRECTIVE 43
30     /**** DEPRECATED ****/
31     #define PMIX_INFO_ARRAY      44
32     /*****
33
34     /* define a boundary for implementers so they can add their own data types *
35     #define PMIX_DATA_TYPE_MAX    500

```



### 36 3.2.2 pmix\_scope\_t

37 The `pmix_scope_t` structure is an integer type for defining the scope for data “put” by PMIx  
38 The table below defines the possible values for `pmix_scope_t` .

```

1  /*
2  * PMI_LOCAL - the data is intended only for other application
3  *              processes on the same node. Data marked in this way
4  *              will not be included in data packages sent to remote requests
5  * PMI_REMOTE - the data is intended solely for applications processes on
6  *              remote nodes. Data marked in this way will not be shared
7  *              with other processes on the same node
8  * PMI_GLOBAL - the data is to be shared with all other requesting processes
9  *              regardless of location
10 /*
11 #define PMIX_SCOPE_UNDEF    0
12 #define PMIX_LOCAL          1 // share to procs also on this node
13 #define PMIX_REMOTE        2 // share with procs not on this node
14 #define PMIX_GLOBAL        3 // share with all procs (local + remote)
15 #define PMIX_INTERNAL      4 // store data in the internal tables

```

16 Scope values are defined as:

```

17 PMIX_LOCAL
18     limit access to this data to processes on the same node
19 PMIX_REMOTE
20     limit access to this data to processes on nodes other than this one
21 PMIX_GLOBAL
22     access permitted by all processes (local and remote)
23 PMIX_INTERNAL
24     data is for use solely within this process

```

25 Specific implementations may support different scope values, but all implementations must support  
26 at least **PMIX\_GLOBAL**.

### 27 3.2.3 pmix\_data\_range\_t

28 The **pmix\_data\_range\_t** structure is an integer type for defining a range for “published” data.  
29 The table below defines the possible values for **pmix\_data\_range\_t**.

```

1 //
2 #define PMIX_RANGE_UNDEF 0
3 #define PMIX_RANGE_RM 1 // data is intended for the host resource
4 #define PMIX_RANGE_LOCAL 2 // available on local node only
5 #define PMIX_RANGE_NAMESPACE 3 // data is available to procs in the same namespace
6 #define PMIX_RANGE_SESSION 4 // data available to all procs in session
7 #define PMIX_RANGE_GLOBAL 5 // data available to all procs
8 #define PMIX_RANGE_CUSTOM 6 // range is specified in a pmix_info_t
9 #define PMIX_RANGE_PROC_LOCAL 7 // restrict range to the local proc

```

### 10 3.2.4 pmix\_persistence\_t

11 The `pmix_persistence_t` structure is an integer type for defining the policy for data  
12 published by clients. The table below defines the possible values for `pmix_persistence_t`.

```

13 //
14 #define PMIX_PERSIST_INDEF 0 // retain until specifically deleted
15 #define PMIX_PERSIST_FIRST_READ 1 // delete upon first access
16 #define PMIX_PERSIST_PROC 2 // retain until publishing process terminates
17 #define PMIX_PERSIST_APP 3 // retain until application terminates
18 #define PMIX_PERSIST_SESSION 4 // retain until session/allocation terminates

```

### 19 3.2.5 pmix\_info\_directives\_t

20 The `pmix_info_directives_t` structure is an integer type for defining the behavior of  
21 command directives via `pmix_info_t` arrays. The table below defines the possible values for  
22 `pmix_info_directives_t`.

```

23 //
24 #define PMIX_INFO_REQD 0x0001

```

## 1 3.2.6 pmix\_alloc\_directive\_t

2 The `pmix_alloc_directive_t` structure is an integer type for defining the behavior of  
3 allocation requests. The table below defines the possible values for  
4 `pmix_alloc_directive_t`.

```
▼ C ▼  
5 //  
6 #define PMIX_ALLOC_NEW          1 // new allocation is being requested.  
7                               // disjoint (i.e., not connected in a  
8 #define PMIX_ALLOC_EXTEND      2 // extend the existing allocation, eit  
9 #define PMIX_ALLOC_RELEASE     3 // release part of the existing alloca  
10                               // pmix_info_t array may be used to sp  
11                               // identified resources, or "lending"  
12                               // of time.  
13 #define PMIX_ALLOC_REAQUIRE   4 // reacquire resources that were previ  
14  
15 /* define a value boundary beyond which implementers are free  
16  * to define their own directive values */  
17 #define PMIX_ALLOC_EXTERNAL    128  
▲ C ▲
```

## 18 3.3 Data Packing/Unpacking

19 ...

### 20 3.3.1 Byte Object

21 The `pmix_byte_object_t` structure is ...

```
▼ C ▼  
22 typedef struct pmix_byte_object  
23     char *bytes;  
24     size_t size;  
25     pmix_byte_object_t;  
26 #define PMIX_BYTE_OBJECT_DESTRUCT(m)  
27 #define PMIX_BYTE_OBJECT_FREE(m, n)  
▲ C ▲
```

28 The `pmix_data_buffer_t` structure is ...

```

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

```

## 22 3.4 Data Structures

23 ...

### 24 3.4.1 Process Structure

25 The `pmix_proc_t` structure is ...



```

1     typedef struct pmix_proc
2         char nspace[PMIX_MAX_NSLEN+1];
3         pmix_rank_t rank;
4         pmix_proc_t;
5     #define PMIX_PROC_CREATE(m, n)
6     #define PMIX_PROC_RELEASE(m)
7     #define PMIX_PROC_CONSTRUCT(m)
8     #define PMIX_PROC_DESTRUCT(m)
9     #define PMIX_PROC_FREE(m, n)

```

C

C

## 10 3.4.2 Process Info Structure

11 The `pmix_proc_info_t` structure is ...

```

12     typedef struct pmix_proc_info
13         pmix_proc_t proc;
14         char *hostname;
15         char *executable_name;
16         pid_t pid;
17         int exit_code;
18         pmix_proc_state_t state;
19         pmix_proc_info_t;
20     #define PMIX_PROC_INFO_CREATE(m, n)
21     #define PMIX_PROC_INFO_RELEASE(m)
22     #define PMIX_PROC_INFO_CONSTRUCT(m)
23     #define PMIX_PROC_INFO_DESTRUCT(m)
24     #define PMIX_PROC_INFO_FREE(m, n)

```

C

C

## 25 3.4.3 Data Array Structure

26 The `pmix_data_array` structure is ...

```

27     typedef struct pmix_data_array
28         pmix_data_type_t type;
29         size_t size;
30         void *array;
31         pmix_data_array_t;

```

C

C

## 1 3.4.4 Value Structure

2 The `pmix_value_t` structure is ...

C

```
3 /* NOTE: operations can supply a collection of values under
4  * a single key by passing a pmix_value_t containing an
5  * array of type PMIX_INFO_ARRAY, with each array element
6  * containing its own pmix_info_t object */
```

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

```

1         pmix_alloc_directive_t adir;
2         /**** DEPRECATED *****/
3         pmix_info_array_t *array;
4         /******/
5         data;
6         pmix_value_t;
7         #define PMIX_VALUE_CREATE(m, n)
8         #define PMIX_VALUE_RELEASE(m)
9         #define PMIX_VALUE_CONSTRUCT(m)
10        #define PMIX_VALUE_DESTRUCT(m)
11        #define PMIX_VALUE_FREE(m, n)
12
13        /* expose some functions that are resolved in the
14         * PMIx library, but part of a header that
15         * includes internal functions - we don't
16         * want to expose the entire header here. For
17         * consistency, we provide macro versions as well
18         */
19        void pmix_value_load(pmix_value_t *v, const void *data, pmix_data_type_t
20        #define PMIX_VALUE_LOAD(v, d, t)      pmix_value_load((v), (d), (t))
21
22        pmix_status_t pmix_value_xfer(pmix_value_t *kv, pmix_value_t *src);
23        #define PMIX_VALUE_XFER(r, v, s)
24        pmix_status_t pmix_argv_append_nosize(char ***argv, const char *arg);
25        #define PMIX_ARGV_APPEND(r, a, b)      (r) = pmix_argv_append_nosize(&(a)
26
27        pmix_status_t pmix_setenv(const char *name, const char *value,
28                                bool overwrite, char ***env);
29        #define PMIX_SETENV(r, a, b, c)      (r) = pmix_setenv((a), (b), true, (c)

```

▲ C ▲

### 30 3.4.5 Info and Info Array Structures

31 The `pmix_info_t` structure is ...

32 The `pmix_info_array` structure is ...

```

1 struct pmix_info_t
2     char key[PMIX_MAX_KEYLEN+1]; // ensure room for the NULL terminator
3     pmix_info_directives_t flags; // bit-mask of flags
4     pmix_value_t value;
5 ;
6
7 typedef struct pmix_info_array
8     size_t size;
9     pmix_info_t *array;
10    pmix_info_array_t;
11
12 /* utility macros for working with pmix_info_t structs */
13 #define PMIX_INFO_CREATE(m, n)
14 #define PMIX_INFO_CONSTRUCT(m)
15 #define PMIX_INFO_DESTRUCT(m)
16 #define PMIX_INFO_FREE(m, n)
17 #define PMIX_INFO_LOAD(m, k, v, t)
18 #define PMIX_INFO_XFER(d, s)
19 #define PMIX_INFO_REQUIRED(m)
20 #define PMIX_INFO_OPTIONAL(m)
21 #define PMIX_INFO_UNLOAD(r, v, l)
22 #define PMIX_INFO_TRUE(m)

```

### 23 3.4.6 Lookup Return Structure

24 The `pmix_pdata_t` structure is ...

```

25 typedef struct pmix_pdata
26     pmix_proc_t proc;
27     char key[PMIX_MAX_KEYLEN+1]; // ensure room for the NULL terminator
28     pmix_value_t value;
29     pmix_pdata_t;
30
31 #define PMIX_PDATA_CREATE(m, n)
32 #define PMIX_PDATA_RELEASE(m)
33 #define PMIX_PDATA_CONSTRUCT(m)
34 #define PMIX_PDATA_DESTRUCT(m)
35 #define PMIX_PDATA_FREE(m, n)
36 #define PMIX_PDATA_LOAD(m, p, k, v, t)
37 #define PMIX_PDATA_XFER(d, s)

```

C

### 1 3.4.7 App Structure

2 The `pmix_app_t` structure is ...

C

```

3 typedef struct pmix_app
4     char *cmd;
5     char **argv;
6     char **env;
7     char *cwd;
8     int maxprocs;
9     pmix_info_t *info;
10    size_t ninfo;
11    pmix_app_t;
12
13    /* utility macros for working with pmix_app_t structs */
14    #define PMIX_APP_CREATE(m, n)
15    #define PMIX_APP_RELEASE(m)
16    #define PMIX_APP_CONSTRUCT(m)
17    #define PMIX_APP_DESTRUCT(m)
18    #define PMIX_APP_FREE(m, n)

```

C

### 19 3.4.8 Query Structure

20 The `pmix_query_t` structure is ...

C

```

21 typedef struct pmix_query
22     char **keys;
23     pmix_info_t *qualifiers;
24     size_t nqual;
25     pmix_query_t;
26
27     /* utility macros for working with pmix_query_t structs */
28     #define PMIX_QUERY_CREATE(m, n)
29     #define PMIX_QUERY_RELEASE(m)
30     #define PMIX_QUERY_CONSTRUCT(m)
31     #define PMIX_QUERY_DESTRUCT(m)
32     #define PMIX_QUERY_FREE(m, n)

```

C

## 1 3.4.9 Modex Structure

2 The `pmix_modex_data_t` structure is ...

```
3 typedef struct pmix_modex_data
4     char nspace[PMIX_MAX_NSLEN+1];
5     int rank;
6     uint8_t *blob;
7     size_t size;
8     pmix_modex_data_t;
9
10 /* utility macros for working with pmix_modex_t structs */
11 #define PMIX_MODEX_CREATE(m, n)
12 #define PMIX_MODEX_RELEASE(m)
13 #define PMIX_MODEX_CONSTRUCT(m)
14 #define PMIX_MODEX_DESTRUCT(m)
15 #define PMIX_MODEX_FREE(m, n)
```

## 16 3.5 Callback Functions

17 ...

### 18 3.5.1 Release Callback Function

19 The `pmix_release_cbfunc_t` ...

```
20 typedef void (*pmix_release_cbfunc_t)(void *cbdata)
```

## 1 3.5.2 Modex Callback Function

2 The `pmix_modex_cbfunc_t` ...

```
3 /* define a callback function that is solely used by servers, and
4  * not clients, to return modex data in response to "fence" and "get"
5  * operations. The returned blob contains the data collected from each
6  * server participating in the operation.
7  *
8  * As the data is "owned" by the host server, provide a secondary
9  * callback function to notify the host server that we are done
10 * with the data so it can be released */
11 typedef void (*pmix_modex_cbfunc_t) (pmix_status_t status,
12                                     const char *data, size_t ndata,
13                                     void *cbdata,
14                                     pmix_release_cbfunc_t release_fn,
15                                     void *release_cbdata)
```

### 17 Description

18 ...

## 19 3.5.3 Spawn Callback Function

20 The `pmix_spawn_cbfunc_t` ...

```
21 /* define a callback function for calls to PMIx_Spawn_nb - the function
22  * will be called upon completion of the spawn command. The status
23  * will indicate whether or not the spawn succeeded. The nspace
24  * of the spawned processes will be returned, along with any provided
25  * callback data. Note that the returned nspace value will be
26  * released by the library upon return from the callback function, so
27  * the receiver must copy it if it needs to be retained */
28 typedef void (*pmix_spawn_cbfunc_t) (pmix_status_t status,
29                                     char nspace[], void *cbdata);
```

## Description

The callback will be executed upon launch of the specified applications, or upon failure to launch any of them.

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

## 3.5.4 Op Callback Function

The `pmix_op_cbfunc_t` ...

```
/* define a callback for common operations that simply return
 * a status. Examples include the non-blocking versions of
 * Fence, Connect, and Disconnect */
typedef void (*pmix_op_cbfunc_t) (pmix_status_t status, void *cbdata);
```

## Description

...

## 3.5.5 Lookup Callback Function

The `pmix_lookup_cbfunc_t` ...

```
/* define 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 of failure of the request. Any
 * retrieved data will be returned in an array of pmix_pdata_t structs.
 * The nspace/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 */
```

```
typedef void (*pmix_lookup_cbfunc_t) (pmix_status_t status,
                                     pmix_pdata_t data[], size_t ndata,
                                     void *cbdata);
```



1           **Description**

2           ...

### 3 3.5.6 Value Callback Function

4           The `pmix_value_cbfunc_t` ...

C

```
5 /* define a callback function for calls to PMIx_Get_nb. The status
6  * indicates if the requested data was found or not - a pointer to the
7  * pmix_value_t structure containing the found data is returned. The
8  * pointer will be NULL if the requested data was not found. */
9 typedef void (*pmix_value_cbfunc_t) (pmix_status_t status,
10                                      pmix_value_t *kv, void *cbdata);
```

C

11           **Description**

12           ...

### 13 3.5.7 Info Function

14           The `pmix_info_cbfunc_t` ...

C

```
15 /* define a callback function for calls to PMIx_Query. The status
16  * indicates if requested data was found or not - an array of
17  * pmix_info_t will contain the key/value pairs. */
18 typedef void (*pmix_info_cbfunc_t) (pmix_status_t status,
19                                      pmix_info_t *info, size_t ninfo,
20                                      void *cbdata,
21                                      pmix_release_cbfunc_t release_fn,
22                                      void *release_cbdata);
```

C

23           **Description**

24           ...

## 1 3.6 Other Support Functions

2 ...

### 3 3.6.1 Unsorted Function

```
4      /* Provide a string representation for several types of value. Note
5      * that the provided string is statically defined and must NOT be
6      * free'd. Supported value types:
7      *
8      * - pmix_status_t (PMIX_STATUS)
9      * - pmix_scope_t (PMIX_SCOPE)
10     * - pmix_persistence_t (PMIX_PERSIST)
11     * - pmix_data_range_t (PMIX_DATA_RANGE)
12     * - pmix_info_directives_t (PMIX_INFO_DIRECTIVES)
13     * - pmix_data_type_t (PMIX_DATA_TYPE)
14     * - pmix_alloc_directive_t (PMIX_ALLOC_DIRECTIVE)
15     */
16     PMIX_EXPORT const char* PMIx_Error_string(pmix_status_t status);
17     PMIX_EXPORT const char* PMIx_Proc_state_string(pmix_proc_state_t state);
18     PMIX_EXPORT const char* PMIx_Scope_string(pmix_scope_t scope);
19     PMIX_EXPORT const char* PMIx_Persistence_string(pmix_persistence_t persist);
20     PMIX_EXPORT const char* PMIx_Data_range_string(pmix_data_range_t range);
21     PMIX_EXPORT const char* PMIx_Info_directives_string(pmix_info_directives_t d
22     PMIX_EXPORT const char* PMIx_Data_type_string(pmix_data_type_t type);
23     PMIX_EXPORT const char* PMIx_Alloc_directive_string(pmix_alloc_directive_t d
```

C

### 24 3.6.2 Key/Value Pair Management

C

```
1  /* Key-Value pair management macros */
2  // TODO: add all possible types/fields here.
3
4  #define PMIX_VAL_FIELD_int(x)          ((x)->data.integer)
5  #define PMIX_VAL_FIELD_uint32_t(x)    ((x)->data.uint32)
6  #define PMIX_VAL_FIELD_uint16_t(x)    ((x)->data.uint16)
7  #define PMIX_VAL_FIELD_string(x)      ((x)->data.string)
8  #define PMIX_VAL_FIELD_float(x)       ((x)->data.fval)
9  #define PMIX_VAL_FIELD_byte(x)        ((x)->data.byte)
10 #define PMIX_VAL_FIELD_flag(x)        ((x)->data.flag)
11
12 #define PMIX_VAL_TYPE_int              PMIX_INT
13 #define PMIX_VAL_TYPE_uint32_t        PMIX_UINT32
14 #define PMIX_VAL_TYPE_uint16_t        PMIX_UINT16
15 #define PMIX_VAL_TYPE_string          PMIX_STRING
16 #define PMIX_VAL_TYPE_float           PMIX_FLOAT
17 #define PMIX_VAL_TYPE_byte            PMIX_BYTE
18 #define PMIX_VAL_TYPE_flag            PMIX_BOOL
```

C

C

```
19 #define PMIX_VAL_set_assign(_v, _field, _val )
20 #define PMIX_VAL_set_strdup(_v, _field, _val )
21
22 #define PMIX_VAL_SET_int              PMIX_VAL_set_assign
23 #define PMIX_VAL_SET_uint32_t        PMIX_VAL_set_assign
24 #define PMIX_VAL_SET_uint16_t        PMIX_VAL_set_assign
25 #define PMIX_VAL_SET_string          PMIX_VAL_set_strdup
26 #define PMIX_VAL_SET_float           PMIX_VAL_set_assign
27 #define PMIX_VAL_SET_byte            PMIX_VAL_set_assign
28 #define PMIX_VAL_SET_flag            PMIX_VAL_set_assign
29
30 #define PMIX_VAL_SET(_v, _field, _val )
```

C

```

1      #define PMIX_VAL_cmp_val(_val1, _val2)      ((_val1) != (_val2))
2      #define PMIX_VAL_cmp_float(_val1, _val2)    (((_val1)>(_val2))?(((_val1)-(_v
3      #define PMIX_VAL_cmp_ptr(_val1, _val2)      strcmp(_val1, _val2, strlen(_va
4
5      #define PMIX_VAL_CMP_int                     PMIX_VAL_cmp_val
6      #define PMIX_VAL_CMP_uint32_t               PMIX_VAL_cmp_val
7      #define PMIX_VAL_CMP_uint16_t               PMIX_VAL_cmp_val
8      #define PMIX_VAL_CMP_float                  PMIX_VAL_cmp_float
9      #define PMIX_VAL_CMP_string                  PMIX_VAL_cmp_ptr
10     #define PMIX_VAL_CMP_byte                     PMIX_VAL_cmp_val
11     #define PMIX_VAL_CMP_flag                     PMIX_VAL_cmp_val
12     #define PMIX_VAL_ASSIGN(_v, _field, _val)    #define PMIX_VAL_CMP(_field, _val

```

## CHAPTER 4

# Initialization and Finalization

---

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

4 There are three sets of initialization and finalization functions depending upon the role of the  
5 process in the PMIx universe. Each of these functional sets are described in this chapter.

## 6 4.1 Query





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

### 9 4.1.1 PMIx\_Initialized





#### 10 Summary

11 Determines if the PMIx library has been initialized.

#### 12 Format

13  C   
`int PMIx_Initialized(void)`  
14  C 

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

15  **Rationale**   
16  The return value is an integer for historical reasons as that was the signature of prior PMI libraries.  
17 

#### 16 Description

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

## 1 4.1.2 PMIx\_Get\_version

### 2 Summary

3 Get the PMIx version information.

### 4 Format

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

### 6 Description

7 Get the PMIx version string. Note that the provided string is statically defined and must NOT be  
8 free'd.

## 9 4.2 Client

10 Initialization and finalization routines for PMIx clients.

### 11 4.2.1 PMIx\_Init

#### 12 Summary

13 Initialize the PMIx client.

#### 14 Format

```
15 pmix_status_t  
16 PMIx_Init(pmix_proc_t *proc,  
17           pmix_info_t info[], size_t ninfo)
```

#### 18 INOUT `proc`

19 `proc` structure (handle)

#### 20 IN `info`

21 Array of info structures (array of handles)

#### 22 IN `ninfo`

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

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

## Description

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

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

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

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

Each call to `PMIx_Init` may contain an array of `pmix_info_t` structures passing directives to the PMIx client library. This might include information about the location of temporary directories set up for the application, or constraints on communication protocols for connecting to the local PMIx server. Multiple calls to `PMIx_Init` shall not include conflicting directives (e.g., a directive indicating that one particular communication method be used to connect to the server, followed by a subsequent call that includes a directive that a different method be used). The `PMIx_Init` function will return an error when directives that conflict with prior directives are encountered.

## 4.2.2 `PMIx_Finalize`

### Summary

Finalize the PMIx client library.

### Format

```

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

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

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

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

1           **Description**

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

5           By default, **PMIx\_Finalize** will not include an internal barrier operation. Users desiring a  
6           barrier as part of the finalize operation can request it by including the **PMIX\_EMBED\_BARRIER**  
7           attribute in the provided **pmix\_info\_t** array.

8           **4.3 Tool**

9           Initialization and finalization routines for PMIx tools.

10          **4.3.1 PMIx\_tool\_init**

11           **Summary**

12           Initialize the PMIx library for a tool connection.

13           **Format**

```

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

```

17           **INOUT proc**

18           **pmix\_proc\_t** structure (handle)

19           **IN info**

20           Array of info structures (array of handles)

21           **IN ninfo**

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

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



1           **Description**

2           Initialize the PMIx tool, returning the process identifier assigned to this tool in the provided

3           **pmix\_proc\_t** struct.

4           When called the PMIx tool library will check for the required connection information of the local

5           PMIx server and will establish the connection. If the information is not found, or the server

6           connection fails, then an appropriate error constant will be returned.

7           If successful, the function will return **PMIX\_SUCCESS** and will fill the provided structure with the

8           server-assigned namespace and rank of the tool.

9           Note that the PMIx tool library is referenced counted, and so multiple calls to **PMIx\_tool\_init**

10           are allowed. Thus, one way to obtain the namespace and rank of the process is to simply call

11           **PMIx\_tool\_init** with a non-NULL parameter.

12           The *info* array is used to pass user requests pertaining to the init and subsequent operations. Passing

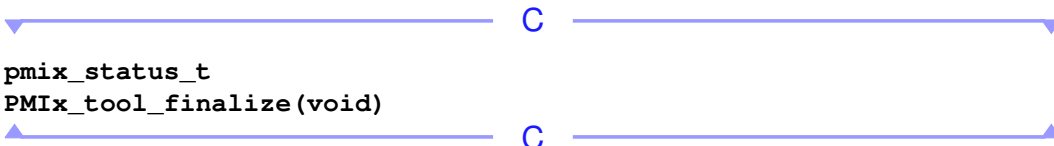
13           a **NULL** value for the array pointer is supported if no directives are desired.

## 14 4.3.2 **PMIx\_tool\_finalize**

15           **Summary**

16           Finalize the PMIx library for a tool connection.

### 17 **Format**

18           The diagram shows the return type `pmix_status_t` and the function signature `PMIx_tool_finalize(void)`. A blue line with a downward-pointing triangle on the left and an upward-pointing triangle on the right spans the width of the text. A blue letter 'C' is centered above the line, and another blue letter 'C' is centered below the line.

19           **pmix\_status\_t**  
20           **PMIx\_tool\_finalize(void)**

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

### 21 **Description**

22           Finalize the PMIx tool library, closing the connection to the local server. An error code will be

23           returned if, for some reason, the connection cannot be closed.

## 24 4.4 **Server**

25           Initialization and finalization routines for PMIx servers.

## 1 4.4.1 PMIx\_server\_init

### 2 Summary

3 Initialize the PMIx server.

### 4 Format

```
5 pmix_status_t  
6 PMIx_server_init(pmix_server_module_t *module,  
7                   pmix_info_t info[], size_t ninfo)
```

### 8 INOUT module

9 `pmix_server_module_t` structure (handle)

10 **IN** `info`

11 Array of info structures (array of handles)

12 **IN** `ninfo`

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

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

### 15 Description

16 Initialize the server support library, and provide a pointer to a `pmix_server_module_t`  
17 structure containing the caller's callback functions. The array of `pmix_info_t` structs is used to  
18 pass additional info that may be required by the server when initializing. For example, a user/group  
19 ID to set on the rendezvous file for the Unix Domain Socket. It also may include the  
20 `PMIX_SERVER_TOOL_SUPPORT` key, thereby indicating that the daemon is willing to accept  
21 connection requests from tools.

## 22 4.4.2 PMIx\_server\_finalize

### 23 Summary

24 Finalize the PMIx server library.

### 25 Format

```
26 pmix_status_t  
27 PMIx_server_finalize(void)
```

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

1           **Description**

2           Finalize the server support library. If internal communication mechanism is in-use, the server will  
3           shut it down at this time. All memory usage is released.

## CHAPTER 5

# Key/Value Management

---

1 ...

## 2 5.1 Setting and Accessing Key/Value Pairs

3 ...

### 4 5.1.1 PMIx\_Put

#### 5 Summary

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

#### 7 Format

```
8 /* Push a value into the client's namespace. The client library will cache
9  * the information locally until _PMIx_Commit_ is called. The provided scope
10 * value is passed to the local PMIx server, which will distribute the data
11 * as directed. */
12 pmix_status_t
13 PMIx_Put (pmix_scope_t scope,
14           const char key[], pmix_value_t *val)
```

15 **IN** `scope`  
16     Distribution scope of the provided value (handle)

17 **IN** `key`  
18     key (string)

19 **IN** `value`  
20     Reference to a `pmix_value_t` structure (handle)

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

1           **Description**

2           Push a value into the client’s namespace. The client library will cache the information locally until

3           **PMIx\_Commit** is called.

4           The provided *scope* is passed to the local PMIx server, which will distribute the data to other

5           processes according to the provided scope. The **pmix\_scope\_t** values are defined in

6           Section 3.2.2 on page ???. Specific implementations may support different scope values, but all

7           implementations must support at least **PMIX\_GLOBAL**.

8           The **pmix\_value\_t** structure supports both string and binary values. Implementations will

9           support heterogeneous environments by properly converting binary values between host

10          architectures, and will copy the provided *value* into internal memory.

▼ ————— Advice to implementers ————— ▼

11          The **PMIx\_Data\_pack** / **PMIx\_Data\_unpack** routines are provided to assist in meeting the

12          heterogeneity requirement.

▲ ————— Advice to users ————— ▲

13          The value is copied by the PMIx client library. Thus, the application is free to release and/or

14          modify the value once the call to **PMIx\_Put** has completed.

▲ —————

## 15   **5.1.2 PMIx\_Get**

### 16          **Summary**

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

## Format

C

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

C

**IN** *proc*  
process reference (handle)

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

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

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

**OUT** *val*  
value (handle)

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

## 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 the specified data has been **PMIx\_Put** by the specified rank in the *proc* structure. 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. This can include the **PMIX\_TIMEOUT** attribute.

### 5.1.3 PMIx\_Get\_nb

#### Summary

Nonblocking **PMIx\_Get** operation.

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

19           **Description**

20       The callback function will be executed once the specified data has been **PMIx\_Put** by the  
21       identified process and retrieved by the local server. The *info* array is used as described by the  
22       **PMIx\_Get** routine.

23   **5.1.4 PMIx\_Store\_internal**

24           **Summary**

25       ...

1

## Format

C

2

/\*

3

pmix\_status\_t

4

PMIx\_Store\_internal(const pmix\_proc\_t \*proc,

5

const char \*key, 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.

13

## 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

## 5.2 Exchanging Key/Value Pairs

17

...

18

### 5.2.1 PMIx\_Commit

19

#### Summary

20

Push all previously **PMIx\_Put** values to the local PMIx server.

21

#### Format

C

22

pmix\_status\_t PMIx\_Commit(void)

C

23

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



## Description

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

### Advice to users

The local PMIx server will cache the information locally. Meaning that the committed data will not be circulated during `PMIx_Commit`. Availability of the data upon completion of `PMIx_Commit` is therefore implementation-dependent.

## 5.2.2 PMIx\_Fence

### Summary

Execute a blocking barrier across the processes identified in the specified array.

### Format

C

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

C

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

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

## 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. This can include:

`PMIX_COLLECT_DATA` "pmix.collect" (bool)

Collect data and return it at the end of the operation

`PMIX_COLLECT_DATA`

(string) A boolean indicating whether or not the barrier operation is to return the *put* data from all participating processes. A value of *false* indicates that the callback is just used as a release and no data is to be returned at that time. A value of *true* indicates that all *put* data is to be collected by the barrier. Returned data is cached at the server to reduce memory footprint, and can be retrieved as needed by calls to `PMIx_Get` / `PMIx_Get_nb`.

`PMIX_COLLECTIVE_ALGO`

(string) A comma-delimited string indicating the algorithm to be used for executing the barrier, in priority order.

`PMIX_COLLECTIVE_ALGO_REQD`

(string) Instructs the host RM that it should return an error if none of the specified algorithms are available. Otherwise, the RM is to use one of the algorithms if possible, but is otherwise free to use any of its available methods to execute the operation.

`PMIX_TIMEOUT`

(string) Maximum time for the fence to execute before declaring an error. By default, the RM shall terminate the operation and notify participants if one or more of the indicated *procs* fails during the fence. However, the timeout parameter can help avoid "hangs" due to programming errors that prevent one or more processes from reaching the "fence".

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

## 5.2.3 `PMIx_Fence_nb`

### Summary

Execute a nonblocking `PMIx_Fence` across the processes identified in the specified array of processes.

### Format

C

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

C

5 **IN** **procs**  
6 Array of [pmix\\_proc\\_t](#) structures (array of handles)  
7 **IN** **nprocs**  
8 Number of element 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 **IN** **cbfunc**  
14 Callback function (function reference)  
15 **IN** **cbdata**  
16 Data to be passed to the callback function (memory reference)

17 Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

## 18 **Description**

19 Nonblocking [PMIx\\_Fence](#) routine. Note that the function will return an error if a **NULL** callback  
20 function is given.

## 21 **5.3 Publish and Lookup Data**

22 ...

### 23 **5.3.1 PMIx\_Publish**

#### 24 **Summary**

25 Publish data for later access via [PMIx\\_Lookup](#).

1

## Format

C

2

`pmix_status_t`

3

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

C

4

**IN** `info`

5

Array of info structures (array of handles)

6

**IN** `ninfo`

7

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

8

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

9

## Description

10

Publish the data in the *info* array for lookup. By default, the data will be published into the `PMIX_SESSION` range and with `PMIX_PERSIST_APP` persistence. Changes to those values, and any additional directives, can be included in the `pmix_info_t` array.

13

Note that the keys must be unique within the specified data range or else an error will be returned (first published wins). Attempts to access the data by processes outside of the provided data range will be rejected.

16

The persistence parameter instructs the server as to how long the data is to be retained.

17

The blocking form will block until the server confirms that the data has been posted and is available. The non-blocking form will return immediately, executing the callback when the server confirms availability of the data.

## 20 5.3.2 PMIx\_Publish\_nb

21

### Summary

22

Nonblocking `PMIx_Publish` routine.

23

### Format

C

24

`pmix_status_t`

25

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

26

```

1      IN   info
2           Array of info structures (array of handles)
3      IN   ninfo
4           Number of element in the info array (integer)
5      IN   cbfunc
6           Callback function pmix_op_cbfunc_t (function reference)
7      IN   cbdata
8           Data to be passed to the callback function (memory reference)
9
9      Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

## 10 Description

```

11      Nonblocking PMIx_Publish routine. Note that the function will return an error if a NULL
12      callback function is given.

```

## 13 5.3.3 PMIx\_Lookup

### 14 Summary

```

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

```

### 17 Format

```

18      pmix_status_t
19      PMIx_Lookup(pmix_pdata_t data[], size_t ndata,
20                  const pmix_info_t info[], size_t ninfo)

```

```

21      IN   data
22           Array of publishable data structures (array of handles)
23      IN   ndata
24           Number of elements in the data array (integer)
25      IN   info
26           Array of info structures (array of handles)
27      IN   ninfo
28           Number of elements in the info array (integer)
29
29      Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

## Description

Lookup information published by this or another process. By default, the search will be conducted across the `PMIX_SESSION` range. Changes to the range, and any additional directives, can be provided in the `pmix_info_t` array.

Note that the search is also constrained to only data published by the current user (i.e., the search will not return data published by an application being executed by another user). 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 `info` 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, or for retrying until the requested data is found.

Optionally, the `info` array can be used to modify this behavior by including:

## 5.3.4 `PMIx_Lookup_nb`

### Summary

Nonblocking version of `PMIx_Lookup`.

## Format

C

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

C

### IN keys

Array to be provided to the callback (array of strings)

### IN info

Array of info structures (array of handles)

### IN ninfo

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

### IN cbfunc

Callback function (handle)

### IN cbdata

Callback data to be provided to the callback function (pointer)

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

## 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* keys can be used to modify the behavior as previously described by **PMIx\_Lookup**.

## 5.3.5 PMIx\_Unpublish

### Summary

Unpublish data posted by this process using the given keys.

### Format

C

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

C

1 **IN** **info**  
2     Array of info structures (array of handles)  
3 **IN** **ninfo**  
4     Number of element in the *info* array (integer)

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

## 6 **Description**

7 Unpublish data posted by this process using the given *keys*. The function will block until the data  
8 has been removed by the server. A value of **NULL** for the *keys* parameter instructs the server to  
9 remove *all* data published by this process.

10 By default, the range is assumed to be **PMIX\_SESSION**. Changes to the range, and any additional  
11 directives, can be provided in the *info* array.

## 12 **5.3.6 PMIx\_Unpublish\_nb**

### 13 **Summary**

14 Nonblocking version of **PMIx\_Unpublish**.

### 15 **Format**

C

```
16 pmix_status_t  
17 PMIx_Unpublish_nb(char **keys,  
18                   const pmix_info_t info[], size_t ninfo,  
19                   pmix_op_cbfunc_t cbfunc, void *cbdata)
```

C

20 **IN** **keys**  
21     (array of strings)  
22 **IN** **info**  
23     Array of info structures (array of handles)  
24 **IN** **ninfo**  
25     Number of element in the *info* array (integer)  
26 **IN** **cbfunc**  
27     Callback function **pmix\_op\_cbfunc\_t** (function reference)  
28 **IN** **cbdata**  
29     Data to be passed to the callback function (memory reference)

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



1        **Description**

2        Non-blocking form of the `PMIx_Unpublish` function. The callback function will be executed  
3        once the server confirms removal of the specified data.

## CHAPTER 6

# Process Management

---

1 ...

## 2 6.1 Abort

3 ...

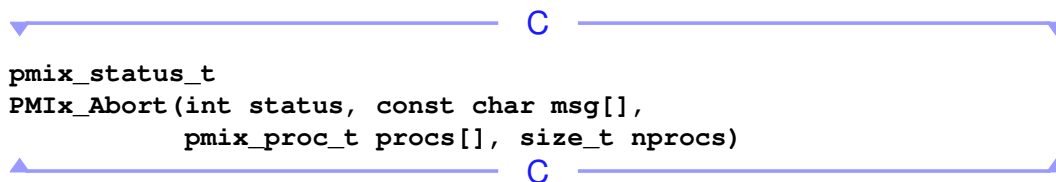
### 4 6.1.1 PMI<sub>x</sub>\_Abort

#### 5 Summary

6 Abort the specified process.

#### 7 Format

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



- 11 **IN** `status`  
12     Error code to return to invoking environment (integer)
- 13 **IN** `msg`  
14     String message to be returned to user (string)
- 15 **IN** `procs`  
16     Array of `pmix_proc_t` structures (array of handles)
- 17 **IN** `nprocs`  
18     Number of elements in the `procs` array (integer)

19 Returns `PMIX_SUCCESS` or a negative value corresponding to a PMI<sub>x</sub> 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

...

### 6.2.1 PMIx\_Spawn

#### Summary

Spawn a new job.

## Format

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

**IN** **job\_info**  
Array of info structures (array of handles)

**IN** **ninfo**  
Number of elements in the *job\_info* array (integer)

**IN** **apps**  
Array of `pmix_app_t` structures (array of handles)

**IN** **napps**  
Number of elements in the *apps* array (integer)

**OUT** **nspace**  
Namespace of the new job (string)

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

## Description

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

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

Job-level directives can be specified in the *job\_info* array. This can include:

**PMIX\_NON\_PMI**  
(string) Processes in the spawned job will not be calling `PMIx_Init`.

**PMIX\_TIMEOUT**  
(string) Declare the spawn as having failed if the launched processes do not call `PMIx_Init` within the specified time.

1 **PMIX\_NOTIFY\_COMPLETION**  
2 (string) Notify the parent process when the child job terminates, either normally or with  
3 error.

## 4 **6.2.2 PMIx\_Spawn\_nb**

### 5 **Summary**

6 Nonblocking version of the [PMIx\\_Spawn](#) routine.

### 7 **Format**

C

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

C

12 **IN** *job\_info*  
13 Array of info structures (array of handles)  
14 **IN** *ninfo*  
15 Number of elements in the *job\_info* array (integer)  
16 **IN** *apps*  
17 Array of [pmix\\_app\\_t](#) structures (array of handles)  
18 **IN** *cbfunc*  
19 Callback function [pmix\\_spawn\\_cbfunc\\_t](#) (function reference)  
20 **IN** *cbdata*  
21 Data to be passed to the callback function (memory reference)  
22 Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

### 23 **Description**

24 Nonblocking version of the [PMIx\\_Spawn](#) routine.

## 25 **6.3 Connecting and Disconnecting Processes**

26 ...

## 1 6.3.1 PMIx\_Connect

### 2 Summary

3 Connect namespaces.

### 4 Format

```
5 pmix_status_t  
6 PMIx_Connect(const pmix_proc_t procs[], size_t nprocs,  
7               const pmix_info_t info[], size_t ninfo)
```

8 **IN** **procs**

9 Array of proc structures (array of handles)

10 **IN** **nprocs**

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

12 **IN** **info**

13 Array of info structures (array of handles)

14 **IN** **ninfo**

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

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

### 17 Description

18 Record the specified processes as “connected”. This means that the resource manager should treat  
19 the failure of any process in the specified group as a reportable event, and take appropriate action.  
20 Note that different resource managers may respond to failures in different manners.

21 The callback function is to be called once all participating processes have called connect. The  
22 server is required to return any job-level info for the connecting processes that might not already  
23 have (i.e., if the connect request involves *procs* from different namespaces, then each *proc* shall  
24 receive the job-level info from those namespaces other than their own.

25 A process can only engage in *one* connect operation involving the identical set of processes at a  
26 time. However, a process *can* be simultaneously engaged in multiple connect operations, each  
27 involving a different set of processes.

28 As in the case of the fence operation, the info array can be used to pass user-level directives  
29 regarding the algorithm to be used for the collective operation involved in the “connect”, timeout  
30 constraints, and other options available from the host RM.

## 1 6.3.2 PMIx\_Connect\_nb

### 2 Summary

3 Nonblocking [PMIx\\_Connect\\_nb](#) routine.

### 4 Format

```
5 pmix_status_t  
6 PMIx_Connect_nb(const pmix_proc_t procs[], size_t nprocs,  
7                 const pmix_info_t info[], size_t ninfo,  
8                 pmix_op_cbfunc_t cbfunc, void *cbdata)
```

- 9 **IN procs**  
10     Array of proc structures (array of handles)
  - 11 **IN nprocs**  
12     Number of elements in the *procs* array (integer)
  - 13 **IN info**  
14     Array of info structures (array of handles)
  - 15 **IN ninfo**  
16     Number of element in the *info* array (integer)
  - 17 **IN cbfunc**  
18     Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)
  - 19 **IN cbdata**  
20     Data to be passed to the callback function (memory reference)
- 21 Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

### 22 Description

23 Nonblocking [PMIx\\_Connect\\_nb](#) routine.

## 24 6.3.3 PMIx\_Disconnect

### 25 Summary

26 Disconnect a previously connected set of processes.

## Format

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

**IN** **procs**  
Array of proc structures (array of handles)

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

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

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

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

## Description

Disconnect a previously connected set of processes. An error will be returned if the specified set of *procs* was not previously “connected”. As with **PMIx\_Connect**, a process may be involved in multiple simultaneous disconnect operations. However, a process is not allowed to reconnect to a set of *procs* that has not fully completed disconnect (i.e., you have to fully disconnect before you can reconnect to the *same* group of processes. The *info* array is used as in **PMIx\_Connect**.

## 6.3.4 PMIx\_Disconnect\_nb

### Summary

Nonblocking **PMIx\_Disconnect** routine.

### Format

```
pmix_status_t
PMIx_Disconnect_nb(const pmix_proc_t ranges[], size_t nprocs,
                  const pmix_info_t info[], size_t ninfo,
                  pmix_op_cbfunc_t cbfunc, void *cbdata);
```



1       **IN** **procs**  
 2            Array of proc structures (array of handles)

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

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

7       **IN** **ninfo**  
 8            Number of element in the *info* array (integer)

9       **IN** **cbfunc**  
 10            Callback function `pmix_op_cbfunc_t` (function reference)

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

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

14       **Description**  
 15        Nonblocking **PMIx\_Disconnect** routine.

## 16 6.4 Query

17       ...

### 18 6.4.1 PMIx\_Resolve\_peers

#### 19 Summary

20       Access an array of processes within the specified namespace on a node.

#### 21 Format

22       **pmix\_status\_t**  
 23       **PMIx\_Resolve\_peers**(const char \*nodename, const char \*nspc,  
 24                            pmix\_proc\_t \*\*procs, size\_t \*nprocs)

C

1     **IN**   **nodename**  
2         Name of the node to query (string)  
3     **IN**   **nospace**  
4         namespace (string)  
5     **OUT**  **procs**  
6         Array of process structures (array of handles)  
7     **OUT**  **nprocs**  
8         Number of elements in the *procs* array (integer)

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

## 10    **Description**

11    Given a *nodename*, return an array of processes within the specified *nospace* on that node. If the  
12    *nospace* is **NULL**, then all processes on the node will be returned. If the specified node does not  
13    currently host any processes, then the returned array will be **NULL**, and *nprocs* will be **0**. The caller  
14    is responsible for releasing the *procs* array when done with it. The **PMIX\_PROC\_FREE** macro is  
15    provided for this purpose.

## 16    **6.4.2 PMIx\_Resolve\_nodes**

### 17    **Summary**

18    Return a list of nodes hosting processes.

### 19    **Format**

C

20    **pmix\_status\_t**  
21    **PMIx\_Resolve\_nodes**(const char \*nospace, char \*\*nodelist)

C

22    **IN**   **nospace**  
23         Namespace (string)  
24    **OUT**  **nodelist**  
25         Comma-delimited list of nodenames (string)

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

## Description

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

### 6.4.3 PMIx\_Query\_info\_nb

#### Summary

Query information about the system in general.

#### Format

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

#### IN queries

Array of query structures (array of handles)

#### IN nqueries

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

#### IN cbfunc

Callback function [pmix\\_info\\_cbfunc\\_t](#) (function reference)

#### IN cbdata

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

[PMIX\\_SUCCESS](#) All data has been returned

[PMIX\\_ERR\\_NOT\\_FOUND](#) None of the requested data was available

[PMIX\\_ERR\\_PARTIAL\\_SUCCESS](#) Some of the data has been returned

[PMIX\\_ERR\\_NOT\\_SUPPORTED](#) The host RM does not support this function

1  
2  
3  
4  
5  
6  
7  
8  
9

## Description

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

NOTE: There is no blocking form of this API as the structures passed to query info differ from those for receiving the results.

The *status* argument to the callback function indicates if requested data was found or not. An array of `pmix_info_t` will contain the key/value pairs.

## CHAPTER 7

# Job Allocation Management

---

1        ...

## 2    7.1    Allocation Requests

3        ...

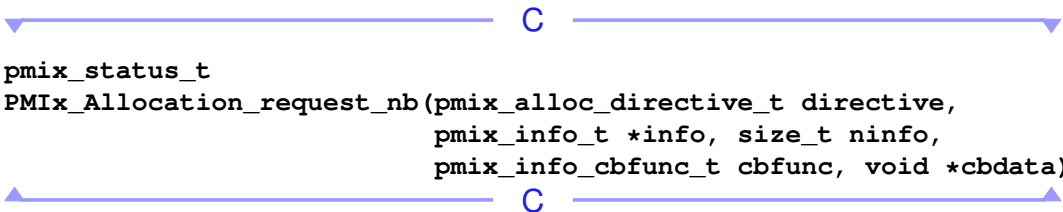
### 4    7.1.1    `PMIx_Allocation_request_nb`

#### 5        **Summary**

6        Request an allocation operation from the host resource manager.

#### 7        **Format**

```
8        pmix_status_t  
9        PMIx_Allocation_request_nb(pmix_alloc_directive_t directive,  
10                                    pmix_info_t *info, size_t ninfo,  
11                                    pmix_info_cbfunc_t cbfunc, void *cbdata);
```



- 12        **IN**    `directive`  
13                Allocation directive (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_info_cbfunc_t` (function reference)
- 20        **IN**    `cbdata`  
21                Data to be passed to the callback function (memory reference)

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

## Description

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

- Request allocation of additional resources, including memory, bandwidth, and compute. This should be accomplished in a non-blocking manner so that the application can continue to progress while waiting for resources to become available. Note that the new allocation will be disjoint from (i.e., not affiliated with) the allocation of the requestor - thus the termination of one allocation will not impact the other.
- Extend the reservation on currently allocated resources, subject to scheduling availability and priorities. This includes extending the time limit on current resources, and/or requesting additional resources be allocated to the requesting job. Any additional allocated resources will be considered as part of the current allocation, and thus will be released at the same time.
- Release currently allocated resources that are no longer required. This is intended to support partial release of resources since all resources are normally released upon termination of the job. The identified use-cases include resource variations across discrete steps of a workflow, as well as applications that spawn sub-jobs and/or dynamically grow/shrink over time.
- “Lend” resources back to the scheduler with an expectation of getting them back at some later time in the job. This can be a proactive operation (e.g., to save on computing costs when resources are temporarily not required), or in response to scheduler requests in lieu of preemption. A corresponding ability to “reacquire” resources previously released is included.

### 7.1.2 PMIx\_Job\_control\_nb

#### Summary

Request a job control action.

#### Format

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

- IN targets**  
Array of proc structures (array of handles)
- IN ntargets**  
Number of element in the *targets* array (integer)
- IN directives**  
Array of info structures (array of handles)

1       **IN**   **ndirs**  
2            Number of element in the *directives* 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 `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

## 8       **Description**

9       Request a job control action. The *targets* array identifies the processes to which the requested job  
10       control action is to be applied. A `NULL` value can be used to indicate all processes in the caller's  
11       namespace. The use of `PMIX_RANK_WILDARD` can also be used to indicate that all processes in  
12       the given namespace are to be included.

13       The directives are provided as `pmix_info_t` structures in the *directives* array. The callback  
14       function provides a *status* to indicate whether or not the request was granted, and to provide some  
15       information as to the reason for any denial in the `pmix_info_cbfunc_t` array of  
16       `pmix_info_t` structures. If non-`NULL`, then the specified *release\_fn* must be called when the  
17       callback function completes - this will be used to release any provided `pmix_info_t` array.

## 18    **7.2 Process and Job Monitoring**



19       ...

### 20    **7.2.1 PMIx\_Process\_monitor\_nb**

#### 21       **Summary**

22       Request that something be monitored.

#### 23       **Format**

24        **C**   
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,`  
          `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 `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

## 14       **Description**

15       Request that something be monitored. For example, that the server monitor this process for periodic  
 16       heartbeats as an indication that the process has not become “wedged”. When a monitor detects the  
 17       specified alarm condition, it will generate an event notification using the provided error code and  
 18       passing along any available relevant information. It is up to the caller to register a corresponding  
 19       event handler.

20       The *monitor* argument is an attribute indicating the type of monitor being requested. For example,  
 21       `PMIX_MONITOR_FILE` to indicate that the requestor is asking that a file be monitored.

22       The *error* argument is the status code to be used when generating an event notification alerting that  
 23       the monitor has been triggered. The range of the notification defaults to  
 24       `PMIX_RANGE_NAMESPACE`. This can be changed by providing a `PMIX_RANGE` directive.

25       The *directives* argument characterizes the monitoring request (e.g., monitor file size) and frequency  
 26       of checking to be done

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

## 30       **7.2.2 PMIx\_Heartbeat**

### 31       **Summary**

32       Send a heartbeat to the RM



1

## Format

▼ \_\_\_\_\_ C \_\_\_\_\_ ▼

2

```
void PMIx_Heartbeat(void)
```

▲ \_\_\_\_\_ C \_\_\_\_\_ ▲

3

## Description

4

A simplified version of [PMIx\\_Process\\_monitor\\_nb](#) that sends a heartbeat to the RM.

## CHAPTER 8

# Event Notification

---

1 ...

## 2 8.1 Logging

3 ...

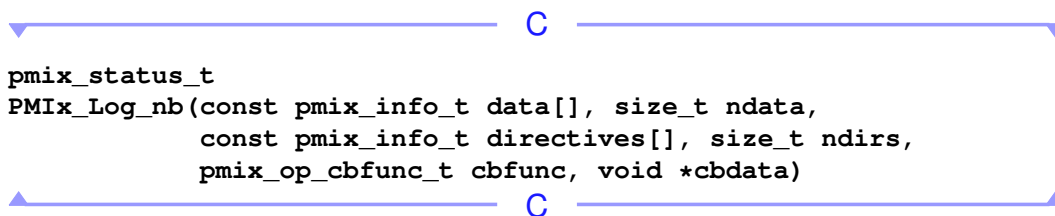
### 4 8.1.1 PMIx\_Log\_nb

#### 5 Summary

6 Log data to a data service.

#### 7 Format

8 `pmix_status_t`  
9 `PMIx_Log_nb(const pmix_info_t data[], size_t ndata,`  
10 `const pmix_info_t directives[], size_t ndirs,`  
11 `pmix_op_cbfunc_t cbfunc, void *cbdata)`



12 **IN data**  
13     Array of info structures (array of handles)

14 **IN ndata**  
15     Number of elements in the *data* array (integer)

16 **IN directives**  
17     Array of info structures (array of handles)

18 **IN ndirs**  
19     Number of elements in the *directives* array (integer)

20 **IN cbfunc**  
21     Callback function `pmix_op_cbfunc_t` (function reference)

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

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

## Description

Log data to a “central” data service/store, subject to the services offered by the host resource manager. The data to be logged is provided in the *data* array. The (optional) *directives* can be used to request specific storage options and direct the choice of storage option.

The callback function will be executed when the log operation has been completed. The *data* array must be maintained until the callback is provided.

## 8.2 Notification and Management

...

### 8.2.1 PMIx\_Register\_event\_handler

C

```
/* Register an event handler to report events. Three types of events
 * can be reported:
 *
 * (a) those that occur within the client library, but are not
 * reportable via the API itself (e.g., loss of connection to
 * the server). These events typically occur during behind-the-scenes
 * non-blocking operations.
 *
 * (b) job-related events such as the failure of another process in
 * the job or in any connected job, impending failure of hardware
 * within the job's usage footprint, etc.
 *
 * (c) system notifications that are made available by the local
 * administrators
 *
 * By default, only events that directly affect the process and/or
 * any process to which it is connected (via the PMIx_Connect call)
 * will be reported. Options to modify that behavior can be provided
 * in the info array
 *
 * Both the client application and the resource manager can register
 * err handlers for specific events. PMIx client/server calls the registe
 * err handler upon receiving event notify notification (via PMIx_Notify_
 * from the other end (Resource Manager/Client application).
 *
```

```

1      * Multiple err handlers can be registered for different events. PMIX return
2      * an integer reference to each register handler in the callback fn. The cal
3      * must retain the reference in order to deregister the evhdlr.
4      * Modification of the notification behavior can be accomplished by
5      * deregistering the current evhdlr, and then registering it
6      * using a new set of info values.
7      *
8      * See pmix_common.h for a description of the notification function */
9      void
10     PMIx_Register_event_handler(pmix_status_t codes[], size_t ncodes,
11                               pmix_info_t info[], size_t ninfo,
12                               pmix_notification_fn_t evhdlr,
13                               pmix_evhdlr_reg_cbfunc_t cbfunc,
14                               void *cbdata);

```

C

## 15 8.2.2 PMIx\_Deregister\_event\_handler

```

16     /* Deregister an event handler
17     * evhdlr_ref is the reference returned by PMIx from the call to
18     * PMIx_Register_event_handler. If non-NULL, the provided cbfunc
19     * will be called to confirm removal of the designated handler */
20     void
21     PMIx_Deregister_event_handler(size_t evhdlr_ref,
22                                   pmix_op_cbfunc_t cbfunc,
23                                   void *cbdata);

```

C

## 24 8.2.3 PMIx\_Notify\_event

```

1  /* Report an event to a process for notification via any
2  * registered evhdlr. The evhdlr registration can be
3  * called by both the server and the client application. On the
4  * server side, the evhdlr is used to report events detected
5  * by PMIx to the host server for handling. On the client side,
6  * the evhdlr is used to notify the process of events
7  * reported by the server - e.g., the failure of another process.
8  *
9  * This function allows the host server to direct the server
10 * convenience library to notify all registered local procs of
11 * an event. The event can be local, or anywhere in the cluster.
12 * The status indicates the event being reported.
13 *
14 * The client application can also call this function to notify the
15 * resource manager of an event it encountered. It can request the host
16 * server to notify the indicated processes about the event.
17 *
18 * The array of procs identifies the processes that will be impacted
19 * by the event. This could consist of a single process, or a number
20 * of processes.
21 *
22 * The info array contains any further info the RM can and/or chooses
23 * to provide.
24 *
25 * The callback function will be called upon completion of the
26 * notify_event function's actions. Note that any messages will
27 * have been queued, but may not have been transmitted by this
28 * time. Note that the caller is required to maintain the input
29 * data until the callback function has been executed!
30 */
31 pmix_status_t
32 PMIx_Notify_event(pmix_status_t status,
33                 const pmix_proc_t *source,
34                 pmix_data_range_t range,
35                 pmix_info_t info[], size_t ninfo,
36                 pmix_op_cbfunc_t cbfunc, void *cbdata);

```

## 37 8.2.4 Event Notification Callback Function

38 The [pmix\\_event\\_notification\\_cbfunc\\_fn\\_t](#) ...

C

```
1 /* define a callback by which an event handler can notify the PMIx library
2  * that it has completed its response to the notification. The handler
3  * is _required_ to execute this callback so the library can determine
4  * if additional handlers need to be called. The handler shall return
5  * PMIX_SUCCESS if no further action is required. The return status
6  * of each event handler and any returned pmix_info_t structures
7  * will be added to the array of pmix_info_t passed to any subsequent
8  * event handlers to help guide their operation.
9  *
10 * If non-NULL, the provided callback function will be called to allow
11 * the event handler to release the provided info array.
12 */
13 typedef void (*pmix_event_notification_cbfunc_fn_t) (pmix_status_t status,
14                                                     pmix_info_t *results, si
15                                                     pmix_op_cbfunc_t cbfunc,
16                                                     void *notification_cbdat
```

C

## Description

...

## 8.2.5 Notification Callback Function

The [pmix\\_notification\\_fn\\_t](#) ...

C

```
1 /* define a callback function for the event handler. Upon receipt of an
2  * event notification, PMIx will execute the specified notification
3  * callback function, providing:
4  *
5  * evhdlr_registration_id - the returned registration number of
6  *                           the event handler being called
7  * status - the event that occurred
8  * source - the nspace and rank of the process that generated
9  *           the event. If the source is the resource manager,
10 *           then the nspace will be empty and the rank will
11 *           be PMIX_RANK_UNDEF
12 * info - any additional info provided regarding the event.
13 * ninfo - the number of info objects in the info array
14 * results - any provided results from event handlers called
```

```

1      *           prior to this one.
2      * nresults - number of info objects in the results array
3      * cbfunc - the function to be called upon completion of the handler
4      * cbdata - pointer to be returned in the completion cbfunc
5      *
6      * Note that different resource managers may provide differing levels
7      * of support for event notification to application processes. Thus, the
8      * info array may be NULL or may contain detailed information of the event.
9      * It is the responsibility of the application to parse any provided info
10     * for defined key-values if it so desires.
11     *
12     * Possible uses of the pmix_info_t object include:
13     *
14     * - for the RM to alert the process as to planned actions, such as
15     *   to abort the session, in response to the reported event
16     *
17     * - provide a timeout for alternative action to occur, such as for
18     *   the application to request an alternate response to the event
19     *
20     * For example, the RM might alert the application to the failure of
21     * a node that resulted in termination of several processes, and indicate
22     * that the overall session will be aborted unless the application
23     * requests an alternative behavior in the next 5 seconds. The application
24     * then has time to respond with a checkpoint request, or a request to
25     * recover from the failure by obtaining replacement nodes and restarting
26     * from some earlier checkpoint.
27     *
28     * Support for these options is left to the discretion of the host RM. In
29     * keys are included in the common definitions above, but also may be augmented
30     * on a per-RM basis.
31     *
32     * On the server side, the notification function is used to inform the host
33     * server of a detected event in the PMIx subsystem and/or client
34     */
35     typedef void (*pmix_notification_fn_t)(size_t evhdlr_registration_id,
36                                           pmix_status_t status,
37                                           const pmix_proc_t *source,
38                                           pmix_info_t info[], size_t ninfo,
39                                           pmix_info_t *results, size_t nresults,
40                                           pmix_event_notification_cbfunc_fn_t cbfunc,
41                                           void *cbdata);

```

1       **Description**

2       ...

### 3   **8.2.6 Event Handler Registration Function**

4       The `pmix_evhdlr_reg_cbfunc_t` ...



```
5       /* define a callback function for calls to PMIx_Register_evhdlr. The  
6       * status indicates if the request was successful or not, evhdlr_ref is  
7       * an integer reference assigned to the event handler by PMIx, this referenc  
8       * must be used to deregister the err handler. A ptr to the original  
9       * cbdata is returned. */
```

```
10       typedef void (*pmix_evhdlr_reg_cbfunc_t) (pmix_status_t status,  
11                                                 size_t evhdlr_ref,  
12                                                 void *cbdata)
```



13       **Description**

14       ...



## CHAPTER 9

# Data Packing and Unpacking

---

1 ...

## 2 9.1 General Routines

3 ...

### 4 9.1.1 PMIx\_Data\_pack

C

```
5  /**
6   * Top-level interface function to pack one or more values into a
7   * buffer.
8   *
9   * The pack function packs one or more values of a specified type into
10  * the specified buffer. The buffer must have already been
11  * initialized via the PMIX_DATA_BUFFER_CREATE or PMIX_DATA_BUFFER_CONSTR
12  * call - otherwise, the pack_value function will return an error.
13  * Providing an unsupported type flag will likewise be reported as an error.
14  *
15  * Note that any data to be packed that is not hard type cast (i.e.,
16  * not type cast to a specific size) may lose precision when unpacked
17  * by a non-homogeneous recipient. The PACK function will do its best to
18  * with heterogeneity issues between the packer and unpacker in such
19  * cases. Sending a number larger than can be handled by the recipient
20  * will return an error code (generated upon unpacking) -
21  * the error cannot be detected during packing.
22  *
23  * @param *buffer A pointer to the buffer into which the value is to
24  * be packed.
25  *
26  * @param *src A void* pointer to the data that is to be packed. Note
27  * that strings are to be passed as (char **) - i.e., the caller must
28  * pass the address of the pointer to the string as the void*. This
```

```

1      * allows PMIx to use a single pack function, but still allow
2      * the caller to pass multiple strings in a single call.
3      *
4      * @param num_values An int32_t indicating the number of values that are
5      * to be packed, beginning at the location pointed to by src. A string
6      * value is counted as a single value regardless of length. The values
7      * must be contiguous in memory. Arrays of pointers (e.g., string
8      * arrays) should be contiguous, although (obviously) the data pointed
9      * to need not be contiguous across array entries.
10     *
11     * @param type The type of the data to be packed - must be one of the
12     * PMIX defined data types.
13     *
14     * @retval PMIX_SUCCESS The data was packed as requested.
15     *
16     * @retval PMIX_ERROR(s) An appropriate PMIX error code indicating the
17     * problem encountered. This error code should be handled
18     * appropriately.
19     *
20     * @code
21     * pmix_data_buffer_t *buffer;
22     * int32_t src;
23     *
24     * PMIX_DATA_BUFFER_CREATE(buffer);
25     * status_code = PMIx_Data_pack(buffer, &src, 1, PMIX_INT32);
26     * @endcode
27     */
28     pmix_status_t
29     PMIx_Data_pack(pmix_data_buffer_t *buffer,
30                   void *src, int32_t num_vals,
31                   pmix_data_type_t type);

```

C

## 32 9.1.2 PMIx\_Data\_unpack

```
1  /**
2  * Unpack values from a buffer.
3  *
4  * The unpack function unpacks the next value (or values) of a
5  * specified type from the specified buffer.
6  *
7  * The buffer must have already been initialized via an PMIX_DATA_BUFFER_
8  * PMIX_DATA_BUFFER_CONSTRUCT call (and assumedly filled with some data)
9  * otherwise, the unpack_value function will return an
10 * error. Providing an unsupported type flag will likewise be reported
11 * as an error, as will specifying a data type that DOES NOT match the
12 * type of the next item in the buffer. An attempt to read beyond the
13 * end of the stored data held in the buffer will also return an
14 * error.
15 *
16 * NOTE: it is possible for the buffer to be corrupted and that
17 * PMIx will *think* there is a proper variable type at the
18 * beginning of an unpack region - but that the value is bogus (e.g., jus
19 * a byte field in a string array that so happens to have a value that
20 * matches the specified data type flag). Therefore, the data type error
21 * is NOT completely safe. This is true for ALL unpack functions.
22 *
23 *
24 * Unpacking values is a "nondestructive" process - i.e., the values are
25 * not removed from the buffer. It is therefore possible for the caller
26 * to re-unpack a value from the same buffer by resetting the unpack_ptr.
27 *
28 * Warning: The caller is responsible for providing adequate memory
29 * storage for the requested data. As noted below, the user
30 * must provide a parameter indicating the maximum number of values that
31 * can be unpacked into the allocated memory. If more values exist in the
32 * buffer than can fit into the memory storage, then the function will un
33 * what it can fit into that location and return an error code indicating
34 * that the buffer was only partially unpacked.
35 *
36 * Note that any data that was not hard type cast (i.e., not type cast
37 * to a specific size) when packed may lose precision when unpacked by
38 * a non-homogeneous recipient. PMIx will do its best to deal with
39 * heterogeneity issues between the packer and unpacker in such
40 * cases. Sending a number larger than can be handled by the recipient
41 * will return an error code generated upon unpacking - these errors
42 * cannot be detected during packing.
```

```

1      *
2      * @param *buffer A pointer to the buffer from which the value will be
3      * extracted.
4      *
5      * @param *dest A void* pointer to the memory location into which the
6      * data is to be stored. Note that these values will be stored
7      * contiguously in memory. For strings, this pointer must be to (char
8      * **) to provide a means of supporting multiple string
9      * operations. The unpack function will allocate memory for each
10     * string in the array - the caller must only provide adequate memory
11     * for the array of pointers.
12     *
13     * @param type The type of the data to be unpacked - must be one of
14     * the BFRDP defined data types.
15     *
16     * @retval *max_num_values The number of values actually unpacked. In
17     * most cases, this should match the maximum number provided in the
18     * parameters - but in no case will it exceed the value of this
19     * parameter. Note that if you unpack fewer values than are actually
20     * available, the buffer will be in an unpackable state - the function will
21     * return an error code to warn of this condition.
22     *
23     * @note The unpack function will return the actual number of values
24     * unpacked in this location.
25     *
26     * @retval PMIX_SUCCESS The next item in the buffer was successfully
27     * unpacked.
28     *
29     * @retval PMIX_ERROR(s) The unpack function returns an error code
30     * under one of several conditions: (a) the number of values in the
31     * item exceeds the max num provided by the caller; (b) the type of
32     * the next item in the buffer does not match the type specified by
33     * the caller; or (c) the unpack failed due to either an error in the
34     * buffer or an attempt to read past the end of the buffer.
35     *
36     * @code
37     * pmix_data_buffer_t *buffer;
38     * int32_t dest;
39     * char **string_array;
40     * int32_t num_values;
41     *
42     * num_values = 1;
43     * status_code = PMIx_Data_unpack(buffer, (void*)&dest, &num_values, PMIX_IN

```

```

1      *
2      * num_values = 5;
3      * string_array = malloc(num_values*sizeof(char *));
4      * status_code = PMIx_Data_unpack(buffer, (void*)(string_array), &num_val
5      *
6      * @endcode
7      */
8      pmix_status_t
9      PMIx_Data_unpack(pmix_data_buffer_t *buffer, void *dest,
10                      int32_t *max_num_values,
11                      pmix_data_type_t type);

```

C

### 12 9.1.3 PMIx\_Data\_copy

C

```

13      /**
14      * Copy a data value from one location to another.
15      *
16      * Since registered data types can be complex structures, the system
17      * needs some way to know how to copy the data from one location to
18      * another (e.g., for storage in the registry). This function, which
19      * can call other copy functions to build up complex data types, defines
20      * the method for making a copy of the specified data type.
21      *
22      * @param **dest The address of a pointer into which the
23      * address of the resulting data is to be stored.
24      *
25      * @param *src A pointer to the memory location from which the
26      * data is to be copied.
27      *
28      * @param type The type of the data to be copied - must be one of
29      * the PMIx defined data types.
30      *
31      * @retval PMIX_SUCCESS The value was successfully copied.
32      *
33      * @retval PMIX_ERROR(s) An appropriate error code.
34      *
35      */
36      pmix_status_t
37      PMIx_Data_copy(void **dest, void *src,
38                    pmix_data_type_t type);

```

C

## 1 9.1.4 PMIx\_Data\_print

```
2      /**  
3      * Print a data value.  
4      *  
5      * Since registered data types can be complex structures, the system  
6      * needs some way to know how to print them (i.e., convert them to a string  
7      * representation). Provided for debug purposes.  
8      *  
9      * @retval PMIX_SUCCESS The value was successfully printed.  
10     *  
11     * @retval PMIX_ERROR(s) An appropriate error code.  
12     */  
13     pmix_status_t  
14     PMIx_Data_print(char **output, char *prefix,  
15                    void *src, pmix_data_type_t type);
```

## 16 9.1.5 PMIx\_Data\_copy\_payload

```
17     /**  
18     * Copy a payload from one buffer to another  
19     *  
20     * This function will append a copy of the payload in one buffer into  
21     * another buffer.  
22     * NOTE: This is NOT a destructive procedure - the  
23     * source buffer's payload will remain intact, as will any pre-existing  
24     * payload in the destination's buffer.  
25     */  
26     pmix_status_t  
27     PMIx_Data_copy_payload(pmix_data_buffer_t *dest,  
28                          pmix_data_buffer_t *src);
```

## CHAPTER 10

# Server Specific Interfaces

---

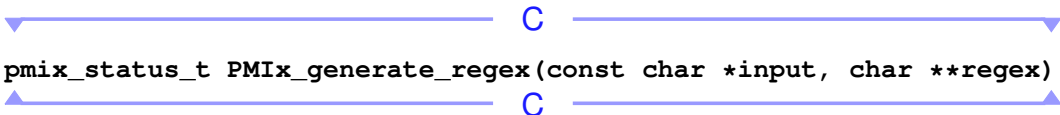
1 ...

## 2 10.0.1 `PMIx_generate_regex`

### 3 Summary

4 Generate a regular expression representation of the input string.

### 5 Format

6   
`pmix_status_t PMIx_generate_regex(const char *input, char **regex)`

7 **IN** `input`  
8 String to process (string)

9 **OUT** `regex`  
10 Regular expression representation of *input* (string)

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

### 12 Description

13 Given a semicolon-separated list of *input* values, generate a regular expression that can be passed  
14 down to the PMIx client for parsing. The caller is responsible for free'ing the resulting string.

15 If values have leading zero's, then that is preserved. You have to add back any prefix/suffix for node  
16 names.

## 17 10.0.2 `PMIx_generate_ppn`

### 18 Summary

19 Generate a regular expression representation of the input string.





## Description

The PMIx connection procedure provides an opportunity for the host PMIx server to pass job-related info down to a child process. This might include the number of processes in the job, relative local ranks of the processes within the job, and other information of use to the process. The server is free to determine which, if any, of the supported elements it will provide (See [Data Structures and Types](#) for values).

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

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

## 10.0.4 PMIx\_server\_deregister\_namespace

### Summary

Deregister a namespace.

### Format

```
void PMIx_server_deregister_namespace(const char namespace[],  
                                     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

#### IN namespace

Namespace (string)

#### IN cbfunc

Callback function [pmix\\_op\\_cbfunc\\_t](#) (function reference)

#### IN cbdata

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

### Description

Deregister the specified *namespace* 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.

## 1 10.0.5 PMIx\_server\_register\_client

### 2 Summary

3 Register a client process with the PMIx server library.

### 4 Format

```
5 pmix_status_t PMIx_server_register_client(const pmix_proc_t *proc,  
6                                           uid_t uid, gid_t gid,  
7                                           void *server_object,  
8                                           pmix_op_cbfunc_t cbfunc, void *cbdata);
```

9 **IN** **proc**  
10 `pmix_proc_t` structure (handle)  
11 **IN** **uid**  
12 user id (integer)  
13 **IN** **gid**  
14 group id (integer)  
15 **IN** **server\_object**  
16 (memory reference)  
17 **IN** **cbfunc**  
18 Callback function `pmix_op_cbfunc_t` (function reference)  
19 **IN** **cbdata**  
20 Data to be passed to the callback function (memory reference)

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

### 22 Description

23 Register a client process with the PMIx server library. The expected user ID and group ID of the  
24 child process helps the server library to properly authenticate clients as they connect by requiring  
25 the two values to match.

26 The host server can also, if it desires, provide an object it wishes to be returned when a server  
27 function is called that relates to a specific process. For example, the host server may have an object  
28 that tracks the specific client. Passing the object to the library allows the library to return that object  
29 when the client calls “finalize”, thus allowing the host server to access the object without  
30 performing a lookup.

## 1 10.0.6 PMIx\_server\_deregister\_client

### 2 Summary

3 Deregister a client and purge all data relating to it.

### 4 Format

```
5 void PMIx_server_deregister_client(const pmix_proc_t *proc,  
6 pmix_op_cbfunc_t cbfunc, void *cbdata)
```

7 **IN** `proc`  
8 `pmix_proc_t` structure (handle)  
9 **IN** `cbfunc`  
10 Callback function `pmix_op_cbfunc_t` (function reference)  
11 **IN** `cbdata`  
12 Data to be passed to the callback function (memory reference)

### 13 Description

14 The `PMIx_server_deregister_namespace` API will automatically delete all client  
15 information for that namespace. This API is therefore intended solely for use in exception cases.

## 16 10.0.7 PMIx\_server\_setup\_fork

### 17 Summary

18 Setup the environment of a child process to be forked by the host.

### 19 Format

```
20 pmix_status_t PMIx_server_setup_fork(const pmix_proc_t *proc, char ***env)
```

21 **IN** `proc`  
22 `pmix_proc_t` structure (handle)  
23 **IN** `env`  
24 Environment array (array of strings)

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

## Description

Setup the environment of a child process to be forked by the host so it can correctly interact with the PMIx server. The PMIx client needs some setup information so it can properly connect back to the server. This function will set appropriate environmental variables for this purpose.

## 10.0.8 PMIx\_server\_dmodex\_request

### Summary

Define a function by which the host server can request modex data from the local PMIx server.

### Format

```
typedef void (*pmix_dmodex_response_fn_t) (pmix_status_t status,  
                                           char *data, size_t sz,  
                                           void *cbdata);  
  
pmix_status_t PMIx_server_dmodex_request (const pmix_proc_t *proc,  
                                           pmix_dmodex_response_fn_t cbfunc,  
                                           void *cbdata)
```

**IN proc**  
    **pmix\_proc\_t** structure (handle)

**IN cbfunc**  
    Callback function **pmix\_dmodex\_response\_fn\_t** (function reference)

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

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

### Description

Define a function by which the host server can request modex data from the local PMIx server. This is used to support the direct modex operation (i.e., where data is cached locally on each PMIx server for its own local clients, and is obtained on-demand for remote requests. Upon receiving a request from a remote server, the host server will call this function to pass the request into the PMIx server. The PMIx server will return a blob (once it becomes available) via the *cbfunc* - the host server shall send the blob back to the original requestor.

The callback function used by the PMIx server to return direct modex requests to the host server. The PMIx server will free the data blob upon return from the response function.

## 1 10.0.9 PMIx\_server\_setup\_application

### 2 Summary

3 Provide a function by which the resource manager can request any application-specific  
4 environmental variables prior to launch of an application.

### 5 Format

```
6 typedef void (*pmix_setup_application_cbfunc_t) (pmix_status_t status,  
7                                               pmix_info_t info[], size_t ninfo,  
8                                               void *provided_cbdata,  
9                                               pmix_op_cbfunc_t cbfunc,  
10  
11 pmix_status_t PMIx_server_setup_application(const char nspace[],  
12                                           pmix_info_t info[], size_t ninfo,  
13                                           pmix_setup_application_cbfunc_t cbfunc,  
14                                           void *cbdata)
```

15 **IN nspace**  
16 namespace (string)

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

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

21 **IN cbfunc**  
22 Callback function `pmix_setup_application_cbfunc_t` (function reference)

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

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

### 26 Description

27 Provide a function by which the resource manager can request any application-specific  
28 environmental variables prior to launch of an application. For example, network libraries may opt  
29 to provide security credentials for the application. This is defined as a non-blocking operation in  
30 case network libraries need to perform some action before responding. The returned env will be  
31 distributed along with the application

32 In the callback function, the returned *info* array is owned by the PMIx server library and will be  
33 free'd when the provided *cbfunc* is called.

## 1 10.0.10 PMIx\_server\_setup\_local\_support

### 2 Summary

3 Provide a function by which the local PMIx server can perform any application-specific operations  
4 prior to spawning local clients of a given application.

### 5 Format

```
6 pmix_status_t PMIx_server_setup_local_support(const char nspace[],  
7 pmix_info_t info[], size_t nin  
8 pmix_op_cbfunc_t cbfunc, void
```

9 **IN nspace**

Namespace (string)

11 **IN info**

Array of info structures (array of handles)

13 **IN ninfo**

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

15 **IN cbfunc**

Callback function **pmix\_op\_cbfunc\_t** (function reference)

17 **IN cbdata**

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

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

### 20 Description

21 Provide a function by which the local PMIx server can perform any application-specific operations  
22 prior to spawning local clients of a given application. For example, a network library might need to  
23 setup the local driver for “instant on” addressing.

## 24 10.1 Server Function Pointers

25 The PMIx Server will set the function pointers in the **pmix\_server\_module\_t** structure that  
26 they then pass to **PMIx\_server\_init**. That module structure and associated function  
27 references is defined in this section.

## 1 10.1.1 pmix\_server\_module\_t Module

### 2 Summary

3 List of function pointers that a PMIx server passes to `PMIx_server_init` during startup.

### 4 Format

C

```
5 typedef struct pmix_server_module_2_0_0_t
6     /* v1x interfaces */
7     pmix_server_client_connected_fn_t    client_connected;
8     pmix_server_client_finalized_fn_t    client_finalized;
9     pmix_server_abort_fn_t               abort;
10    pmix_server_fence_nb_fn_t             fence_nb;
11    pmix_server_dmodex_req_fn_t           direct_modex;
12    pmix_server_publish_fn_t              publish;
13    pmix_server_lookup_fn_t               lookup;
14    pmix_server_unpublish_fn_t            unpublish;
15    pmix_server_spawn_fn_t                spawn;
16    pmix_server_connect_fn_t              connect;
17    pmix_server_disconnect_fn_t           disconnect;
18    pmix_server_register_events_fn_t      register_events;
19    pmix_server_deregister_events_fn_t    deregister_events;
20    pmix_server_listener_fn_t             listener;
21    /* v2x interfaces */
22    pmix_server_notify_event_fn_t         notify_event;
23    pmix_server_query_fn_t                query;
24    pmix_server_tool_connection_fn_t      tool_connected;
25    pmix_server_log_fn_t                   log;
26    pmix_server_alloc_fn_t                 allocate;
27    pmix_server_job_control_fn_t          job_control;
28    pmix_server_monitor_fn_t              monitor;
29    pmix_server_module_t;
```

C

### 30 Description

31 NOTE: for performance purposes, the host server is required to return as quickly as possible from  
32 all functions. Execution of the function is thus to be done asynchronously so as to allow the PMIx  
33 server support library to handle multiple client requests as quickly and scalably as possible.

34 All data passed to the host server functions is “owned” by the PMIX server support library and  
35 MUST NOT be free’d. Data returned by the host server via callback function is owned by the host  
36 server, which is free to release it upon return from the callback.

## 1 10.1.2 pmix\_server\_client\_connected\_fn\_t

### 2 Summary

3 Notify the host server that a client connected to this server.

### 4 Format

```
5 typedef pmix_status_t (*pmix_server_client_connected_fn_t) (  
6     const pmix_proc_t *proc, void* server_object,  
7     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

8 **IN** `proc`

9 `pmix_proc_t` structure (handle)

10 **IN** `server_object`

11 object reference (memory reference)

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

### 17 Description

18 Notify the host server that a client connected to us. Note that the client will be in a blocked state  
19 until the host server executes the callback function, thus allowing the PMIx server support library to  
20 release the client.

## 21 10.1.3 pmix\_server\_client\_finalized\_fn\_t

### 22 Summary

23 Notify the host server that a client called `PMIx_Finalize`.



## Format

C

```
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)
```

C

**IN** `proc`

`pmix_proc_t` structure (handle)

**IN** `server_object`

object reference (memory reference)

**IN** `cbfunc`

Callback function `pmix_op_cbfunc_t` (function reference)

**IN** `cbdata`

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

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

## Description

Notify the host server 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.

## 10.1.4 pmix\_server\_abort\_fn\_t

### Summary

Notify PMIx Server that a local client called `PMIx_Abort`.

### Format

C

```
typedef pmix_status_t (*pmix_server_abort_fn_t) (  
    const pmix_proc_t *proc, void *server_object  
    int status, const char msg[],  
    pmix_proc_t procs[], size_t nprocs,  
    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

```

1  IN  proc
2      pmix_proc_t structure (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 (array of handles)
11  IN  nprocs
12     Number of elements in the procs array (integer)
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 **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

## 18 Description

19 A local client called **PMIx\_Abort**. Note that the client will be in a blocked state until the host  
20 server executes the callback function, thus allowing the PMIx server support library to release the  
21 client. The array of *procs* indicates which processes are to be terminated. A **NULL** indicates that all  
22 processes in the client's namespace are to be terminated.

## 23 10.1.5 pmix\_server\_fenceb\_fn\_t

### 24 Summary

25 At least one client called either **PMIx\_Fence** or **PMIx\_Fence\_nb**.

### 26 Format

```

27 typedef pmix_status_t (*pmix_server_fenceb_fn_t) (
28     const pmix_proc_t procs[], size_t nprocs,
29     const pmix_info_t info[], size_t ninfo,
30     char *data, size_t ndata,
31     pmix_modex_cbfunc_t cbfunc, void *cbdata)

```

1       **IN**   **procs**  
 2            Array of **pmix\_proc\_t** structures (array of handles)  
 3       **IN**   **nprocs**  
 4            Number of elements in the *procs* array (integer)  
 5       **IN**   **info**  
 6            Array of info structures (array of handles)  
 7       **IN**   **ninfo**  
 8            Number of elements in the *info* array (integer)  
 9       **IN**   **data**  
 10            (string)  
 11       **IN**   **ndata**  
 12            (integer)  
 13       **IN**   **cbfunc**  
 14            Callback function **pmix\_modex\_cbfunc\_t** (function reference)  
 15       **IN**   **cbdata**  
 16            Data to be passed to the callback function (memory reference)  
 17       Returns **PMIX\_SUCCESS** or a negative value corresponding to a PMIx error constant.

## 18       **Description**

19       At least one client called either **PMIx\_Fence** or **PMIx\_Fence\_nb** . In either case, the host  
 20       server will be called via a non-blocking function to execute the specified operation once all  
 21       participating local processes have contributed. All processes in the specified *procs* array are  
 22       required to participate in the **PMIx\_Fence** / **PMIx\_Fence\_nb** operation. The callback is to be  
 23       executed once each daemon hosting at least one participant has called the host server's  
 24       **pmix\_server\_fence\_nb\_fn\_t** function.

25       The provided data is to be collectively shared with all PMIx servers involved in the fence operation,  
 26       and returned in the modex *cbfunc*. A **NULL** data value indicates that the local processes had no data  
 27       to contribute.

28       The array of *info* structs is used to pass user-requested options to the server. This can include  
 29       directives as to the algorithm to be used to execute the fence operation. The directives are optional  
 30       *unless* the *mandatory* flag has been set - in such cases, the host RM is required to return an error if  
 31       the directive cannot be met.

## 32   **10.1.6 pmix\_server\_dmodex\_req\_fn\_t**

### 33       **Summary**

34       Used by the PMIx server to request its local host contact the PMIx server on the remote node that  
 35       hosts the specified proc to obtain and return a direct modex blob for that proc.

## Format

```
typedef pmix_status_t (*pmix_server_dmodex_req_fn_t) (  
    const pmix_proc_t *proc,  
    const pmix_info_t info[], size_t ninfo,  
    pmix_modex_cbfunc_t cbfunc, void *cbdata)
```

**IN** **proc**  
    **pmix\_proc\_t** structure (handle)

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

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

**IN** **cbfunc**  
    Callback function **pmix\_modex\_cbfunc\_t** (function reference)

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

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

## 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 a direct modex blob for that proc.

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.

## 10.1.7 pmix\_server\_publish\_fn\_t

### Summary

Publish data per the PMIx API specification.

## Format

```
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 (handle)

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

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

**IN** **cbfunc**  
    Callback function **pmix\_op\_cbfunc\_t** (function reference)

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

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

## Description

Publish data per the PMIx API specification. The callback is to be executed upon completion of the operation. The default data range is expected to be **PMIX\_SESSION**, and the default persistence **PMIX\_PERSIST\_SESSION**. These values can be modified by including the respective **pmix\_info\_t** struct in the *info* array.

Note that the host server is not required to guarantee support for any specific range - i.e., the server does not need to return an error if the data store doesn't support range-based isolation. 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 server to allow info-key-based flags to modify this behavior.

The persistence indicates how long the server should retain the data.

The identifier of the publishing process is also provided and is expected to be returned on any subsequent lookup request.

## 10.1.8 pmix\_server\_lookup\_fn\_t

### Summary

Lookup published data.

## Format

```
typedef pmix_status_t (*pmix_server_lookup_fn_t) (  
    const pmix_proc_t *proc, char **keys,  
    const pmix_info_t info[], size_t ninfo,  
    pmix_lookup_cbfunc_t cbfunc, void *cbdata)
```

**IN** **proc**  
    **pmix\_proc\_t** structure (handle)

**IN** **keys**  
    (array of strings)

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

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

**IN** **cbfunc**  
    Callback function **pmix\_lookup\_cbfunc\_t** (function reference)

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

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

## Description

Lookup published data. The host server will be passed a NULL-terminated array of string keys.

The array of *info* structs is used to pass user-requested options to the server. This can include a wait flag to indicate that the server should wait for all data to become available before executing the callback function, or should immediately callback with whatever data is available. In addition, a timeout can be specified on the wait to preclude an indefinite wait for data that may never be published.

### 10.1.9 pmix\_server\_unpublish\_fn\_t

## Summary

Delete data from the data store.

## Format

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

## Description

Delete data from the data store. The host server will be passed a NULL-terminated array of string keys, plus potential directives such as the data range within which the keys should be deleted. The callback is to be executed upon completion of the delete procedure.

### 10.1.10 pmix\_server\_spawn\_fn\_t

#### Summary

Spawn a set of applications/processes as per the PMIx API.

## Format

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

**IN** `proc`  
    **pmix\_proc\_t** structure (handle)

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

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

**IN** `apps`  
    Array of **pmix\_app\_t** structures (array of handles)

**IN** `napps`  
    Number of elements in the *apps* array (integer)

**IN** `cbfunc`  
    Callback function **pmix\_spawn\_cbfunc\_t** (function reference)

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

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

## Description

Spawn a set of applications/processes as per the PMIx 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.

### 10.1.11 pmix\_server\_connect\_fn\_t

#### Summary

Record the specified processes as “connected”.



## 1 Format

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

### 6 IN **procs**

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

### 8 IN **nprocs**

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

### 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 `pmix_op_cbfunc_t` (function reference)

### 16 IN **cbdata**

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

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

## 19 Description

20 Record the specified processes as “connected”. This means that the resource manager should treat  
21 the failure of any process in the specified group as a reportable event, and take appropriate action.  
22 The callback function is to be called once all participating processes have called connect. Note that  
23 a process can only engage in **one** connect operation involving the identical set of processes at a  
24 time. However, a process *can* be simultaneously engaged in multiple connect operations, each  
25 involving a different set of processes.

26 Note also that this is a collective operation within the client library, and thus the client will be  
27 blocked until all processes participate. Thus, the *info* array can be used to pass user directives,  
28 including a timeout. The directives are optional *unless* the *mandatory* flag has been set - in such  
29 cases, the host RM is required to return an error if the directive cannot be met.

## 30 10.1.12 pmix\_server\_disconnect\_fn\_t

### 31 Summary

32 Disconnect a previously connected set of processes.

## Format

```
1                                     C
2 typedef pmix_status_t (*pmix_server_disconnect_fn_t) (
3                                     const pmix_proc_t procs[], size_t nprocs,
4                                     const pmix_info_t info[], size_t ninfo,
5                                     pmix_op_cbfunc_t cbfunc, void *cbdata)
6                                     C
```

6 **IN** **procs**  
7     Array of `pmix_proc_t` 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 elements 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 `PMIX_SUCCESS` or a negative value corresponding to a PMIx error constant.

## Description

20 Disconnect a previously connected set of processes. An error should be returned if the specified set  
21 of processes was not previously “connected”. As above, a process may be involved in multiple  
22 simultaneous disconnect operations. However, a process is not allowed to reconnect to a set of  
23 ranges that has not fully completed disconnect (i.e., you have to fully disconnect before you can  
24 reconnect to the same group of processes).

25 Note also that this is a collective operation within the client library, and thus the client will be  
26 blocked until all processes participate. Thus, the *info* array can be used to pass user directives,  
27 including a timeout. The directives are optional *unless* the *mandatory* flag has been set - in such  
28 cases, the host RM is required to return an error if the directive cannot be met.

## 10.1.13 pmix\_server\_register\_events\_fn\_t

### Summary

31 Register to receive notifications for the specified events.

## 1 Format

```
2 typedef pmix_status_t (*pmix_server_register_events_fn_t) (  
3     pmix_status_t *codes, size_t ncodes,  
4     const pmix_info_t info[], size_t ninfo,  
5     pmix_op_cbfunc_t cbfunc, void *cbdata)
```

### 6 IN codes

7 Array of [pmix\\_status\\_t](#) structures (array of handles)

### 8 IN ncodes

9 Number of elements in the *codes* array (integer)

### 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 [pmix\\_op\\_cbfunc\\_t](#) (function reference)

### 16 IN cbdata

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

18 Returns [PMIX\\_SUCCESS](#) or a negative value corresponding to a PMIx error constant.

## 19 Description

20 Register to receive notifications for the specified events. The resource manager is *required* to pass  
21 along to the local PMIx server all events that directly relate to a registered namespace. However, the  
22 RM may have access to events beyond those (e.g., environmental events). The PMIx server will  
23 register to receive environmental events that match specific PMIx event codes. If the host RM  
24 supports such notifications, it will need to translate its own internal event codes to fit into a  
25 corresponding PMIx event code - any specific info beyond that can be passed in via the  
26 [pmix\\_info\\_t](#) upon notification.

27 The *info* array included in this API is reserved for possible future directives to further steer  
28 notification.

## 29 10.1.14 pmix\_server\_deregister\_events\_fn\_t

### 30 Summary

31 Deregister to receive notifications for the specified events.

1

## Format

C

2

```
typedef pmix_status_t (*pmix_server_deregister_events_fn_t) (
    pmix_status_t *codes, size_t ncodes,
    pmix_op_cbfunc_t cbfunc, void *cbdata)
```

3

4

C

5

### IN codes

6

Array of `pmix_status_t` structures (array of handles)

7

### IN ncodes

8

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

9

### IN cbfunc

10

Callback function `pmix_op_cbfunc_t` (function reference)

11

### IN cbdata

12

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

13

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

14

## Description

15

Deregister to receive notifications for the specified environmental events for which the PMIx server has previously registered. The host RM remains required to notify of any job-related events.

16

## 17 10.1.15 pmix\_server\_notify\_event\_fn\_t

18

## Summary

19

Notify the specified processes of an event.

20

## Format

C

21

```
typedef pmix_status_t (*pmix_server_notify_event_fn_t) (pmix_status_t code,
    const pmix_proc_t *so,
    pmix_data_range_t ran,
    pmix_info_t info[], s
    pmix_op_cbfunc_t cbfu
```

22

23

24

25

```

1      IN  code
2          pmix_status_t structure (handle)
3      IN  source
4          pmix_proc_t (handle)
5      IN  range
6          pmix_data_range_t (handle)
7      IN  info
8          Array of info structures (array of handles)
9      IN  ninfo
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
16     Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

## Description

```

17     Notify the specified processes of an event generated either by the PMIx server itself, or by one of its
18     local clients. The process generating the event is provided in the source parameter.

```

## 19 10.1.16 pmix\_connection\_cbfunc\_t

### 20 Summary

```

21     Callback function for incoming connection requests from local clients.

```

### 22 Format

```

23     typedef void (*pmix_connection_cbfunc_t) (
24         int incoming_sd, void *cbdata)

```

```

25     IN  incoming_sd
26         (integer)
27     IN  cbdata
28         (memory reference)
29
30     Returns PMIX_SUCCESS or a negative value corresponding to a PMIx error constant.

```

1 **Description**

2 Callback function for incoming connection requests from local clients.

3 **10.1.17 pmix\_server\_listener\_fn\_t**

4 **Summary**

5 Register a socket the host server can monitor for connection requests.

6 **Format**

```
7 typedef pmix_status_t (*pmix_server_listener_fn_t) (  
8                       int listening_sd,  
9                       pmix_connection_cbfunc_t cbfunc,  
10                       void *cbdata)  
11                                               C
```

11 **IN incoming\_sd**  
12 (integer)  
13 **IN cbfunc**  
14 Callback function [pmix\\_connection\\_cbfunc\\_t](#) (function reference)  
15 **IN cbdata**  
16 (memory reference)

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

18 **Description**

19 Register a socket the host server can monitor for connection requests, harvest them, and then call  
20 our internal callback function for further processing. A listener thread is essential to efficiently  
21 harvesting connection requests from large numbers of local clients such as occur when running on  
22 large SMPs. The host server listener is required to call `accept` on the incoming connection request,  
23 and then passing the resulting socket to the provided `cbfunc`. A NULL for this function will cause the  
24 internal PMIx server to spawn its own listener thread.

25 **10.1.18 pmix\_server\_query\_fn\_t**

26 **Summary**

27 Query information from the resource manager.

## Format

C

```
typedef pmix_status_t (*pmix_server_query_fn_t) (  
    pmix_proc_t *proct,  
    pmix_query_t *queries, size_t nqueries,  
    pmix_info_cbfunc_t cbfunc,  
    void *cbdata)
```

C

**IN** `proct`

`pmix_proc_t` structure (handle)

**IN** `queries`

Array of `pmix_query_t` structures (array of handles)

**IN** `nqueries`

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

**IN** `cbfunc`

Callback function `pmix_info_cbfunc_t` (function reference)

**IN** `cbdata`

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

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

## Description

Query information from the resource manager. The query will include the nspace/rank of the process that is requesting the info, an array of `pmix_query_t` describing the request, and a callback function/data for the return.

## 10.1.19 pmix\_tool\_connection\_cbfunc\_t

### Summary

Callback function for incoming tool connections.

### Format

C

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

C

1 **IN** **status**  
2 **pmix\_status\_t** structure (handle)  
3 **IN** **proc**  
4 **pmix\_proc\_t** structure (handle)  
5 **IN** **cbdata**  
6 Data to be passed (memory reference)

## 7 **Description**

8 Callback function for incoming tool connections. The host RM shall provide an nspace/rank for the  
9 connecting tool. We assume that a **rank=0** will be the normal assignment, but allow for the future  
10 possibility of a parallel set of tools connecting, and thus each proc requiring a rank.

## 11 **10.1.20 pmix\_server\_tool\_connection\_fn\_t**

### 12 **Summary**

13 Register that a tool has connected to the server.

### 14 **Format**

C

```
15 typedef void (*pmix_server_tool_connection_fn_t) (  
16             pmix_info_t *info, size_t ninfo,  
17             pmix_tool_connection_cbfunc_t cbfunc,  
18             void *cbdata)
```

C

19 **IN** **info**  
20 Array of info structures (array of handles)  
21 **IN** **ninfo**  
22 Number of elements in the *info* array (integer)  
23 **IN** **cbfunc**  
24 Callback function **pmix\_tool\_connection\_cbfunc\_t** (function reference)  
25 **IN** **cbdata**  
26 Data to be passed to the callback function (memory reference)



## Description

Register that a tool has connected to the server, and request that the tool be assigned an nspace/rank for further interactions. The optional `pmix_info_t` array can be used to pass qualifiers for the connection request:

### **PMIX\_USERID**

effective userid of the tool

### **PMIX\_GRPID**

effective groupid of the tool

### **PMIX\_FWD\_STDOUT**

forward any stdout to this tool

### **PMIX\_FWD\_STDERR**

forward any stderr to this tool

### **PMIX\_FWD\_STDIN**

forward stdin from this tool to any processes spawned on its behalf

## 10.1.21 pmix\_server\_log\_fn\_t

### Summary

Log data on behalf of a client.

### Format

```
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)

- 1 **IN cbfunc**
- 2     Callback function `pmix_op_cbfunc_t` (function reference)
- 3 **IN cbdata**
- 4     Data to be passed to the callback function (memory reference)

5 **Description**

6 Log data on behalf of a client.

7 **10.1.22 pmix\_server\_alloc\_fn\_t**

8 **Summary**

9 Request allocation modifications on behalf of a client.

10 **Format**

```

11 typedef pmix_status_t (*pmix_server_alloc_fn_t) (
12     const pmix_proc_t *client,
13     pmix_alloc_directive_t directive,
14     const pmix_info_t data[], size_t ndata,
15     pmix_info_cbfunc_t cbfunc, void *cbdata)

```

- 16 **IN client**
- 17     `pmix_proc_t` structure (handle)
- 18 **IN directive**
- 19     (handle)
- 20 **IN data**
- 21     Array of info structures (array of handles)
- 22 **IN ndata**
- 23     Number of elements in the *data* array (integer)
- 24 **IN cbfunc**
- 25     Callback function `pmix_info_cbfunc_t` (function reference)
- 26 **IN cbdata**
- 27     Data to be passed to the callback function (memory reference)

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

29 **Description**

30 Request allocation modifications on behalf of a client.

## 1 10.1.23 pmix\_server\_job\_control\_fn\_t

### 2 Summary

3 Execute a job control action on behalf of a client.

### 4 Format

```
5 typedef pmix_status_t (*pmix_server_job_control_fn_t) (  
6     const pmix_proc_t *requestor,  
7     const pmix_proc_t targets[], size_t ntargets  
8     const pmix_info_t directives[], size_t ndirs  
9     pmix_info_cbfunc_t cbfunc, void *cbdata)
```

10 **IN requestor**

11 `pmix_proc_t` structure (handle)

12 **IN targets**

13 Array of proc structures (array of handles)

14 **IN ntargets**

15 Number of elements in the *targets* array (integer)

16 **IN directives**

17 Array of info structures (array of handles)

18 **IN ndirs**

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

20 **IN cbfunc**

21 Callback function `pmix_op_cbfunc_t` (function reference)

22 **IN cbdata**

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

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

### 25 Description

26 Execute a job control action on behalf of a client.

## 27 10.1.24 pmix\_server\_monitor\_fn\_t

### 28 Summary

29 Request that a client be monitored for activity.

1

## Format

C

2

```
/* Request that a client be monitored for activity */
```

3

```
typedef pmix_status_t (*pmix_server_monitor_fn_t) (
```

4

```
    const pmix_proc_t *requestor,
```

5

```
    const pmix_info_t *monitor, pmix_status_t error
```

6

```
    const pmix_info_t directives[], size_t ndirs,
```

7

```
    pmix_info_cbfunc_t cbfunc, void *cbdata);
```

C

8

**IN requestor**

9

`pmix_proc_t` structure (handle)

10

**IN monitor**

11

`pmix_proc_t` structure (handle)

12

**IN error**

13

(integer)

14

**IN directives**

15

Array of info structures (array of handles)

16

**IN ndirs**

17

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

18

**IN cbfunc**

19

Callback function `pmix_op_cbfunc_t` (function reference)

20

**IN cbdata**

21

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

22

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

23

## Description

24

Request that a client be monitored for activity.

## APPENDIX A

# Document Revision History

---

### 1 **A.1 Version 2.0: Date TBD**

- 2       • ...

### 3 **A.2 Version 1.0: ad hoc release**

4       An ad hoc standard was defined in the PMIx Reference Implementation header files before the  
5       creation of the formal 2.0 standard. Below are a summary listing of the interfaces defined in the 1.0  
6       headers.

- 7       • ...

## 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 2.0**

6 The following list includes some of the active participants in the PMIx standardization process.

7 • ...

8 The following institutions supported this effort through time and travel support for the people listed  
9 above.

10 • ...

## 11 **B.2 Version 1.0**

12 The following list includes some of the active participants in the PMIx standardization process.

13 • ...

14 The following institutions supported this effort through time and travel support for the people listed  
15 above.

16 • ...

# 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

---

- PMIx\_Abort, [23](#), [67](#), [105](#), [106](#)
  - Defintion, [66](#)
- PMIX\_ADD\_HOST
  - Defintion, [17](#)
- PMIX\_ADD\_HOSTFILE
  - Defintion, [17](#)
- PMIX\_ALLOC\_BANDWIDTH
  - Defintion, [21](#)
- PMIX\_ALLOC\_CPU\_LIST
  - Defintion, [21](#)
- pmix\_alloc\_directive\_t, [31](#)
  - Defintion, [31](#)
- PMIX\_ALLOC\_ID
  - Defintion, [20](#)
- PMIX\_ALLOC\_MEM\_SIZE
  - Defintion, [21](#)
- PMIX\_ALLOC\_NETWORK
  - Defintion, [21](#)
- PMIX\_ALLOC\_NETWORK\_ID
  - Defintion, [21](#)
- PMIX\_ALLOC\_NETWORK\_QOS
  - Defintion, [21](#)
- PMIX\_ALLOC\_NODE\_LIST
  - Defintion, [20](#)
- PMIX\_ALLOC\_NUM\_CPU\_LIST
  - Defintion, [21](#)
- PMIX\_ALLOC\_NUM\_CPUS
  - Defintion, [20](#)
- PMIX\_ALLOC\_NUM\_NODES
  - Defintion, [20](#)
- PMIX\_ALLOC\_TIME
  - Defintion, [21](#)
- PMIX\_ALLOCATED\_NODELIST
  - Defintion, [12](#)
- PMIx\_Allocation\_request\_nb
  - Defintion, [77](#)
- PMIX\_ANL\_MAP
  - Defintion, [15](#)
- PMIX\_APP\_MAP\_REGEX
  - Defintion, [15](#)
- PMIX\_APP\_MAP\_TYPE
  - Defintion, [15](#)
- PMIX\_APP\_RANK
  - Defintion, [11](#)
- PMIX\_APP\_SIZE
  - Defintion, [12](#)
- pmix\_app\_t, [37](#), [68](#), [69](#), [112](#)
  - Defintion, [37](#)
- PMIX\_APPLDR
  - Defintion, [11](#)
- PMIX\_APPNUM
  - Defintion, [11](#)
- PMIX\_ARCH
  - Defintion, [10](#)
- PMIX\_ATTR\_UNDEF
  - Defintion, [7](#)
- PMIX\_AVAIL\_PHYS\_MEMORY
  - Defintion, [13](#)
- PMIX\_BINDTO
  - Defintion, [17](#)
- pmix\_byte\_object\_t, [31](#)
  - Defintion, [31](#)
- PMIX\_CLIENT\_AVG\_MEMORY
  - Defintion, [13](#)
- PMIX\_COLLECT\_DATA, [58](#)
  - Defintion, [14](#)
- PMIX\_COLLECTIVE\_ALGO, [58](#)
  - Defintion, [14](#)
- PMIX\_COLLECTIVE\_ALGO\_REQD, [58](#)
  - Defintion, [14](#)
- PMIx\_Commit, [53](#), [57](#)
  - Defintion, [56](#)



PMIx\_Connect, 68, 72  
     Defintion, 70  
 PMIX\_CONNECT\_MAX\_RETRIES  
     Defintion, 9  
 PMIx\_Connect\_nb, 71  
     Defintion, 71  
 PMIX\_CONNECT\_RETRY\_DELAY  
     Defintion, 9  
 PMIX\_CONNECT\_SYSTEM\_FIRST  
     Defintion, 8  
 PMIX\_CONNECT\_TO\_SYSTEM  
     Defintion, 8  
 pmix\_connection\_cbfunc\_t, 118  
     Defintion, 117  
 PMIX\_COSPAWN\_APP  
     Defintion, 18  
 PMIX\_CPU\_LIST  
     Defintion, 18  
 PMIX\_CPUS\_PER\_PROC  
     Defintion, 18  
 PMIX\_CPUSET  
     Defintion, 10  
 PMIX\_CREDENTIAL  
     Defintion, 10  
 PMIX\_DAEMON\_MEMORY  
     Defintion, 13  
 pmix\_data\_array, 33  
     Defintion, 33  
 pmix\_data\_buffer\_t, 31  
     Defintion, 31  
 PMIx\_Data\_copy  
     Defintion, 93  
 PMIx\_Data\_copy\_payload  
     Defintion, 94  
 PMIx\_Data\_pack, 53  
     Defintion, 89  
 PMIx\_Data\_print  
     Defintion, 94  
 pmix\_data\_range\_t, 29, 117  
     Defintion, 29  
 PMIX\_DATA\_SCOPE  
     Defintion, 14  
 pmix\_data\_type\_t, 27  
     Defintion, 27  
 PMIx\_Data\_unpack, 53  
     Defintion, 90  
 PMIX\_DEBUG\_JOB  
     Defintion, 20  
 PMIX\_DEBUG\_STOP\_IN\_INIT  
     Defintion, 20  
 PMIX\_DEBUG\_STOP\_ON\_EXEC  
     Defintion, 20  
 PMIX\_DEBUG\_WAIT\_FOR\_NOTIFY  
     Defintion, 20  
 PMIX\_DEBUG\_WAITING\_FOR\_NOTIFY  
     Defintion, 20  
 PMIX\_DEBUGGER\_DAEMONS  
     Defintion, 18  
 PMIx\_Deregister\_event\_handler  
     Defintion, 84  
 PMIx\_Disconnect, 72, 73  
     Defintion, 71  
 PMIx\_Disconnect\_nb  
     Defintion, 72  
 PMIX\_DISPLAY\_MAP  
     Defintion, 17  
 pmix\_dmodex\_response\_fn\_t, 100  
     Defintion, 100  
 PMIX\_DSTPATH  
     Defintion, 9  
 PMIX\_EMBED\_BARRIER, 48  
     Defintion, 14  
 PMIX\_ERR\_BASE, 24  
 PMIX\_EVENT\_ACTION\_TIMEOUT  
     Defintion, 16  
 PMIX\_EVENT\_AFFECTED\_PROC  
     Defintion, 16  
 PMIX\_EVENT\_AFFECTED\_PROCS  
     Defintion, 16  
 PMIX\_EVENT\_BASE  
     Defintion, 8  
 PMIX\_EVENT\_CUSTOM\_RANGE  
     Defintion, 16  
 PMIX\_EVENT\_DO\_NOT\_CACHE  
     Defintion, 16  
 PMIX\_EVENT\_ENVIRO\_LEVEL

Defintion, [15](#)  
 PMIX\_EVENT\_HDLR\_AFTER  
     Defintion, [16](#)  
 PMIX\_EVENT\_HDLR\_APPEND  
     Defintion, [16](#)  
 PMIX\_EVENT\_HDLR\_BEFORE  
     Defintion, [16](#)  
 PMIX\_EVENT\_HDLR\_FIRST  
     Defintion, [15](#)  
 PMIX\_EVENT\_HDLR\_FIRST\_IN\_CATEGORY  
     Defintion, [15](#)  
 PMIX\_EVENT\_HDLR\_LAST  
     Defintion, [15](#)  
 PMIX\_EVENT\_HDLR\_LAST\_IN\_CATEGORY  
     Defintion, [15](#)  
 PMIX\_EVENT\_HDLR\_NAME  
     Defintion, [15](#)  
 PMIX\_EVENT\_HDLR\_PREPEND  
     Defintion, [16](#)  
 PMIX\_EVENT\_JOB\_LEVEL  
     Defintion, [15](#)  
 PMIX\_EVENT\_NO\_TERMINATION  
     Defintion, [16](#)  
 PMIX\_EVENT\_NON\_DEFAULT  
     Defintion, [16](#)  
 pmix\_event\_notification\_cbfunc\_fn\_t, [85](#)  
     Defintion, [85](#)  
 PMIX\_EVENT\_RETURN\_OBJECT  
     Defintion, [16](#)  
 PMIX\_EVENT\_SILENT\_TERMINATION  
     Defintion, [16](#)  
 PMIX\_EVENT\_TERMINATE\_JOB  
     Defintion, [16](#)  
 PMIX\_EVENT\_TERMINATE\_NODE  
     Defintion, [16](#)  
 PMIX\_EVENT\_TERMINATE\_PROC  
     Defintion, [16](#)  
 PMIX\_EVENT\_TERMINATE\_SESSION  
     Defintion, [16](#)  
 PMIX\_EVENT\_WANT\_TERMINATION  
     Defintion, [16](#)  
 pmix\_evhdr\_reg\_cbfunc\_t, [88](#)  
     Defintion, [88](#)  
 PMIx\_Fence, [58](#), [59](#), [106](#), [107](#)  
     Defintion, [57](#)  
 PMIx\_Fence\_nb, [106](#), [107](#)  
     Defintion, [58](#)  
 PMIx\_Finalize, [14](#), [23](#), [47](#), [48](#), [104](#), [105](#)  
     Defintion, [47](#)  
 PMIX\_FWD\_STDERR, [121](#)  
     Defintion, [17](#)  
 PMIX\_FWD\_STDIN, [121](#)  
     Defintion, [17](#)  
 PMIX\_FWD\_STDOUT, [121](#)  
     Defintion, [17](#)  
 PMIX\_GDS\_MODULE  
     Defintion, [10](#)  
 PMIx\_generate\_ppn  
     Defintion, [95](#)  
 PMIx\_generate\_regex  
     Defintion, [95](#)  
 PMIx\_Get, [14](#), [47](#), [54](#), [55](#), [58](#)  
     Defintion, [53](#)  
 PMIx\_Get\_nb, [58](#)  
     Defintion, [54](#)  
 PMIx\_Get\_version  
     Defintion, [46](#)  
 PMIX\_GLOBAL, [29](#)  
 PMIX\_GLOBAL\_RANK  
     Defintion, [11](#)  
 PMIX\_GRPID, [121](#)  
     Defintion, [9](#)  
 PMIx\_Heartbeat  
     Defintion, [80](#)  
 PMIX\_HOST  
     Defintion, [17](#)  
 PMIX\_HOSTFILE  
     Defintion, [17](#)  
 PMIX\_HOSTNAME  
     Defintion, [12](#)  
 PMIX\_HWLOC\_SHMEM\_ADDR  
     Defintion, [13](#)  
 PMIX\_HWLOC\_SHMEM\_FILE  
     Defintion, [13](#)  
 PMIX\_HWLOC\_SHMEM\_SIZE  
     Defintion, [13](#)

PMIX\_HWLOC\_XML\_V1  
     Defintion, [13](#)  
 PMIX\_HWLOC\_XML\_V2  
     Defintion, [13](#)  
 PMIX\_IMMEDIATE  
     Defintion, [14](#)  
 PMIX\_INDEX\_ARGV  
     Defintion, [18](#)  
 pmix\_info\_array, [35](#)  
     Defintion, [35](#)  
 pmix\_info\_cbfunc\_t, [41](#), [75](#), [77](#), [79](#), [80](#), [119](#),  
     [122](#)  
     Defintion, [41](#), [75](#)  
 pmix\_info\_directives\_t, [30](#)  
     Defintion, [30](#)  
 pmix\_info\_t, [30](#), [35](#), [47](#), [48](#), [50](#), [60](#), [62](#), [76](#),  
     [79](#), [80](#), [109](#), [115](#), [121](#)  
     Defintion, [35](#)  
 PMIx\_Init, [17](#), [45](#), [47](#), [68](#)  
     Defintion, [46](#)  
 PMIx\_Initialized  
     Defintion, [45](#)  
 PMIX\_INTERNAL, [29](#)  
 PMIX\_JOB\_CONTINUOUS  
     Defintion, [18](#)  
 PMIx\_Job\_control\_nb  
     Defintion, [78](#)  
 PMIX\_JOB\_CTRL\_CANCEL  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_EVENT  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_METHOD  
     Defintion, [22](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_SIGNAL  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_CHECKPOINT\_TIMEOUT  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_ID  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_KILL  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_PAUSE  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_PREEMPTIBLE  
     Defintion, [22](#)  
 PMIX\_JOB\_CTRL\_PROVISION  
     Defintion, [22](#)  
 PMIX\_JOB\_CTRL\_PROVISION\_IMAGE  
     Defintion, [22](#)  
 PMIX\_JOB\_CTRL\_RESTART  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_RESUME  
     Defintion, [21](#)  
 PMIX\_JOB\_CTRL\_SIGNAL  
     Defintion, [22](#)  
 PMIX\_JOB\_CTRL\_TERMINATE  
     Defintion, [22](#)  
 PMIX\_JOB\_NUM\_APPS  
     Defintion, [12](#)  
 PMIX\_JOB\_RECOVERABLE  
     Defintion, [18](#)  
 PMIX\_JOB\_SIZE  
     Defintion, [12](#)  
 PMIX\_JOB\_TERM\_STATUS  
     Defintion, [14](#)  
 PMIX\_JOBID  
     Defintion, [11](#)  
 PMIX\_LOCAL, [29](#)  
 PMIX\_LOCAL\_CPUSSETS  
     Defintion, [12](#)  
 PMIX\_LOCAL\_PEERS  
     Defintion, [12](#)  
 PMIX\_LOCAL\_PROCS  
     Defintion, [12](#)  
 PMIX\_LOCAL\_RANK  
     Defintion, [11](#)  
 PMIX\_LOCAL\_SIZE  
     Defintion, [12](#)  
 PMIX\_LOCAL\_TOPO  
     Defintion, [13](#)  
 PMIX\_LOCALITY  
     Defintion, [12](#)  
 PMIX\_LOCALITY\_STRING  
     Defintion, [13](#)

PMIX\_LOCALLDR  
     Defintion, [11](#)  
 PMIX\_LOG\_EMAIL  
     Defintion, [19](#)  
 PMIX\_LOG\_EMAIL\_ADDR  
     Defintion, [19](#)  
 PMIX\_LOG\_EMAIL\_MSG  
     Defintion, [20](#)  
 PMIX\_LOG\_EMAIL\_SUBJECT  
     Defintion, [19](#)  
 PMIX\_LOG\_MSG  
     Defintion, [19](#)  
 PMix\_Log\_nb  
     Defintion, [82](#)  
 PMIX\_LOG\_STDERR  
     Defintion, [19](#)  
 PMIX\_LOG\_STDOUT  
     Defintion, [19](#)  
 PMIX\_LOG\_SYSLOG  
     Defintion, [19](#)  
 PMix\_Lookup, [59](#), [62](#), [63](#)  
     Defintion, [61](#)  
 pmix\_lookup\_cbfunc\_t, [40](#), [110](#)  
     Defintion, [40](#)  
 PMix\_Lookup\_nb  
     Defintion, [62](#)  
 PMIX\_MAP\_BLOB  
     Defintion, [15](#)  
 PMIX\_MAPBY  
     Defintion, [17](#)  
 PMIX\_MAPPER  
     Defintion, [17](#)  
 PMIX\_MAX\_KEYLEN, [7](#)  
 PMIX\_MAX\_NSLEN, [7](#)  
 PMIX\_MAX\_PROCS  
     Defintion, [12](#)  
 PMIX\_MAX\_RESTARTS  
     Defintion, [18](#)  
 PMIX\_MERGE\_STDERR\_STDOUT  
     Defintion, [18](#)  
 PMIX\_MODEL\_LIBRARY\_NAME  
     Defintion, [9](#)  
 PMIX\_MODEL\_LIBRARY\_VERSION  
     Defintion, [9](#)  
 pmix\_modex\_cbfunc\_t, [39](#), [107](#), [108](#)  
     Defintion, [39](#)  
 pmix\_modex\_data\_t, [38](#)  
     Defintion, [38](#)  
 PMIX\_MONITOR\_APP\_CONTROL  
     Defintion, [22](#)  
 PMIX\_MONITOR\_CANCEL  
     Defintion, [22](#)  
 PMIX\_MONITOR\_FILE, [80](#)  
     Defintion, [22](#)  
 PMIX\_MONITOR\_FILE\_ACCESS  
     Defintion, [22](#)  
 PMIX\_MONITOR\_FILE\_CHECK\_TIME  
     Defintion, [22](#)  
 PMIX\_MONITOR\_FILE\_DROPS  
     Defintion, [22](#)  
 PMIX\_MONITOR\_FILE\_MODIFY  
     Defintion, [22](#)  
 PMIX\_MONITOR\_FILE\_SIZE  
     Defintion, [22](#)  
 PMIX\_MONITOR\_HEARTBEAT  
     Defintion, [22](#)  
 PMIX\_MONITOR\_HEARTBEAT\_DROPS  
     Defintion, [22](#)  
 PMIX\_MONITOR\_HEARTBEAT\_TIME  
     Defintion, [22](#)  
 PMIX\_MONITOR\_ID  
     Defintion, [22](#)  
 PMIX\_NET\_TOPO  
     Defintion, [13](#)  
 PMIX\_NO\_OVERSUBSCRIBE  
     Defintion, [18](#)  
 PMIX\_NO\_PROCS\_ON\_HEAD  
     Defintion, [18](#)  
 PMIX\_NODE\_LIST  
     Defintion, [12](#), [13](#)  
 PMIX\_NODE\_MAP  
     Defintion, [15](#)  
 PMIX\_NODE\_RANK  
     Defintion, [11](#)  
 PMIX\_NODE\_SIZE  
     Defintion, [12](#)

PMIX\_NODEID  
     Definition, [12](#)  
 PMIX\_NON\_PMI, [68](#)  
     Definition, [17](#)  
 pmix\_notification\_fn\_t, [86](#)  
     Definition, [86](#)  
 PMIX\_NOTIFY\_COMPLETION, [69](#)  
     Definition, [14](#)  
 PMIx\_Notify\_event  
     Definition, [84](#)  
 PMIX\_NPROC\_OFFSET  
     Definition, [11](#)  
 PMIX\_NSDIR  
     Definition, [11](#)  
 PMIX\_NSPACE  
     Definition, [11](#)  
 PMIX\_NUM\_NODES  
     Definition, [12](#)  
 pmix\_op\_cbfunc\_t, [40](#), [61](#), [64](#), [71](#), [73](#), [82](#),  
     [96–99](#), [102](#), [104–106](#), [109](#), [111](#),  
     [113–117](#), [122–124](#)  
     Definition, [40](#)  
 PMIX\_OPTIONAL  
     Definition, [14](#)  
 PMIX\_OUTPUT\_TO\_FILE  
     Definition, [18](#)  
 PMIX\_PARENT\_ID  
     Definition, [12](#)  
 pmix\_pdata\_t, [36](#), [62](#)  
     Definition, [36](#)  
 PMIX\_PERSISTENCE  
     Definition, [14](#)  
 pmix\_persistence\_t, [30](#)  
     Definition, [30](#)  
 PMIX\_PERSONALITY  
     Definition, [17](#)  
 PMIX\_PPR  
     Definition, [17](#)  
 PMIX\_PREFIX  
     Definition, [17](#)  
 PMIX\_PRELOAD\_BIN  
     Definition, [17](#)  
 PMIX\_PRELOAD\_FILES  
     Definition, [17](#)  
 PMIX\_PROC\_BLOB  
     Definition, [15](#)  
 PMIX\_PROC\_DATA  
     Definition, [15](#)  
 PMIX\_PROC\_FREE, [74](#)  
 pmix\_proc\_info\_t, [33](#)  
     Definition, [33](#)  
 PMIX\_PROC\_MAP  
     Definition, [15](#)  
 PMIX\_PROC\_PID  
     Definition, [11](#)  
 PMIX\_PROC\_STATE\_ABORTED, [23](#)  
 PMIX\_PROC\_STATE\_ABORTED\_BY\_SIG,  
     [23](#)  
 PMIX\_PROC\_STATE\_CALLED\_ABORT,  
     [23](#)  
 PMIX\_PROC\_STATE\_CANNOT\_RESTART,  
     [24](#)  
 PMIX\_PROC\_STATE\_COMM\_FAILED,  
     [23](#)  
 PMIX\_PROC\_STATE\_CONNECTED, [23](#)  
 PMIX\_PROC\_STATE\_ERROR, [23](#)  
 PMIX\_PROC\_STATE\_FAILED\_TO\_LAUNCH,  
     [24](#)  
 PMIX\_PROC\_STATE\_FAILED\_TO\_START,  
     [23](#)  
 PMIX\_PROC\_STATE\_KILLED\_BY\_CMD,  
     [23](#)  
 PMIX\_PROC\_STATE\_LAUNCH\_UNDERWAY,  
     [23](#)  
 PMIX\_PROC\_STATE\_MIGRATING, [23](#)  
 PMIX\_PROC\_STATE\_PREPPED, [23](#)  
 PMIX\_PROC\_STATE\_RESTART, [23](#)  
 PMIX\_PROC\_STATE\_RUNNING, [23](#)  
 PMIX\_PROC\_STATE\_STATUS  
     Definition, [14](#)  
 pmix\_proc\_state\_t, [23](#)  
     Definition, [23](#)  
 PMIX\_PROC\_STATE\_TERM\_NON\_ZERO,  
     [24](#)  
 PMIX\_PROC\_STATE\_TERM\_WO\_SYNC,  
     [23](#)

PMIX\_PROC\_STATE\_TERMINATE, [23](#)  
 PMIX\_PROC\_STATE\_TERMINATED, [23](#)  
 PMIX\_PROC\_STATE\_UNDEF, [23](#)  
 PMIX\_PROC\_STATE\_UNTERMINATED,  
     [23](#)  
 pmix\_proc\_t, [32](#), [47–49](#), [54](#), [57–59](#), [66](#),  
     [98–100](#), [104–114](#), [117](#), [119–124](#)  
     Defintion, [32](#)  
 PMIX\_PROC\_URI  
     Defintion, [12](#)  
 PMIX\_PROCDIR  
     Defintion, [11](#)  
 PMIx\_Process\_monitor\_nb, [81](#)  
     Defintion, [79](#)  
 PMIX\_PROCID  
     Defintion, [11](#)  
 PMIX\_PROGRAMMING\_MODEL  
     Defintion, [9](#)  
 PMIx\_Publish, [60](#), [61](#)  
     Defintion, [59](#)  
 PMIx\_Publish\_nb, [61](#)  
     Defintion, [60](#)  
 PMIx\_Put, [53–56](#)  
     Defintion, [52](#)  
 PMIX\_QUERY\_ALLOC\_STATUS  
     Defintion, [19](#)  
 PMIX\_QUERY\_AUTHORIZATIONS  
     Defintion, [19](#)  
 PMIX\_QUERY\_DEBUG\_SUPPORT  
     Defintion, [19](#)  
 PMIx\_Query\_info\_nb  
     Defintion, [75](#)  
 PMIX\_QUERY\_JOB\_STATUS  
     Defintion, [18](#)  
 PMIX\_QUERY\_LOCAL\_ONLY  
     Defintion, [19](#)  
 PMIX\_QUERY\_LOCAL\_PROC\_TABLE  
     Defintion, [19](#)  
 PMIX\_QUERY\_MEMORY\_USAGE  
     Defintion, [19](#)  
 PMIX\_QUERY\_NAMESPACES  
     Defintion, [18](#)  
 PMIX\_QUERY\_PROC\_TABLE  
     Defintion, [19](#)  
 PMIX\_QUERY\_QUEUE\_LIST  
     Defintion, [18](#)  
 PMIX\_QUERY\_QUEUE\_STATUS  
     Defintion, [19](#)  
 PMIX\_QUERY\_REPORT\_AVG  
     Defintion, [19](#)  
 PMIX\_QUERY\_REPORT\_MINMAX  
     Defintion, [19](#)  
 PMIX\_QUERY\_SPAWN\_SUPPORT  
     Defintion, [19](#)  
 pmix\_query\_t, [37](#), [119](#)  
     Defintion, [37](#)  
 PMIX\_RANGE  
     Defintion, [14](#)  
 PMIX\_RANK  
     Defintion, [11](#)  
 PMIX\_RANK\_LOCAL\_NODE, [7](#)  
 pmix\_rank\_t, [7](#)  
     Defintion, [7](#)  
 PMIX\_RANK\_UNDEF, [7](#)  
 PMIX\_RANK\_WILDCARD, [7](#)  
 PMIX\_RANKBY  
     Defintion, [17](#)  
 PMIx\_Register\_event\_handler  
     Defintion, [83](#)  
 PMIX\_REGISTER\_NODATA  
     Defintion, [8](#), [15](#)  
 pmix\_release\_cbfunc\_t, [38](#)  
     Defintion, [38](#)  
 PMIX\_REMOTE, [29](#)  
 PMIX\_REPORT\_BINDINGS  
     Defintion, [18](#)  
 PMIX\_REQUESTOR\_IS\_CLIENT  
     Defintion, [9](#)  
 PMIX\_REQUESTOR\_IS\_TOOL  
     Defintion, [9](#)  
 PMIx\_Resolve\_nodes  
     Defintion, [74](#)  
 PMIx\_Resolve\_peers  
     Defintion, [73](#)  
 PMIX\_RM\_NAME  
     Defintion, [20](#)

PMIX\_RM\_VERSION  
     Definition, [20](#)  
 pmix\_scope\_t, [28](#), [53](#)  
     Definition, [28](#)  
 PMIX\_SEND\_HEARTBEAT  
     Definition, [22](#)  
 pmix\_server\_abort\_fn\_t  
     Definition, [105](#)  
 pmix\_server\_alloc\_fn\_t  
     Definition, [122](#)  
 pmix\_server\_client\_connected\_fn\_t  
     Definition, [104](#)  
 pmix\_server\_client\_finalized\_fn\_t  
     Definition, [104](#)  
 pmix\_server\_connect\_fn\_t  
     Definition, [112](#)  
 PMIx\_server\_deregister\_client  
     Definition, [99](#)  
 pmix\_server\_deregister\_events\_fn\_t  
     Definition, [115](#)  
 PMIx\_server\_deregister\_nspace, [99](#)  
     Definition, [97](#)  
 pmix\_server\_disconnect\_fn\_t  
     Definition, [113](#)  
 pmix\_server\_dmodex\_req\_fn\_t  
     Definition, [107](#)  
 PMIx\_server\_dmodex\_request  
     Definition, [100](#)  
 PMIX\_SERVER\_ENABLE\_MONITORING  
     Definition, [8](#)  
 pmix\_server\_fencefn\_fn\_t, [107](#)  
     Definition, [106](#)  
 PMIx\_server\_finalize  
     Definition, [50](#)  
 PMIX\_SERVER\_HOSTNAME  
     Definition, [8](#)  
 PMIx\_server\_init, [45](#), [102](#), [103](#)  
     Definition, [50](#)  
 pmix\_server\_job\_control\_fn\_t  
     Definition, [123](#)  
 pmix\_server\_listener\_fn\_t  
     Definition, [118](#)  
 pmix\_server\_log\_fn\_t  
     Definition, [121](#)  
 pmix\_server\_lookup\_fn\_t  
     Definition, [109](#)  
 pmix\_server\_module\_t, [50](#), [102](#)  
     Definition, [103](#)  
 pmix\_server\_monitor\_fn\_t  
     Definition, [123](#)  
 pmix\_server\_notify\_event\_fn\_t  
     Definition, [116](#)  
 PMIX\_SERVER\_NSSPACE  
     Definition, [8](#)  
 PMIX\_SERVER\_PIDINFO  
     Definition, [8](#)  
 pmix\_server\_publish\_fn\_t  
     Definition, [108](#)  
 pmix\_server\_query\_fn\_t  
     Definition, [118](#)  
 PMIX\_SERVER\_RANK  
     Definition, [8](#)  
 PMIx\_server\_register\_client  
     Definition, [98](#)  
 pmix\_server\_register\_events\_fn\_t  
     Definition, [114](#)  
 PMIx\_server\_register\_nspace  
     Definition, [96](#)  
 PMIX\_SERVER\_REMOTE\_CONNECTIONS  
     Definition, [8](#)  
 PMIx\_server\_setup\_application  
     Definition, [101](#)  
 PMIx\_server\_setup\_fork  
     Definition, [99](#)  
 PMIx\_server\_setup\_local\_support  
     Definition, [102](#)  
 pmix\_server\_spawn\_fn\_t  
     Definition, [111](#)  
 PMIX\_SERVER\_SYSTEM\_SUPPORT  
     Definition, [8](#)  
 PMIX\_SERVER\_TMPDIR  
     Definition, [8](#)  
 pmix\_server\_tool\_connection\_fn\_t  
     Definition, [120](#)  
 PMIX\_SERVER\_TOOL\_SUPPORT  
     Definition, [8](#)

pmix\_server\_unpublish\_fn\_t  
     Defintion, [110](#)  
 PMIX\_SERVER\_URI  
     Defintion, [8](#)  
 PMIX\_SESSION\_ID  
     Defintion, [12](#)  
 PMIX\_SET\_ENVAR  
     Defintion, [20](#)  
 PMIX\_SET\_SESSION\_CWD  
     Defintion, [18](#)  
 pmix\_setup\_application\_cbfunc\_t, [101](#)  
     Defintion, [101](#)  
 PMIX\_SINGLE\_LISTENER  
     Defintion, [9](#)  
 PMIX\_SOCKET\_MODE  
     Defintion, [9](#)  
 PMIx\_Spawn, [10](#), [69](#)  
     Defintion, [67](#)  
 pmix\_spawn\_cbfunc\_t, [39](#), [69](#), [112](#)  
     Defintion, [39](#)  
 PMIx\_Spawn\_nb  
     Defintion, [69](#)  
 PMIX\_SPAWNED  
     Defintion, [10](#)  
 pmix\_status\_t, [24](#), [115–117](#), [120](#)  
     Defintion, [24](#)  
 PMIX\_STDIN\_TGT  
     Defintion, [17](#)  
 PMIx\_Store\_internal  
     Defintion, [55](#)  
 PMIX\_SYSTEM\_TMPDIR  
     Defintion, [8](#)  
 PMIX\_TAG\_OUTPUT  
     Defintion, [18](#)  
 PMIX\_TCP\_DISABLE\_IPV4  
     Defintion, [10](#)  
 PMIX\_TCP\_DISABLE\_IPV6  
     Defintion, [10](#)  
 PMIX\_TCP\_IF\_EXCLUDE  
     Defintion, [10](#)  
 PMIX\_TCP\_IF\_INCLUDE  
     Defintion, [10](#)  
 PMIX\_TCP\_IPV4\_PORT  
     Defintion, [10](#)  
 PMIX\_TCP\_IPV6\_PORT  
     Defintion, [10](#)  
 PMIX\_TCP\_REPORT\_URI  
     Defintion, [10](#)  
 PMIX\_TCP\_URI  
     Defintion, [10](#)  
 PMIX\_TDIR\_RMCLEAN  
     Defintion, [11](#)  
 PMIX\_THREADING\_MODEL  
     Defintion, [9](#)  
 PMIX\_TIME\_REMAINING  
     Defintion, [19](#)  
 PMIX\_TIMEOUT, [54](#), [58](#), [68](#)  
     Defintion, [14](#)  
 PMIX\_TIMESTAMP\_OUTPUT  
     Defintion, [18](#)  
 PMIX\_TMPDIR  
     Defintion, [11](#)  
 pmix\_tool\_connection\_cbfunc\_t, [120](#)  
     Defintion, [119](#)  
 PMIX\_TOOL\_DO\_NOT\_CONNECT  
     Defintion, [9](#)  
 PMIx\_tool\_finalize  
     Defintion, [49](#)  
 PMIx\_tool\_init, [45](#), [49](#)  
     Defintion, [48](#)  
 PMIX\_TOOL\_NSSPACE  
     Defintion, [8](#)  
 PMIX\_TOOL\_RANK  
     Defintion, [8](#)  
 PMIX\_TOPOLOGY  
     Defintion, [13](#)  
 PMIX\_TOPOLOGY\_SIGNATURE  
     Defintion, [13](#)  
 PMIX\_UNIV\_SIZE  
     Defintion, [12](#)  
 PMIx\_Unpublish, [64](#), [65](#)  
     Defintion, [63](#)  
 PMIx\_Unpublish\_nb  
     Defintion, [64](#)  
 PMIX\_UNSET\_ENVAR  
     Defintion, [20](#)



PMIX\_USERID, [121](#)

Defintion, [9](#)

PMIX\_USOCK\_DISABLE

Defintion, [9](#)

pmix\_value\_cbfunc\_t, [41](#)

Defintion, [41](#)

pmix\_value\_t, [34](#), [52](#), [53](#)

Defintion, [34](#)

PMIX\_VERSION\_INFO

Defintion, [9](#)

PMIX\_WAIT

Defintion, [14](#)

PMIX\_WDIR

Defintion, [17](#)